The PERSEUS lib: Open Source Library for TRANSEC and COMSEC Security

Eric Filiol & Eddy Deligne
{filiol,deligne}@esiea.fr

ESIEA - Laval
Operational Cryptology and Virology Lab $(C+V)^O$

iAWACS 2010

Introduction

- Many of the computer attacks rely on gathering information on the future targets.
  - Gain technical and/or social intelligence.
  - Steal personal/confidential data.
- This initial step generally relies on listening unprotected data flows (IRC or HTTP flows).
- It is far more simple and far more productive to eavesdrop open (unprotected) trafics than any other more agressive techniques (intrusion, malware attacks...).
- It is above all passive thus transparent.

Introduction (2)

- This issue goes beyond computer attacks and also relates to
  - (right to) privacy concerns,
  - misuse of attack techniques by private intelligence companies,
  - abusive gathering of data or citizens' surveillance for commercial purposes.
- The question is : how to hinder any misuse of eavesdropping techniques while
  - preserving the technical ability of nation states for REAL interior security purposes,
  - being compliant with any existing national laws or regulations (very important issue with respect to transnational data streams),
  - AND preserving the natural right for privacy.
- The solution is PERSEUS technology.

Introduction (3)

Obvious solution : use encryption. But can be slow and there might be legal national regulations ! Moreover it hinders the legitimate action of Nation States (against terrorism, child pornography...).

- To solve all possible constraints (legal + technical), we need :
  - a system that can be broken ONLY if you devote a huge computer powwer and if you have enough time ;
  - otherwise you cannot break it in a reasonable amount of time !
- This naturally limits the number of eavesdropping attempts.
- The solution : replace crypto techniques with noisy coding techniques !
- Typical example : a Western journalist in China.
  - He writes and sends its paper from China (about Human rights).
  - Chinese can break the message but the journalist has left China already.

Encryption vs Noisy Coding

- "Legal" definition of whether it is cryptography or not, relates in a way or another directly to the following probability

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

  where c_t, m_t are the ciphertext and plaintext bits respectively and where e_t can be defined as the noise bit produced by the key and the cryptosystem (at time instant t).
  - If $P[e_t = 1] = \frac{1}{2} + \epsilon$ (with ϵ close to 0) then it is cryptography.
  - Otherwise (ϵ significantly different from 0) it is coding theory.
- So the solution is to consider a computationally hard (for the attacker) problem from the coding theory.

Encryption vs Noisy Coding

- Why use noisy encoded data instead of encrypted data ?
  - Encrypted data exhibit a high entropy profile. It is then easy to detect encrypted data.
  - Noisy encoded data exhibit a low entropy profile. The statistical profile is close to that of normal communications (for example cell phone communications).
- This particular statistical profile enabled to bypass any detection by entropy test or any other statistical detection while crypto does not.
  - Some sort of TRANSEC aspect (hide noisy encoded data among other encoded data).
  - Can be applied to bypass any kind of detection based on entropy (malware detection, firewall filtering. . .).

PERSEUS Technology

- Open technology based on punctured convolutional codes with deterministic controlled noise.
  - PERSEUS module : Firefox module to protect HTTP streams (GET and POST methods)..
  - First step for practical validation.
  - Presented at Hack.lu 2009.
  - A lot of interest and feedback.
  - Many requests for other protocols (P2P, torrent, email...).
- Need for a free, open source library for use/modification/analysis by anyone.
- Official webpage `http://www.esiea-recherche.eu/perseus_en.html`
- Google repository `http://code.google.com/p/perseus-firefox/`
- Mozilla repository `http://www.mozdev.org/source/browse/perseus`

Plan

What are Error-Correcting Codes ?

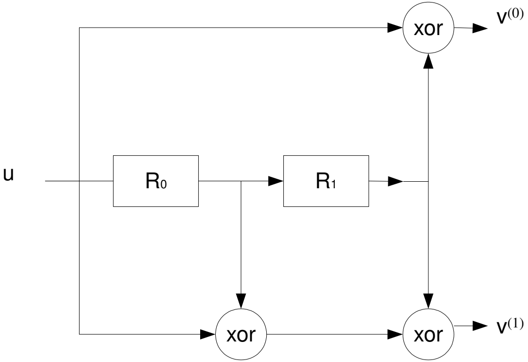
- Mathematical tool introduced by C. E. Shannon (1948) to correct the effect of "natural" noise on a communication channel (Shannon's 2nd Theorem).

$$m_t \rightarrow m_t \oplus e_t = \hat{m}_t \rightarrow \text{ decoding } m'_t$$

  - Consists in adding a limited amount of redundancy bits to the message in order to recover from the noise.
  - There always exist codes such that $P[m'_t = m_t]$ tends towards 0.
- Different issues :
  - Efficiency of the decoding (legitimate users).
  - Reconcontruction of unknown encoders (attackers'concern).
- Large variety of codes. We consider one the fastest family for stream coding : convolutional codes.

Convolutional Code

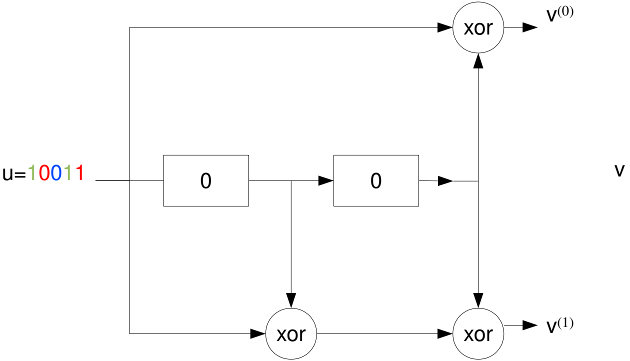
Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

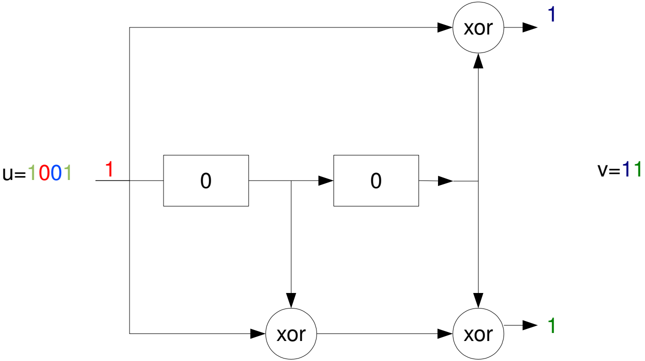
A message $u = 10011$



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

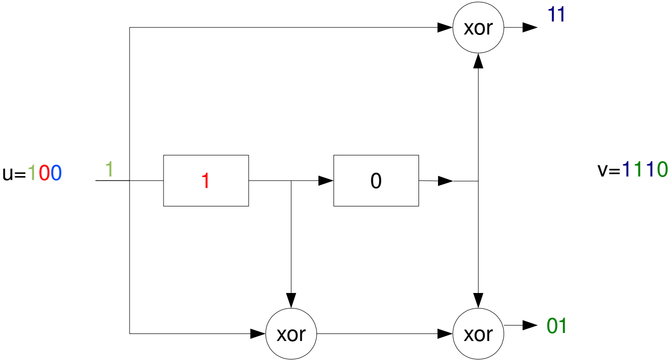
A message $u = 10011$



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

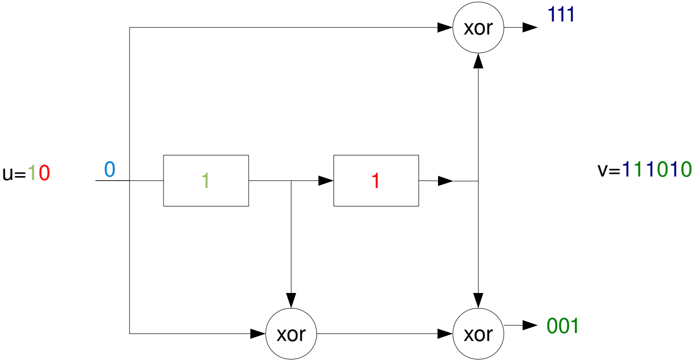
A message $u = 10011$



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

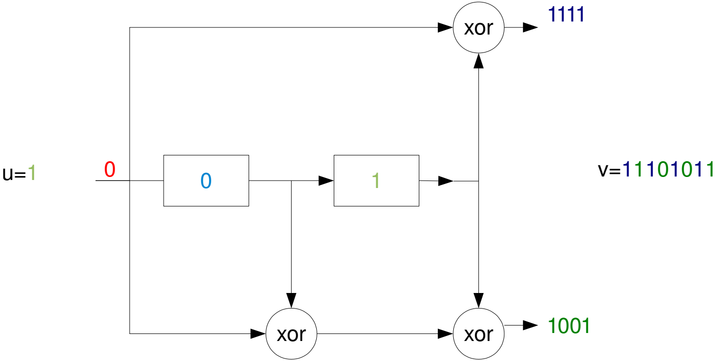
A message $u = 10011$



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

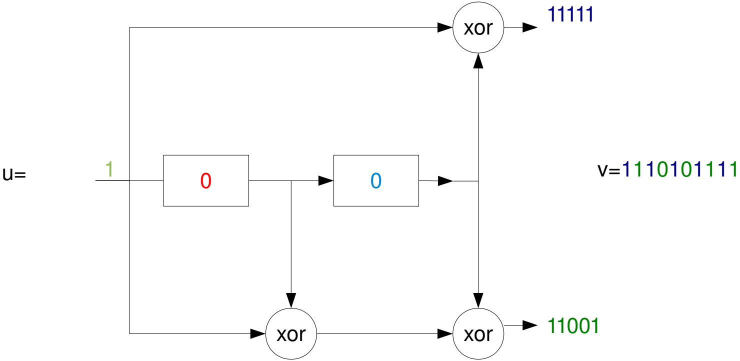
A message $u = 10011$



Convolutional Code

Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.

A message $u = 10011$



Presentation

A convolutional code is defined by

- a rate : $\frac{k}{n}$
- a memory size (or constraint length) $K = M + 1$.

Notation

(n, k, K)-convolutional code

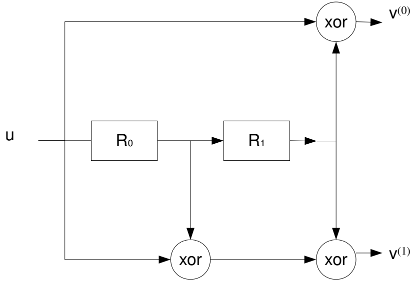
Alternative View

A Convolutional Code

k registers with n polynomials operating on each register.

$n \times k$ polynomials for a (n, k, K)-convolutional code.

The degree of polynomials will be equal to $K - 1$.



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

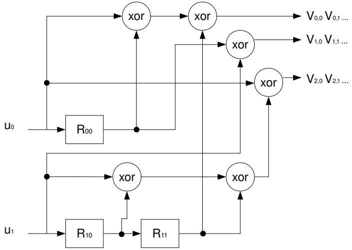
$v_0 : 1 + x^2$

$v_1 : 1 + x + x^2$

Alternative View

A Convolutional Code

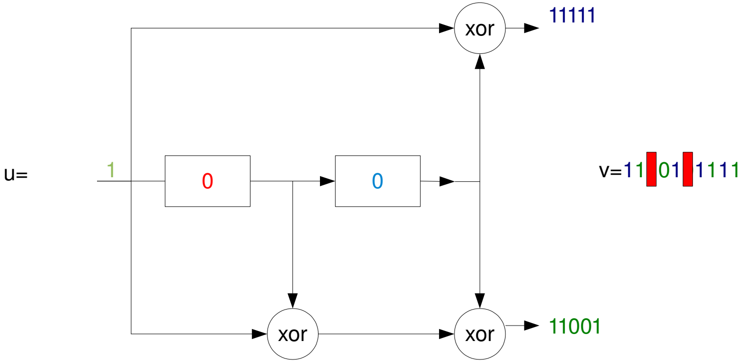
k registers with n polynomials operating on each register.
$n \times k$ polynomials for a (n, k, K)-convolutional code.
The degree of polynomials will be equal to $K - 1$.



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

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

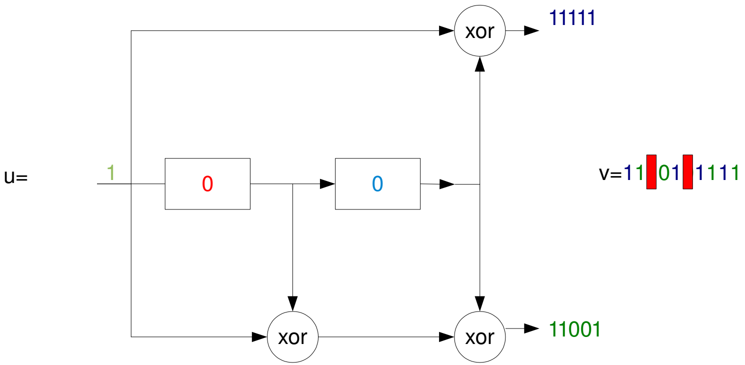
Puncturing



Puncturing pattern

P : a $J \times n$ matrix of weight I.

Puncturing



Puncturing pattern

P : a $J \times n$ matrix of weight I.

Example

Let P be the puncturing pattern given by :

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

and let v be the $(2,1,3)$ encoder output sequence :

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

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

Why puncturing ?

1. Save bandwidth (reduce the redundancy added).
2. Produce an equivalent (non punctured) convolutional code which is stronger for our purposes (see further).

Why puncturing?

1. Save bandwidth (reduce the redundancy added).
2. Produce an equivalent (non punctured) convolutional code which is stronger for our purposes (see further).

Equivalent (non punctured) convolutional code

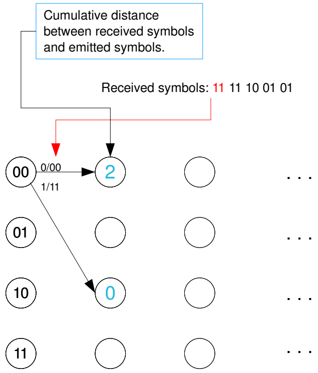
A (n, k, K)-convolutional code and a $J \times n$ puncturing matrix P of weight I.

$\Rightarrow (I, kJ, K)$-convolutional code

Viterbi Algorithm

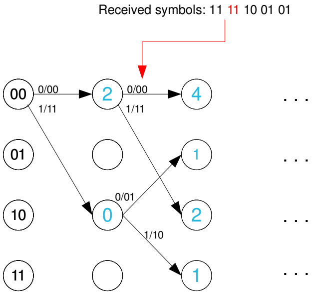
Viterbi Algorithm

Lattice construction



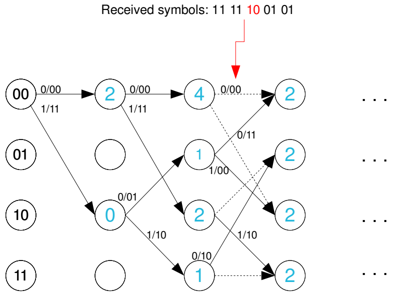
Viterbi Algorithm

Lattice construction



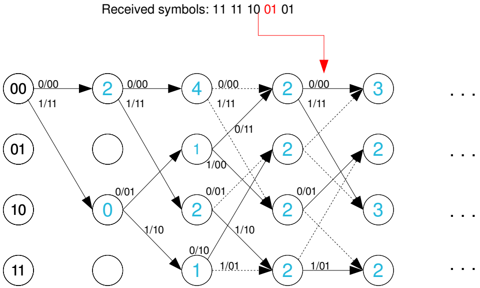
Viterbi Algorithm

Lattice construction



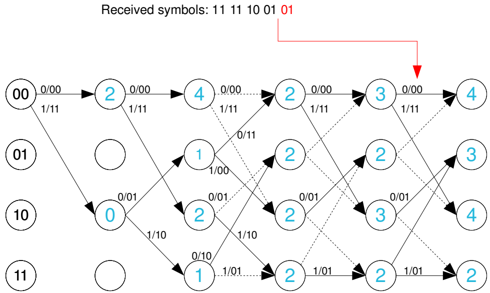
Viterbi Algorithm

Lattice construction



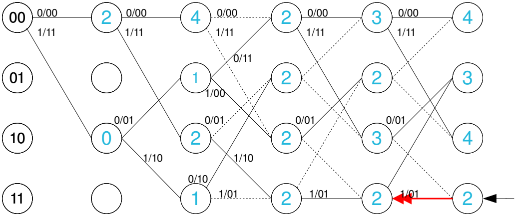
Viterbi Algorithm

Lattice construction



Viterbi Algorithm

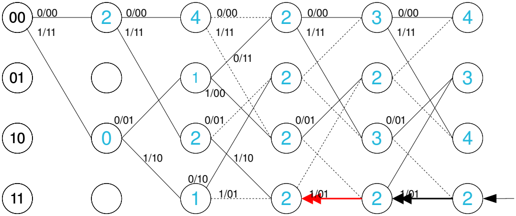
Backtracking



Decoded symbols: 1

Viterbi Algorithm

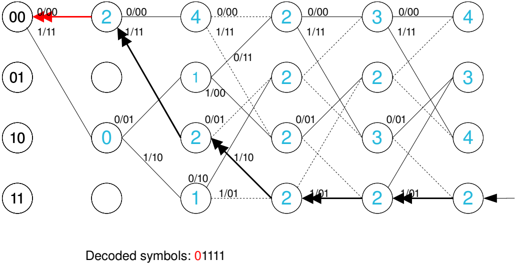
Backtracking



Decoded symbols: 11

Viterbi Algorithm

Backtracking



- Decoding has exponential complexity in K.
- When dealing with puncturing, replace removed bits with zeroes.

Convolutional Code Reconstruction (Filiol 1997 - Barbier 2007)

Aim : recovering all the parameters of an unknow encoder from the encoded data only, to be able to decode data afterwards.

Puncturing effect

Let us consider a (n, k, K)-convolutional code and a $J \times n$ puncturing matrix P of weight I :

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

Reconstruction has the following complexity

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

α : grows exponentialy with p, the noise probability

Convolutional Code Reconstruction (Filiol 1997 - Barbier 2007)

Aim : recovering all the parameters of an unknow encoder from the encoded data only, to be able to decode data afterwards.

Puncturing effect

Let us consider a (n, k, K)-convolutional code and a $J \times n$ puncturing matrix P of weight I :

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

Reconstruction has the following complexity

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

α : grows exponentialy with p, the noise probability

The Noise Impact

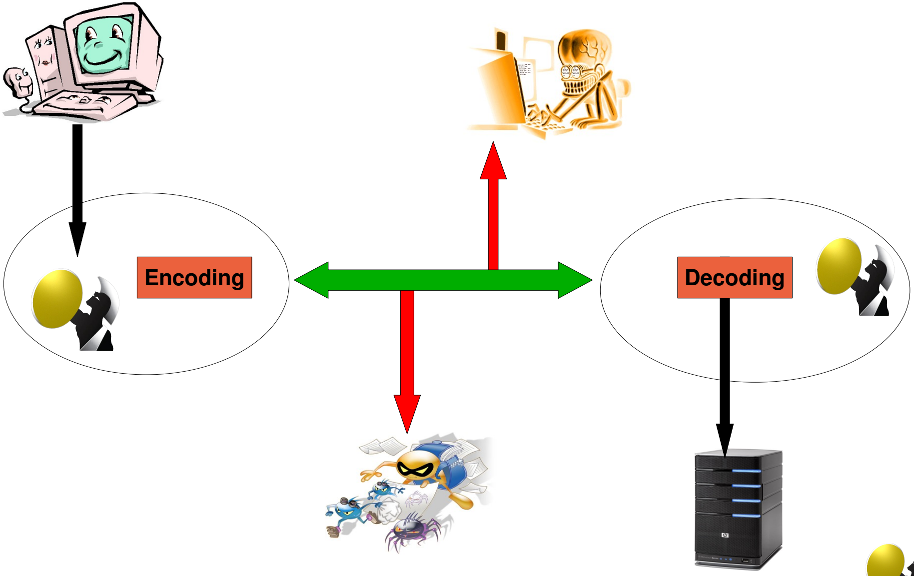
The probability to successfully reconstruct a code exponentially decreases with p. If $p > 10\%$ $\Rightarrow$ online reconstruction is impossible ; offline reconstruction is computationally very hard.

Encoder	Reconstruction Time ($p = 10^{-2}$)	Reconstruction Time ($p = 2.10^{-2}$)
(4, 3, 8)	7 min 12 sec	Failure
(4, 3, 9)	6 min 16 sec	Failure

TABLE: Examples of reconstruction times (Pentium IV 2.0 Ghz) for two levels of noise

Plan

1. Introduction
2. Punctured Convolutional Codes
   - Convolutional Codes
   - Puncturing
   - Convolutional Decoding
   - Convolutional Code Reconstruction
3. PERSEUS Description
   - General Description
   - Noise Generation
   - Parameter Management
4. PERSEUS Library
   - Implementation
5. Conclusion


Encoding
Decoding

General Principle

- The attacker (botnet client, attacker, eavesdropper...) must face a computationally untractable problem.
- For every different communication, a random encoder is used to encode the HTTP traffic.
- To make the reconstruction computationally untractable we add a secret-based deterministic noise.
- The legitimate knows the exact noise bit indices and can remove the noise to perform a noiseless sequence decoding.

Added Deterministic Noise

To greatly increase the security of data stream as well as the encoder used, noise must be added to the encoded data before transmission.

Problem

Viterbi decoding is easy as long as $p < 5\%$
Encoder reconstruction is impossible as long as $p > 10\%$ (inline mode)

Added Deterministic Noise

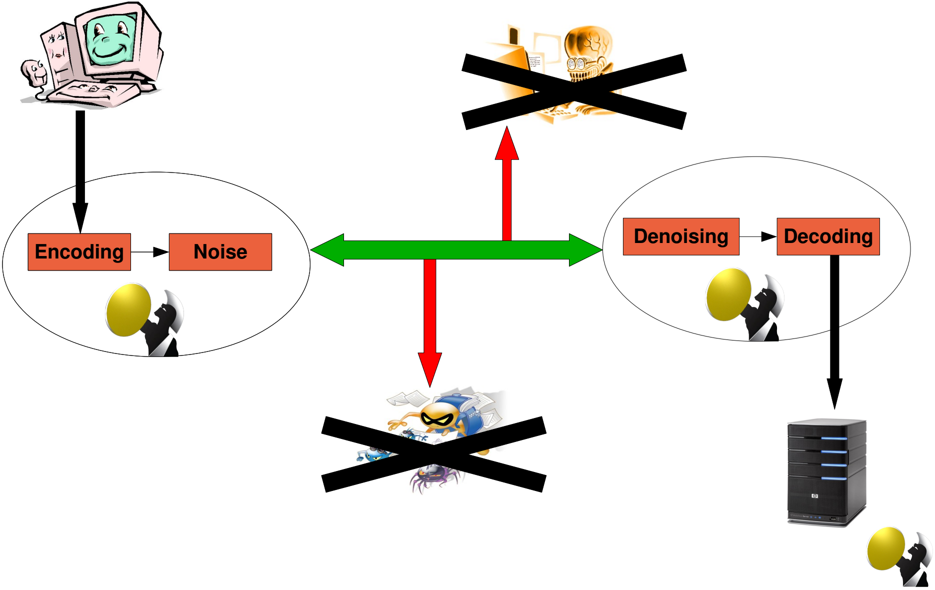
To greatly increase the security of data stream as well as the encoder used, noise must be added to the encoded data before transmission.

Problem

Viterbi decoding is easy as long as $p < 5\%$
Encoder reconstruction is impossible as long as $p > 10\%$ (inline mode)

Solution

We add a deterministic noise with $p \approx 30\%$


Encoding
Noise
Denoising
Decoding

What we need

1. A noise with probability around 30%.
2. A deterministic noise that must be controlled (except for the attackers).

Principle

The noise is produced by a biased pseudo-random generator (stream cipher).
The filtering function is underbalanced (30% of 1s).
Secret initial state of 64 bits.

What we need

1. A noise with probability around 30%.
2. A deterministic noise that must be controlled (except for the attackers).

Principle

The noise is produced by a biased pseudo-random generator (stream cipher).
The filtering function is underbalanced (30% of 1s).
Secret initial state of 64 bits.

What are the parameters

For every new session :

1. $2 < n \leq 12$
2. $1 < k < n$
3. $15 < N \leq 30$
4. $n \times k$ polynomials of degree $N - 1$
5. A $J \times n$-matrix P and of weight $(n \times J) - (J - 1)$
6. X_0 a 64-bit (secret) integer

What are the parameters

For every new session :

1. $2 < n \leq 12$
2. $1 < k < n$
3. $15 < N \leq 30$
4. $n \times k$ polynomials of degree $N - 1$
5. A $J \times n$-matrix P and of weight $(n \times J) - (J - 1)$
6. X_0 a 64-bit (secret) integer

What are the parameters

For every new session :

1. $2 < n \leq 12$
2. $1 < k < n$
3. $15 < N \leq 30$
4. $n \times k$ polynomials of degree $N - 1$
5. A $J \times n$-matrix P and of weight $(n \times J) - (J - 1)$
6. X_0 a 64-bit (secret) integer

Sending to the server

Parameters are sent through an initial HTTPS session. The parameter lifetime is limited (either the session has limited length or it is reinitialized with new parameters).

Implementation

PERSEUS library

- Written in optimized C.
- Under the triple MPL/GPL/LGPL licence.
- Source, documentation available soon (mid-june 2010) on `http://www.esiea-recherche.eu/perseus_en.html`
- Feedback, bug report and contributions requested (contact authors).

Library Structure

Very simple structure : 5 main procedures.

- 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) ;

TRANSEC Aspect

Any PERSEUS-protected stream exhibits a statistical profile that is far from any encrypted data.

- On a 175-letter text of English
  - The text has entropy 3.90.
  - A encrypted version (simple transpositions and substitutions) of that text has entropy equal to 5.5.
  - With PERSEUS the entropy is about 2.57.
- Since encoder is changing very frequently, the entropy profile is itself polymorphic but remain always equivalent to plaintext data.

Conclusion

- PERSEUS technology gives an elegant answer to a critical issue :
  - How to protect against HTTP traffic eavesdropping by botnet clients...
  - ... and abuses against citizens'privacy fundamental rights...
  - ... without crypto...
  - while preserving TRUE, LEGITIMATE ability for national security enforcement (internal and external) ?
- PERSEUS replaces cryptography by coding theory techniques.
- Only agencies with a strong enough computer power can eavesdrop traffic in an acceptable amount of time... provided that there are not too many communications to deal with at the same time.
- Interesting issue : security is not a technical matter only. Legal aspects are a critical part.

Questions ?