PERSEUS: A Firefox Plug-in to Fight Botnets ... and Many Other Things

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

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

Hack.lu 2009

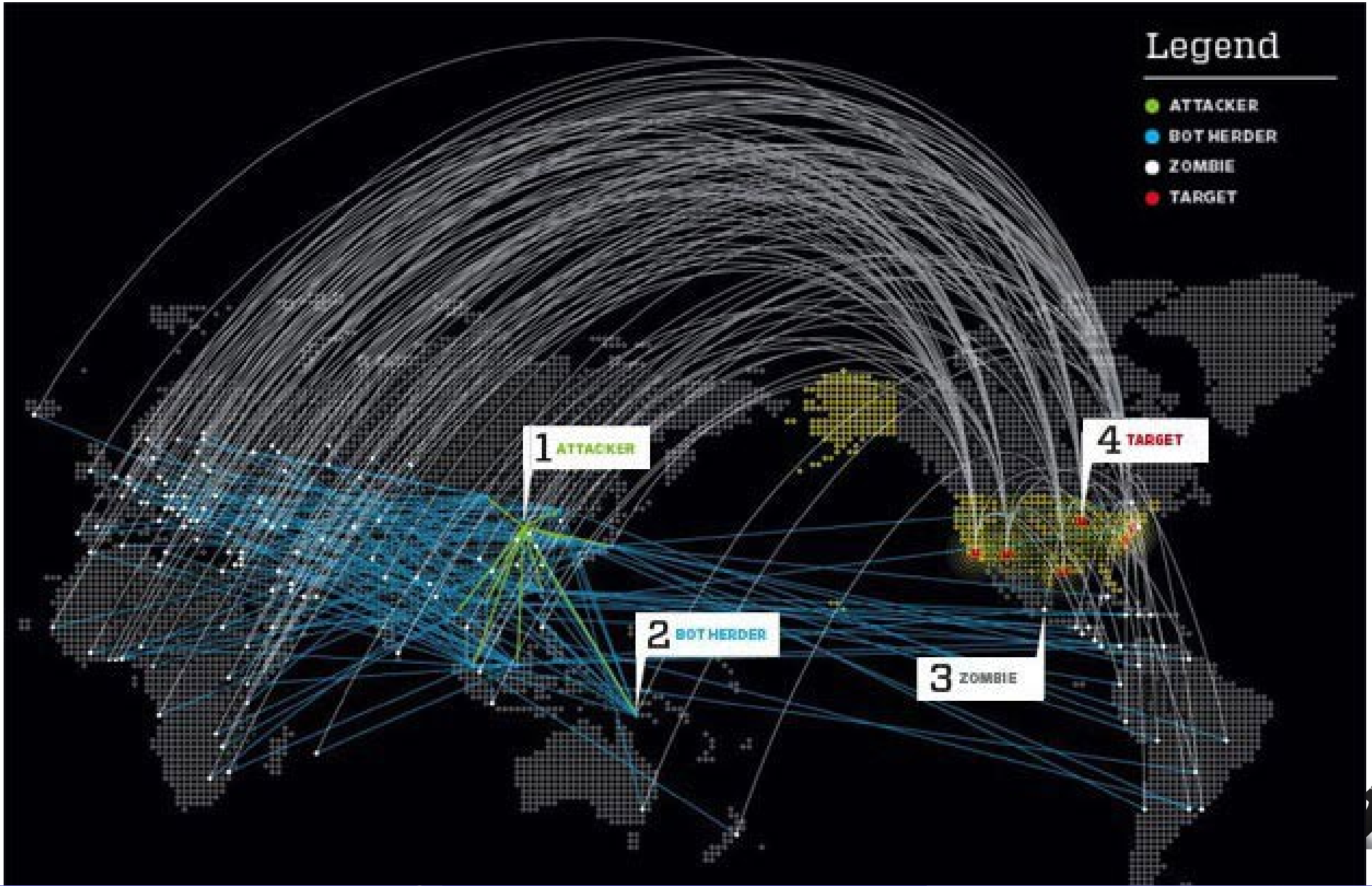
Introduction

- 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.

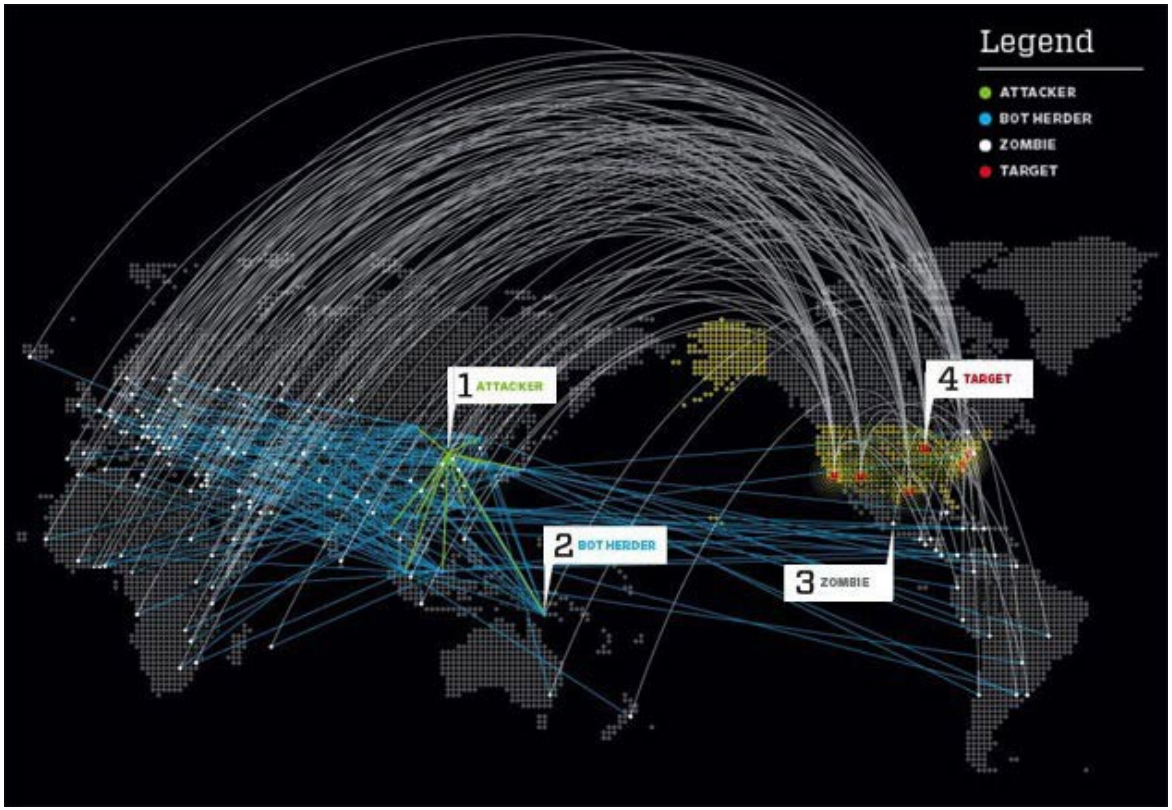
Introduction (2)

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

Introduction : The Botnet Case



Introduction : The Botnet Case



Obvious solution : use encryption. But can be slow and there might be legal national regulations !

Introduction : The Botnet Case

To solve all possible constraints (legal + technical)

How to protect our personal data without infringing any national crypto regulations and preserving the true needs of national security ?

Introduction : The Botnet Case

To solve all possible constraints (legal + technical)

How to protect our personal data without infringing any national crypto regulations and preserving the true needs of national security ?

Solution

Replace crypto techniques with noisy coding techniques !

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 controlled noise.
  - PERSEUS module : Firefox module to protect HTTP streams.
  - First step for practical validation.
- Can be used to protect any kind of protocols (FTP, mail . . .).

Plan

What are Error-Correcting Codes ?

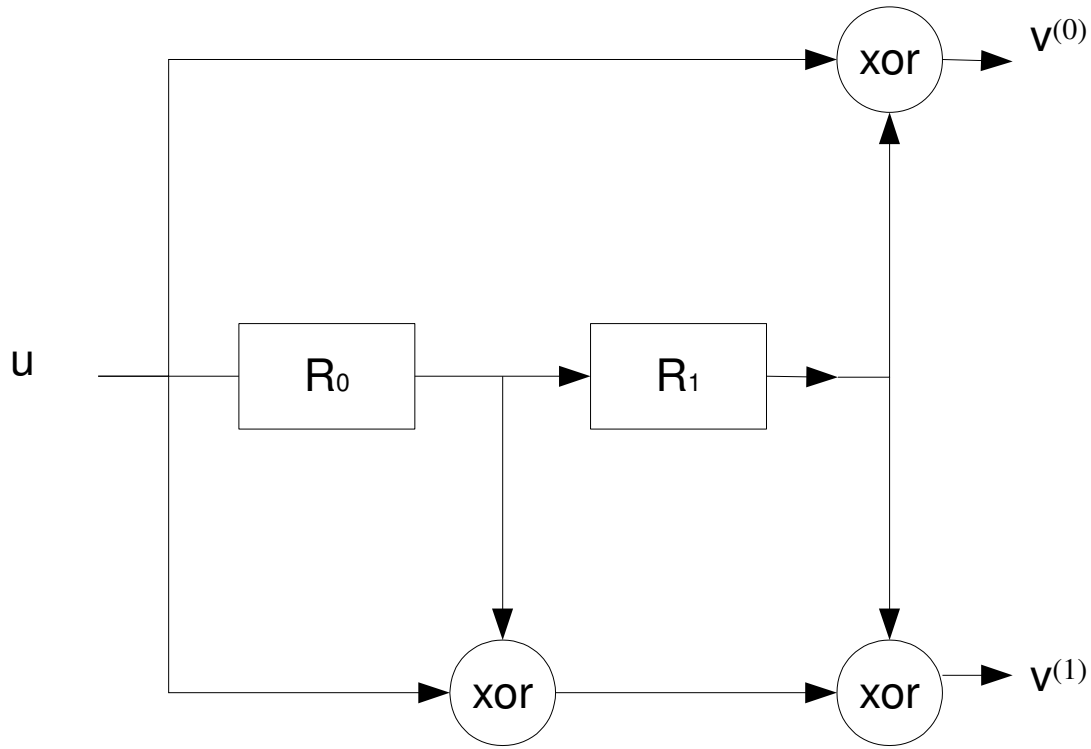
- Mathematical tool introduced by C. E. Shannon (1948) to correct the effect of "natural" noise on a communication channel (Shannon's 2nd Theorem).

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

  - 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

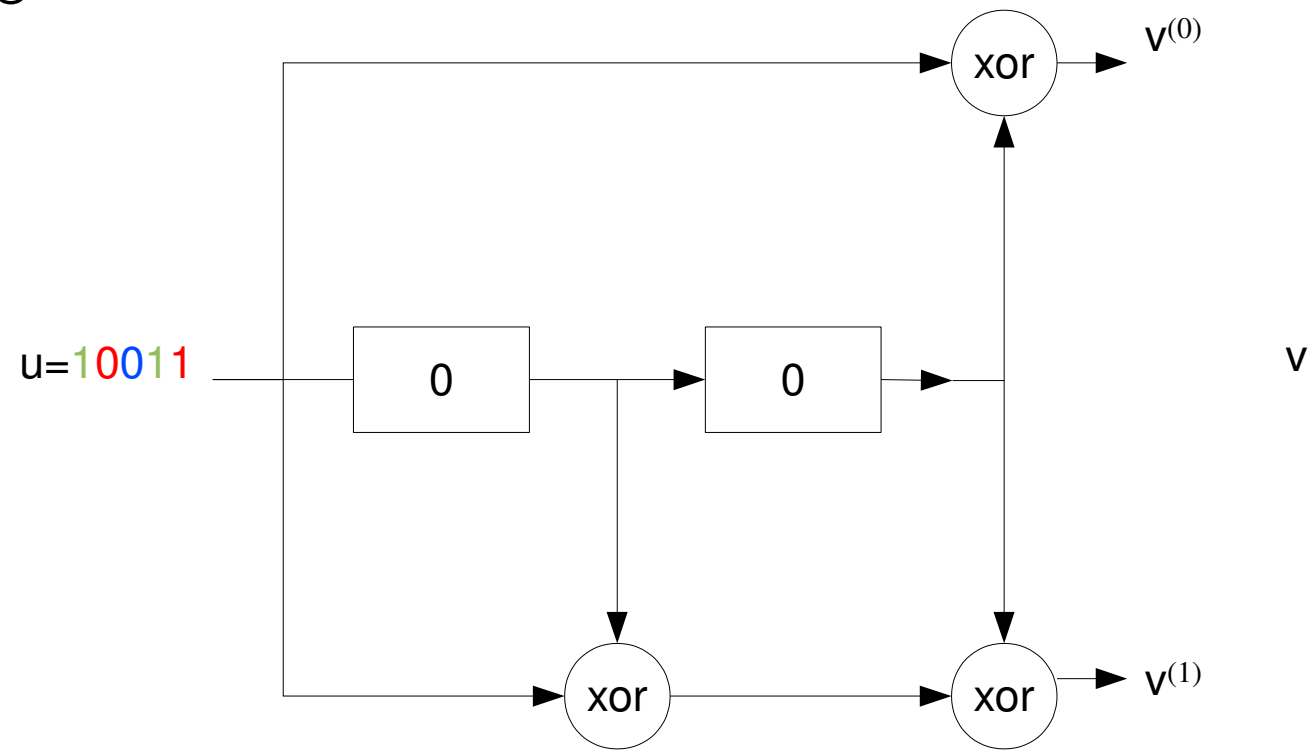
Let us consider a convolutional code $\mathcal{C}$ of rate $\frac{1}{2}$ with a memory size of $M = 2$.



Convolutional Code

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

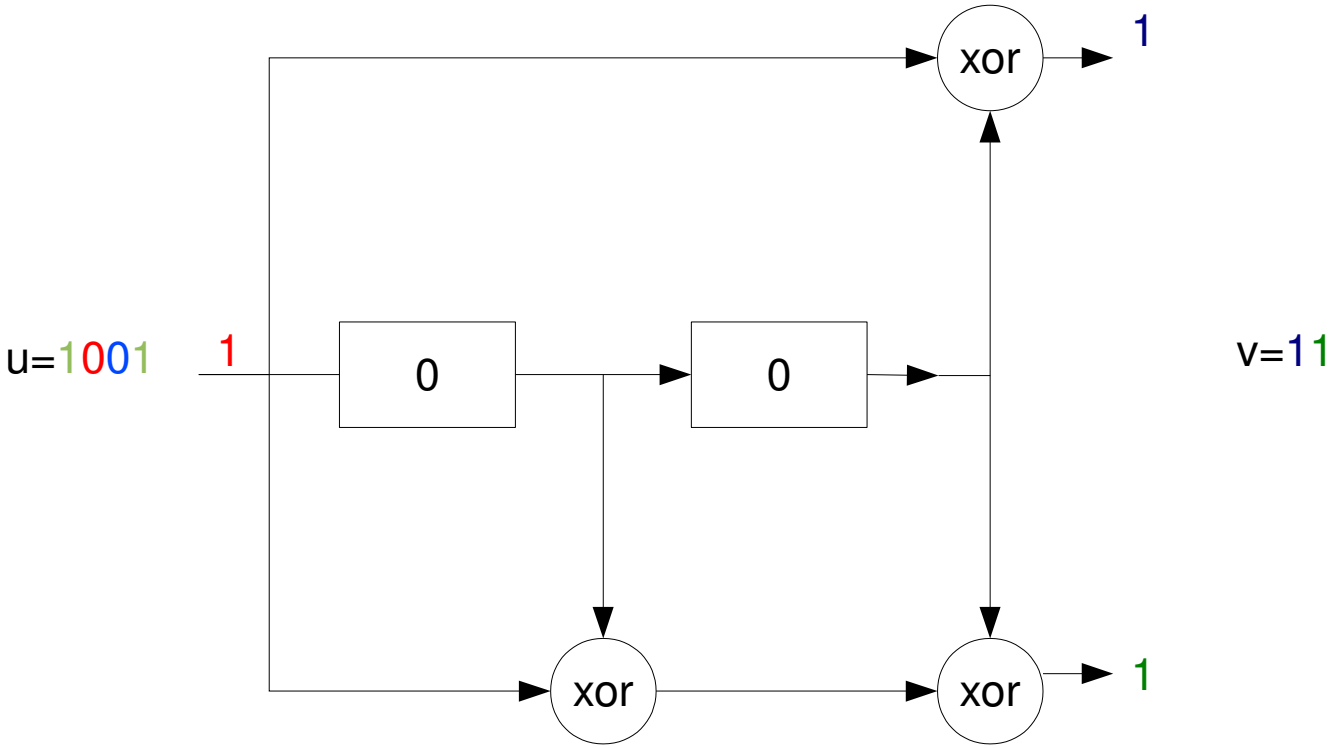
A message $u = 10011$



Convolutional Code

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

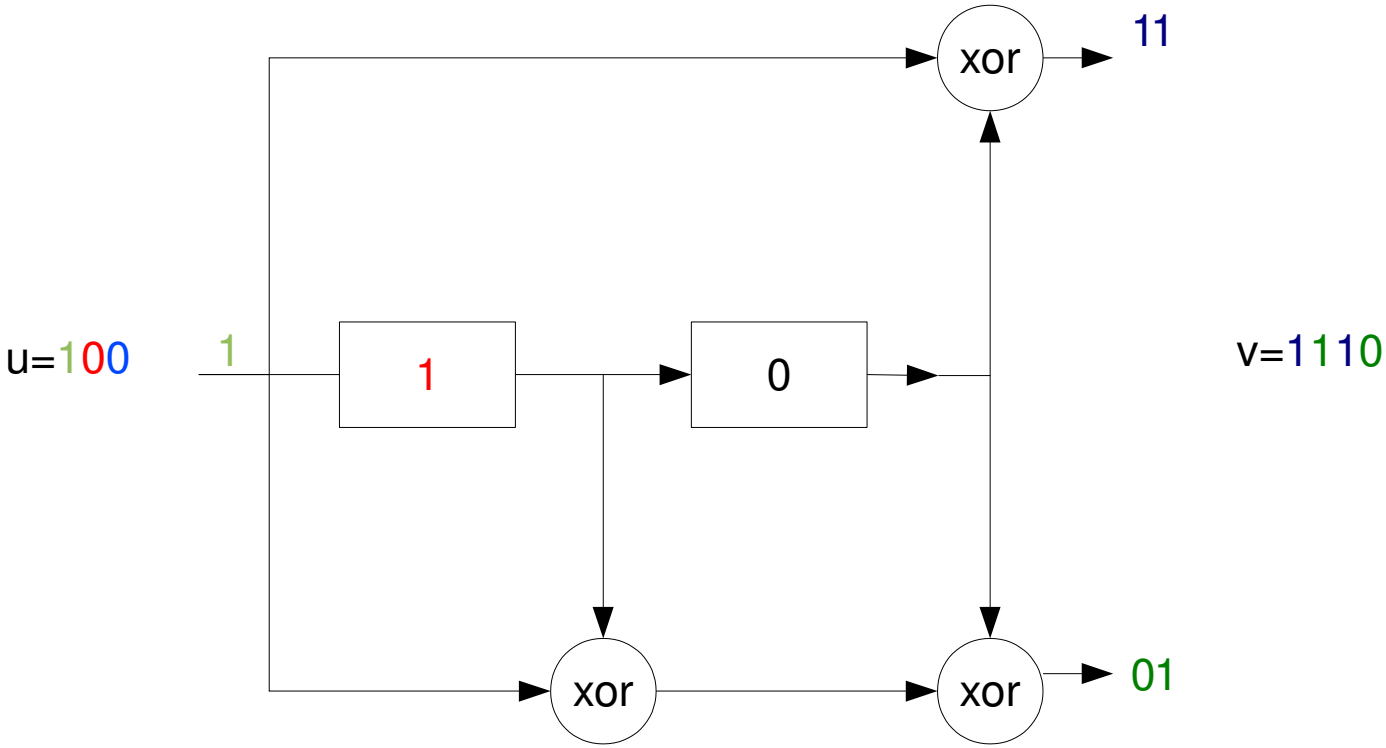
A message $u = 10011$



Convolutional Code

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

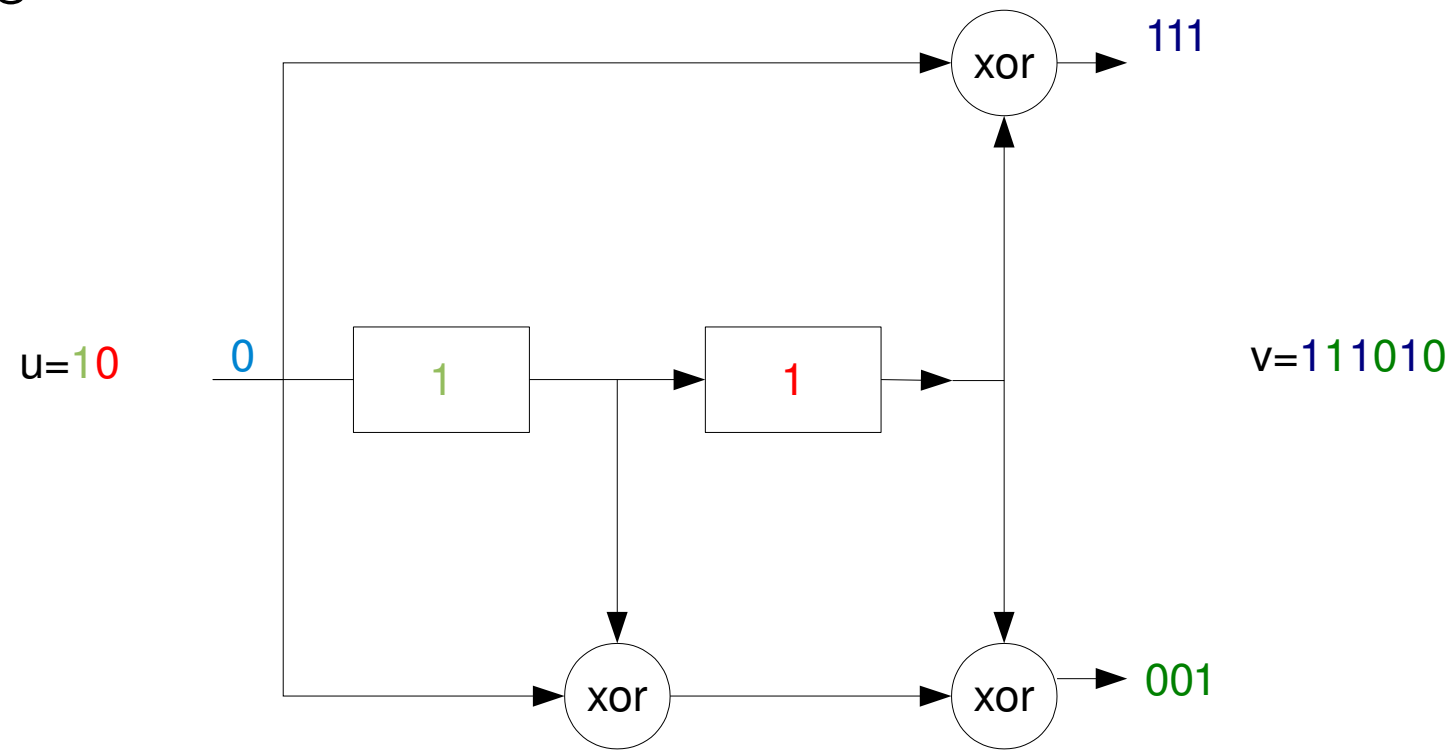
A message $u = 10011$



Convolutional Code

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

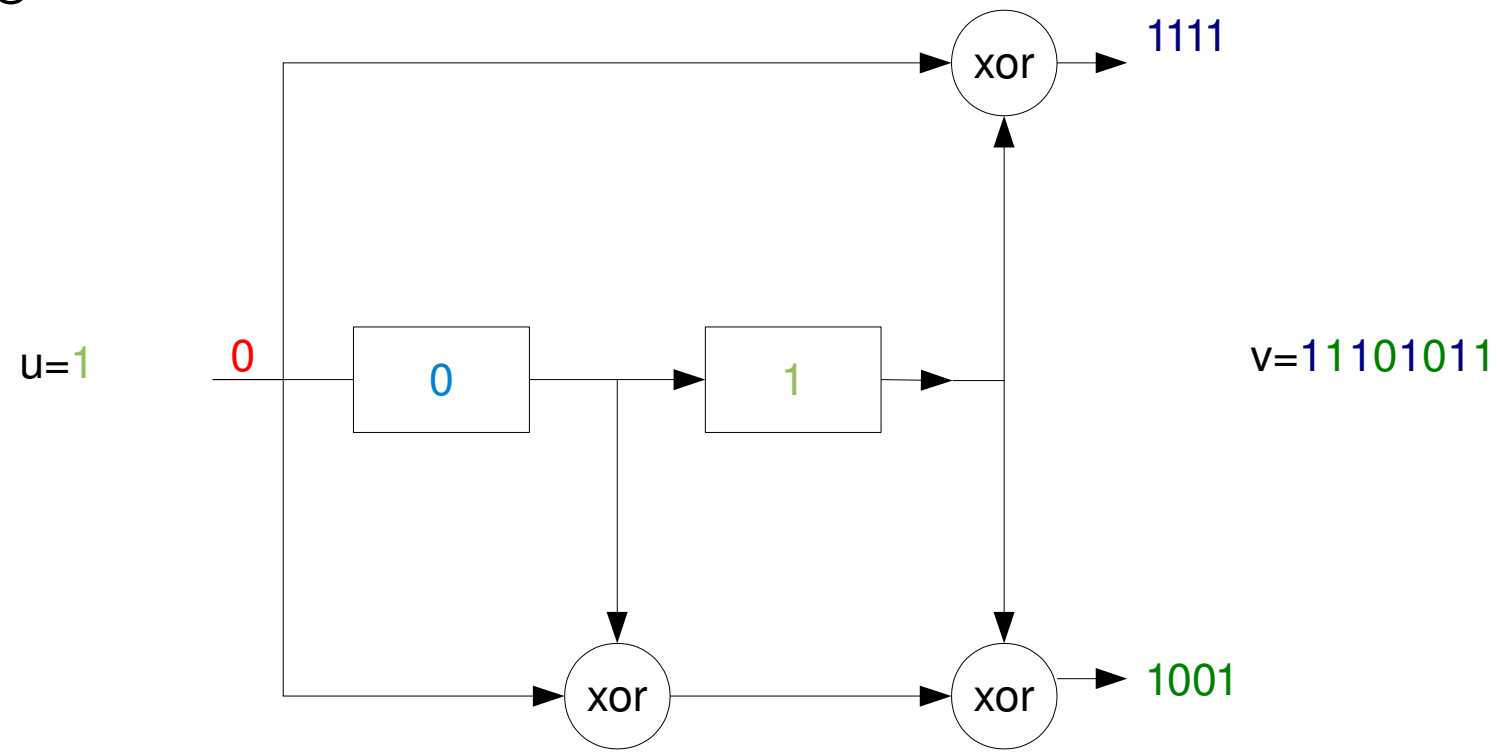
A message $u = 10011$



Convolutional Code

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

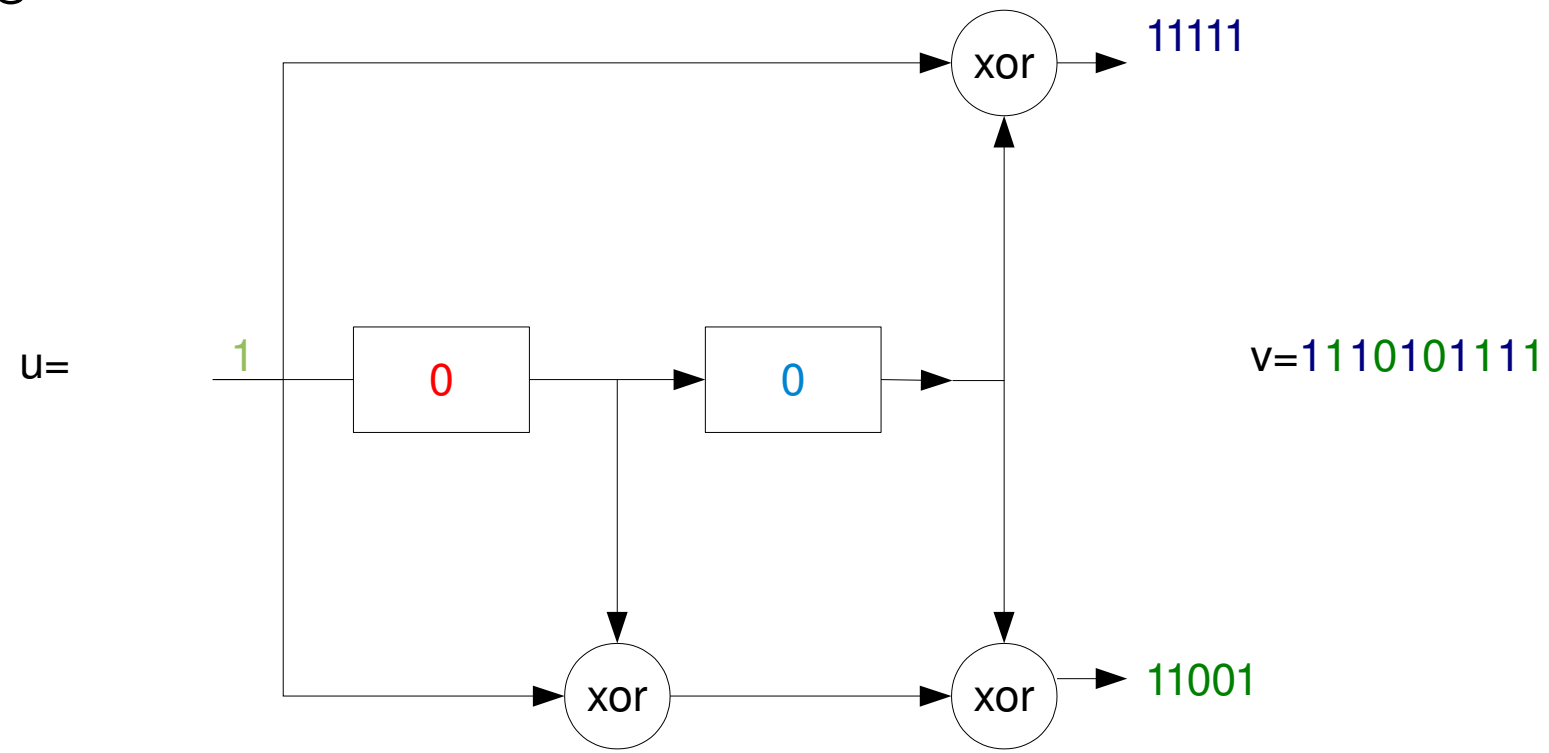
A message $u = 10011$



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) $K = M + 1$.

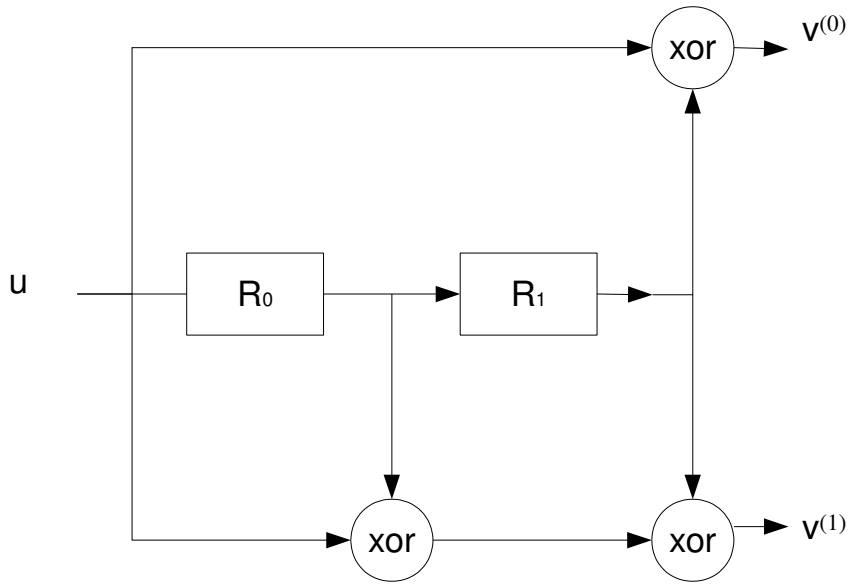
Notation

(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 $K - 1$.



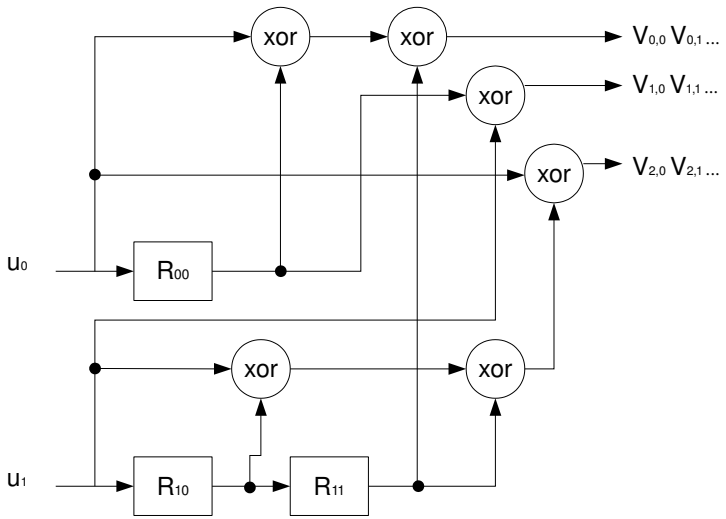
$\mathcal{C}$: $(2, 1, 3)$-convolutional code

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

Alternative View

A Convolutional Code

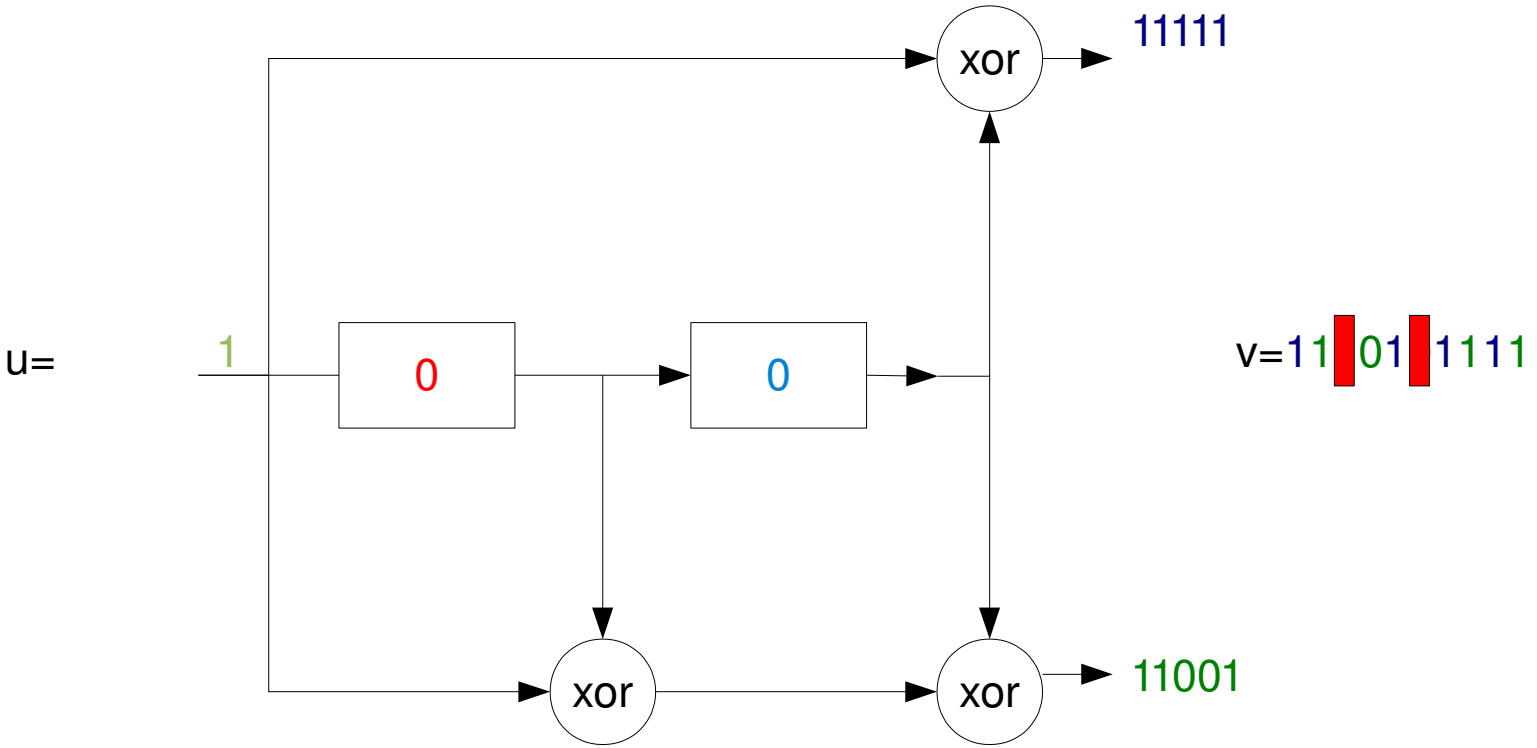
k registers with n polynomials operating on each register.
$n \times k$ polynomials for a (n, k, K)-convolutional code.
The degree of polynomials will be equal to $K - 1$.



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

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

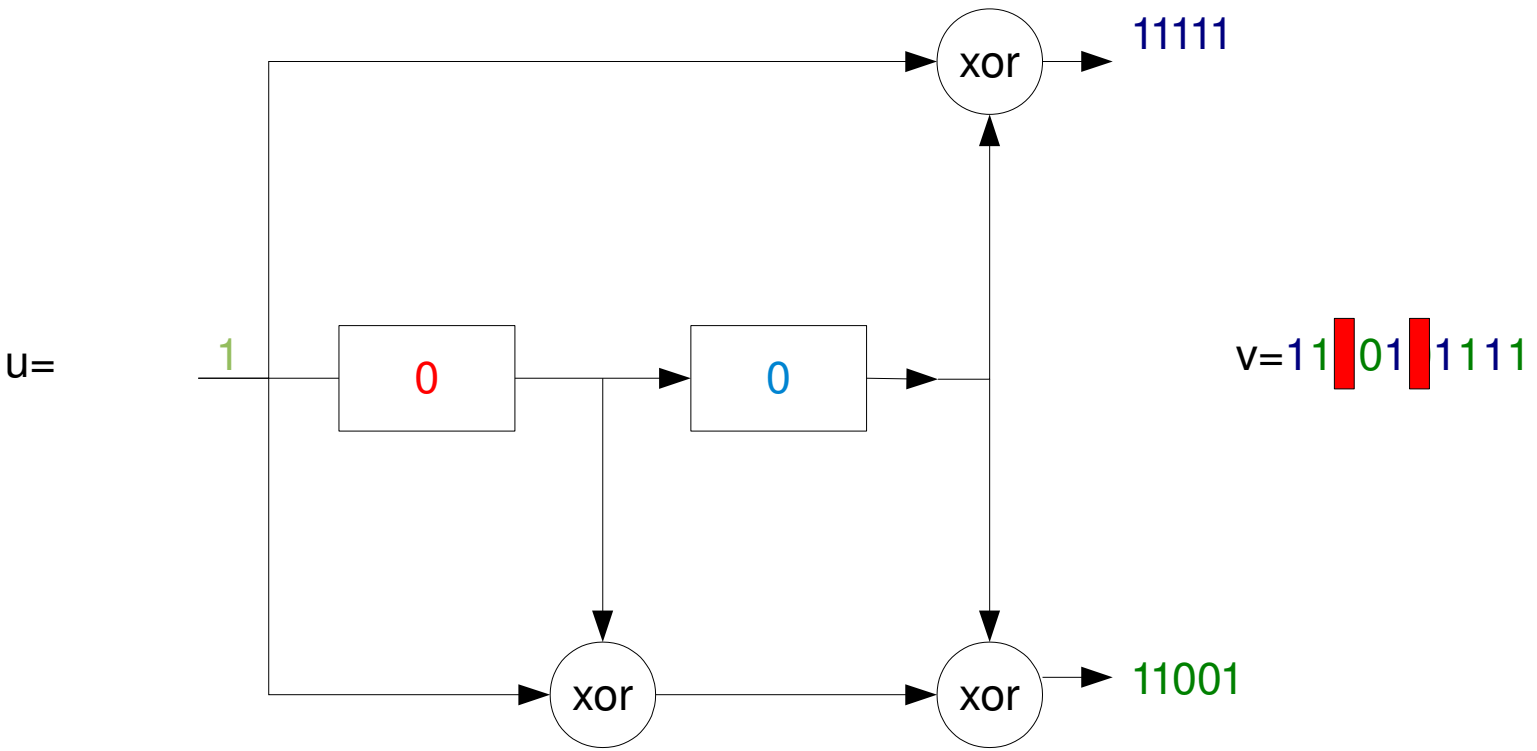
Puncturing



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

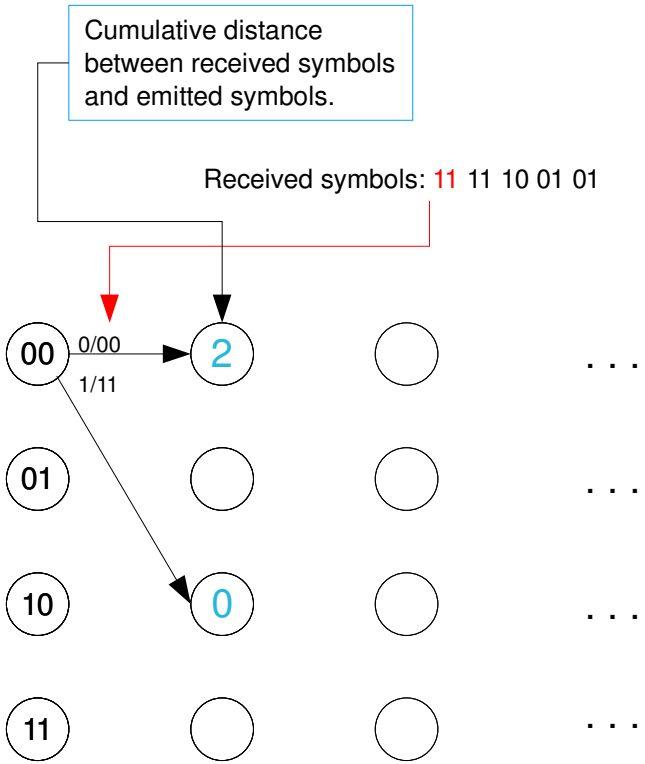
A (n, k, K)-convolutional code and a $J \times n$ puncturing matrix P of weight I.

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

Viterbi Algorithm

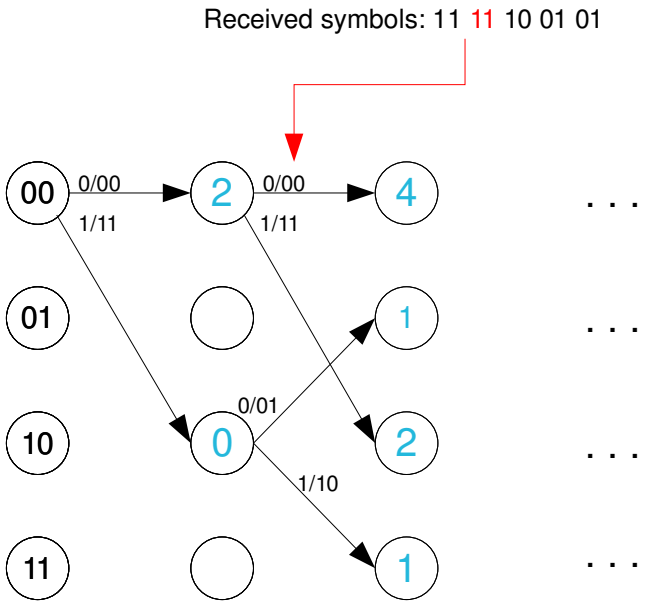
Viterbi Algorithm

Lattice construction



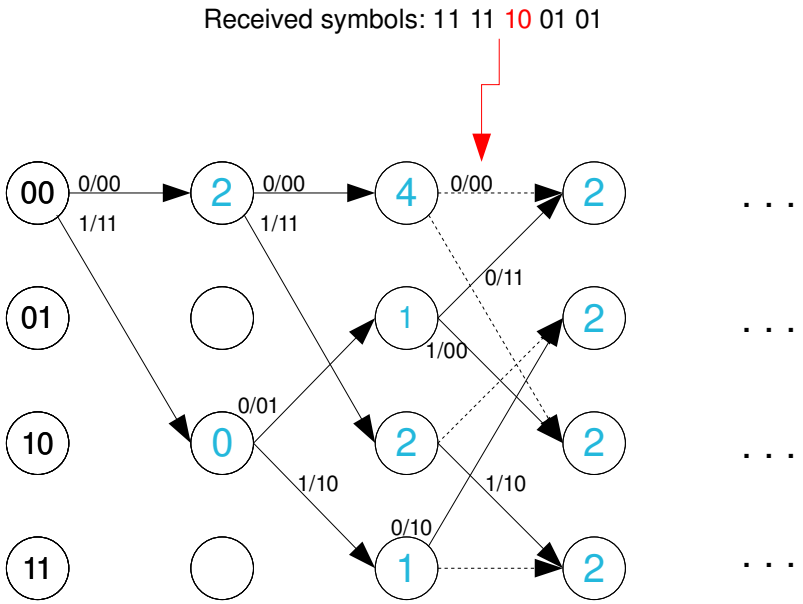
Viterbi Algorithm

Lattice construction



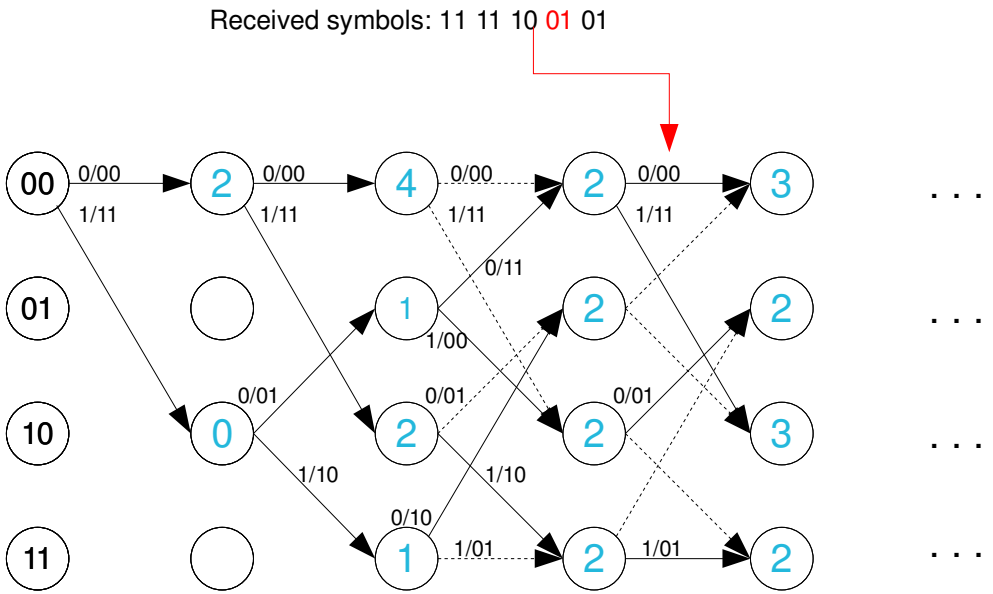
Viterbi Algorithm

Lattice construction



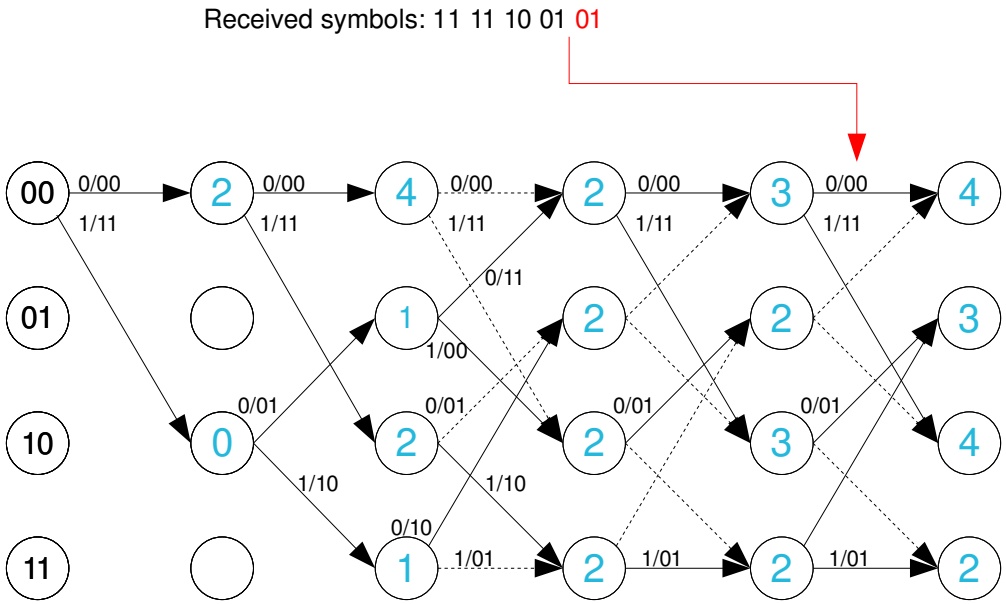
Viterbi Algorithm

Lattice construction



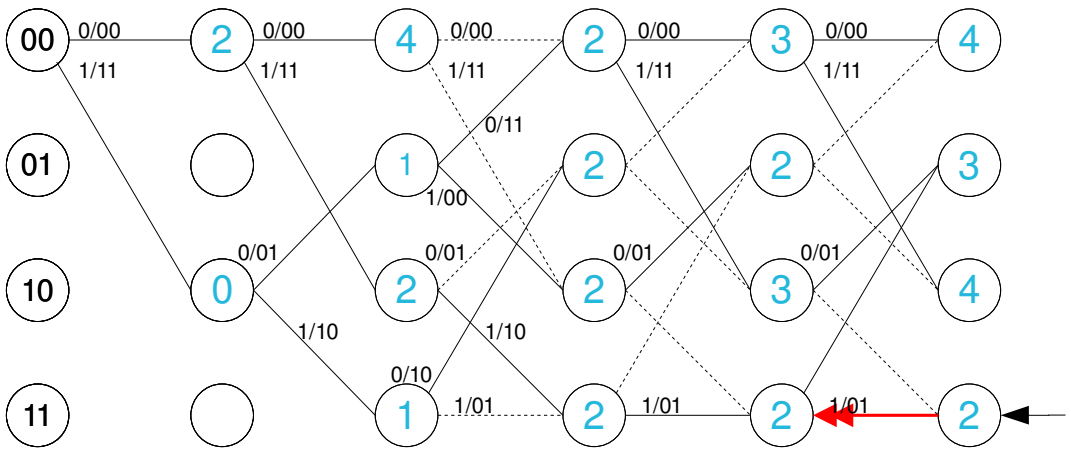
Viterbi Algorithm

Lattice construction



Viterbi Algorithm

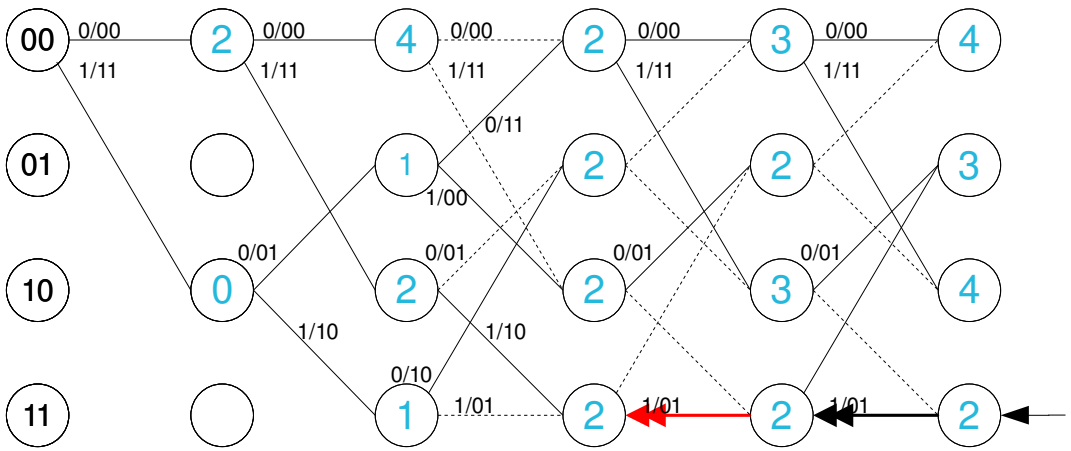
Backtracking



Decoded symbols: 1

Viterbi Algorithm

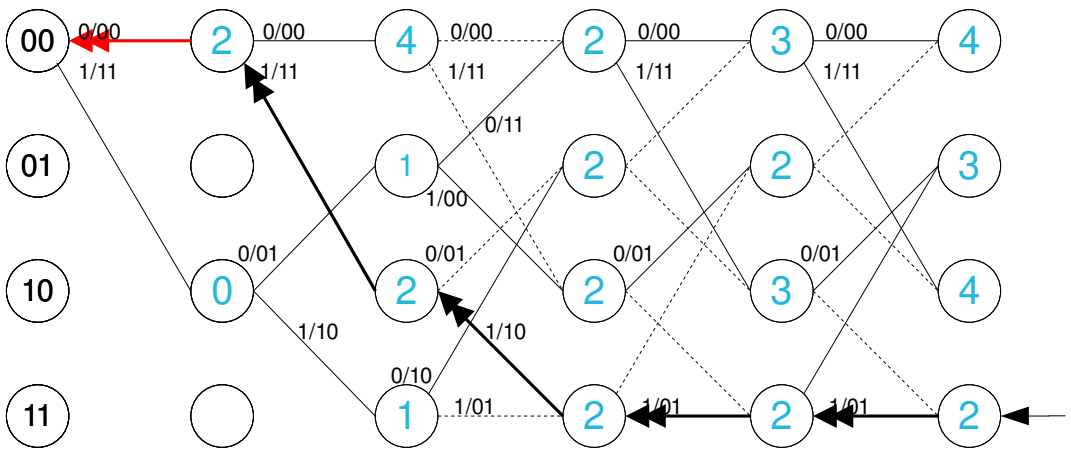
Backtracking



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

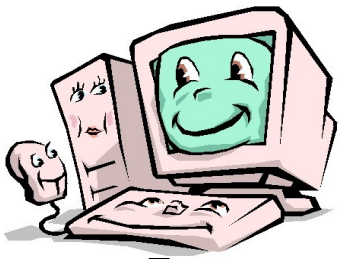
The Noise Impact

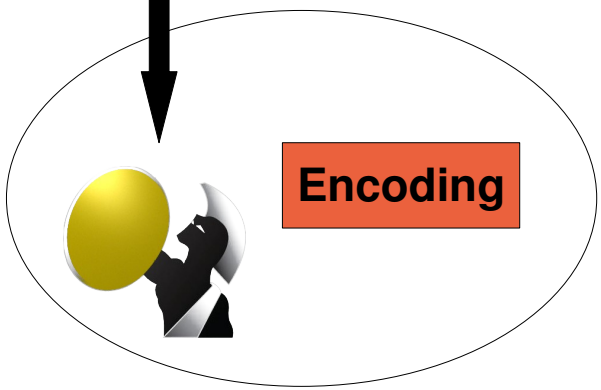
The probability to successfully reconstruct a code exponentially decreases with p. If $p > 10\% \Rightarrow$ online reconstruction is impossible ; offline reconstruction is computationally very hard.

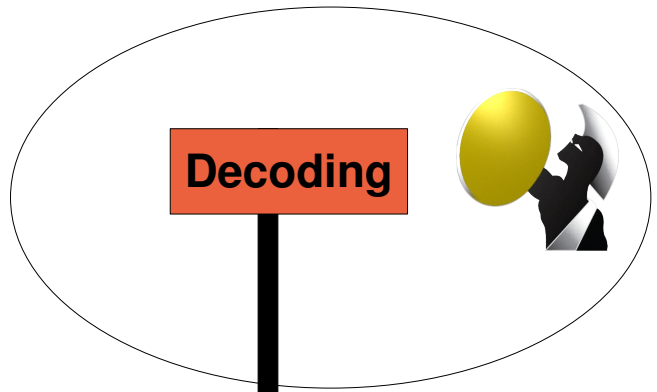
Encoder	Reconstruction Time ($p = 10^{-2}$)	Reconstruction Time ($p = 2.10^{-2}$)
(4, 3, 8)	7 min 12 sec	Failure
(4, 3, 9)	6 min 16 sec	Failure

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

Plan




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

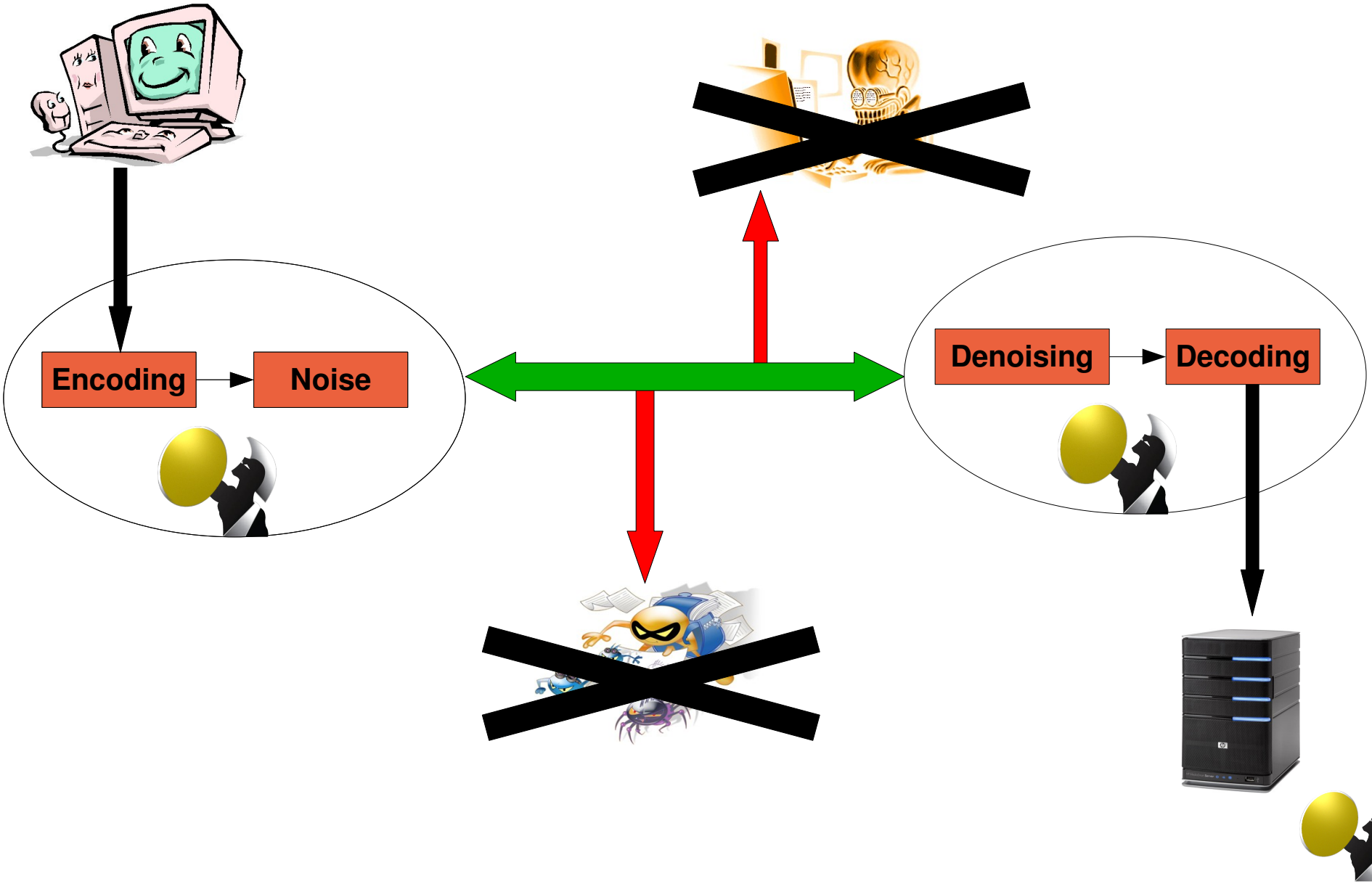
To greatly increase the security of data stream as well as the encoder used, noise must be added to the encoded data before transmission.

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

An array of ten 64-bit integers
Each integer contains approximatively 30% of 1s.
We choose one of the ten integers at random

What we need

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

Principle

An array of ten 64-bit integers
Each integer contains approximatively 30% of 1s.
We choose one of the ten integers at random

Shared Parameters

1. $0 \leq j < 10$ (random) enables to select an integer in the array.
2. X_0 a secret random 64-bit integer to initialize the pseudo-random noise generator.

We then have $\approx 2^{67}$ different noise generators.

What are the parameters

For every 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
7. j a random integer such that $0 \leq j < 10$

What are the parameters

For every 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
7. j a random integer such that $0 \leq j < 10$

What are the parameters

For every 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
7. j a random integer such that $0 \leq j < 10$

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).

To communicate with the server, PERSEUS uses HTTP headers.

- **check** : enables to ask the server whether the PERSEUS module is present or not.
- **check-ack** : server answer to the check request (means that PERSEUS is present).
- **known** : ask whether the server knows the parameters of the encoder or not. This header is used along with the **Perseus_id :XYZ** where XYZ is a random integer which enables to identify the encoder parameters.
- **known-yes** : the server acknowledges the parameters. Use the **Perseus_id :XYZ** as well.
- **known-no** : the server does not know the parameters.
- **init** : means that the request contains the encoder parameters ("POST" method). Uses the **Perseus_id :XYZ** field as well.
- **init-ack** : answer to the "init" request if everything is OK on the server's side.
- **init-nack** : answer to the "init" request if an error occured on the server's side.
- **PCC** : specifies that data in the request body are encoded (with a PCC). Uses the **Perseus_id :XYZ** field as well.

Scenarii
PERSEUS

protects GET and POST methods.
Several scenarii have been considered :

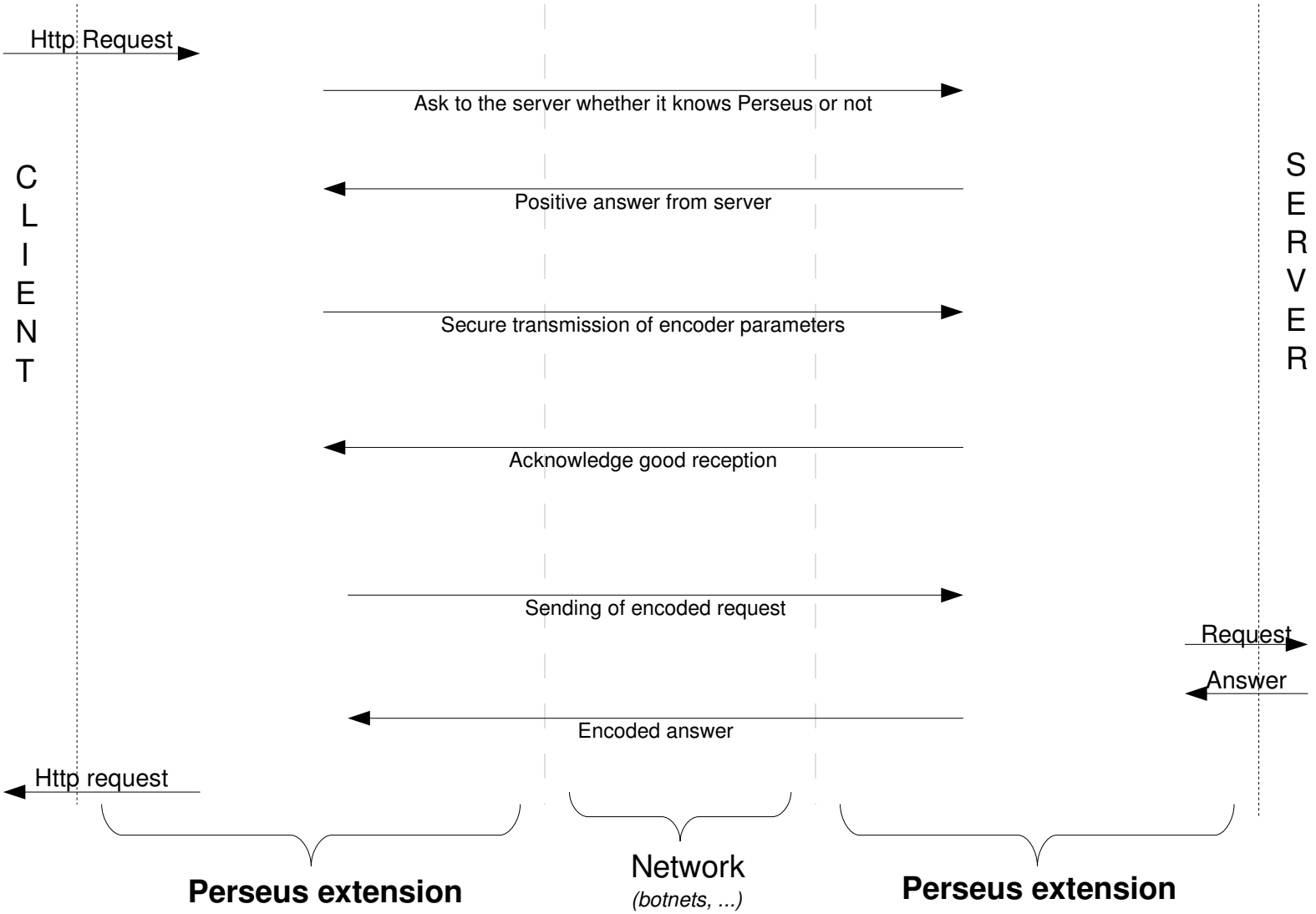
1. First (initial) communication between the server and the client.
   - The server does not use the PERSEUS module.
   - The server uses the PERSEUS module.
2. A communication link already exists (or existed).
   - The server already knows the encoder parameters.
   - The server does no longer know the encoder parameters.

Scenarii
PERSEUS

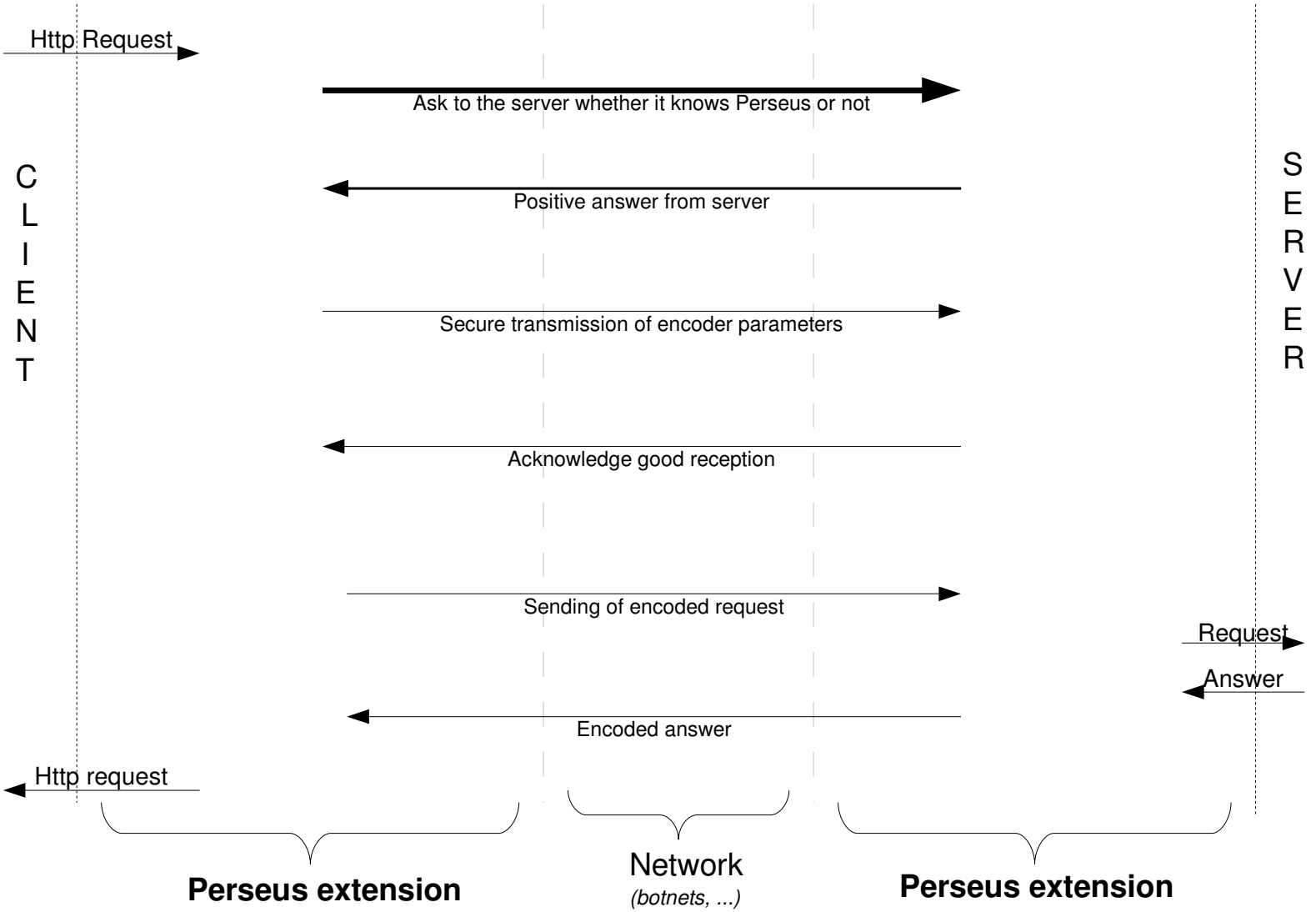
protects GET and POST methods.
Several scenarii have been considered :

1. First (initial) communication between the server and the client.
   - The server does not use the PERSEUS module.
   - The server uses the PERSEUS module.
2. A communication link already exists (or existed).
   - The server already knows the encoder parameters.
   - The server does no longer know the encoder parameters.

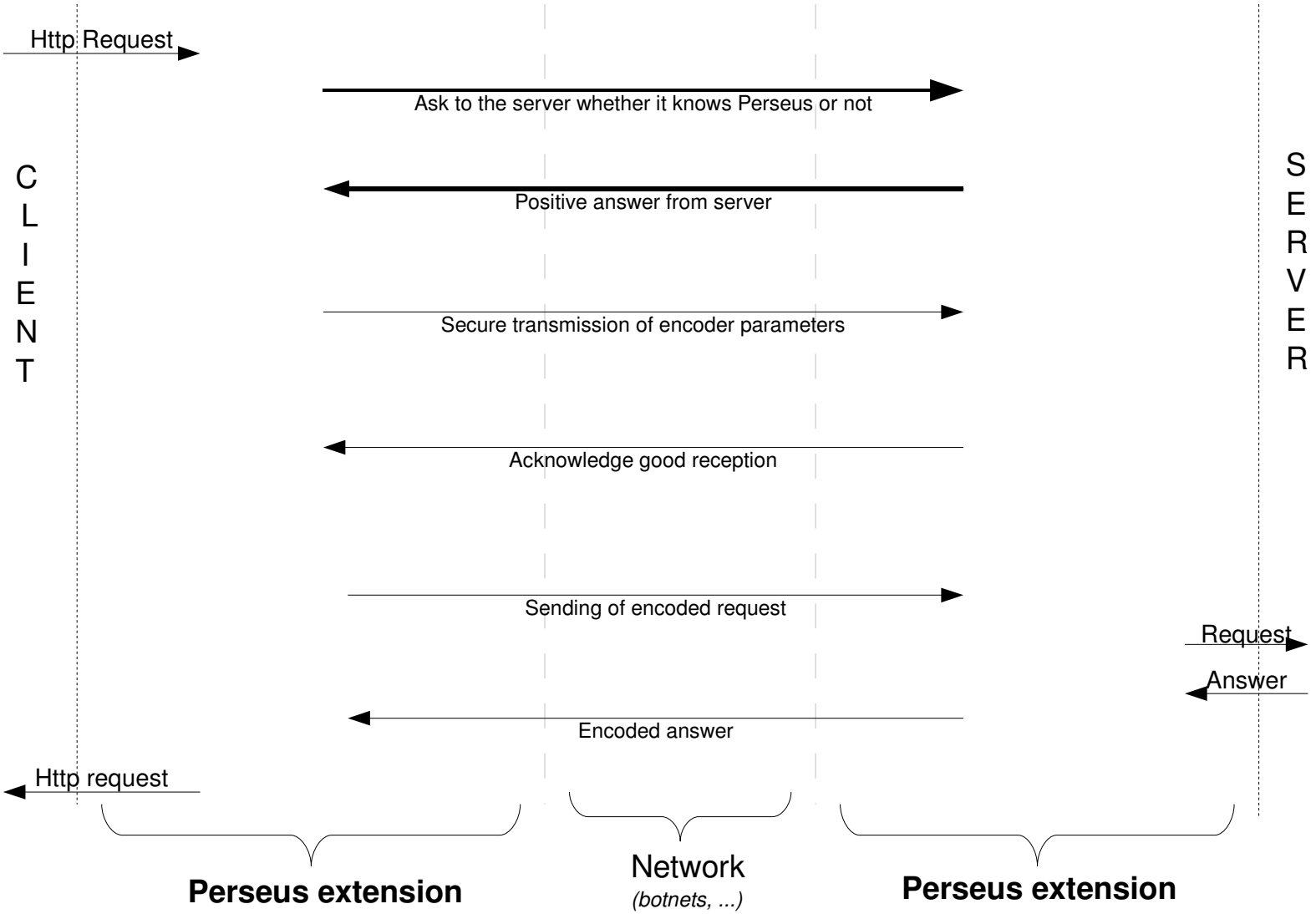
Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

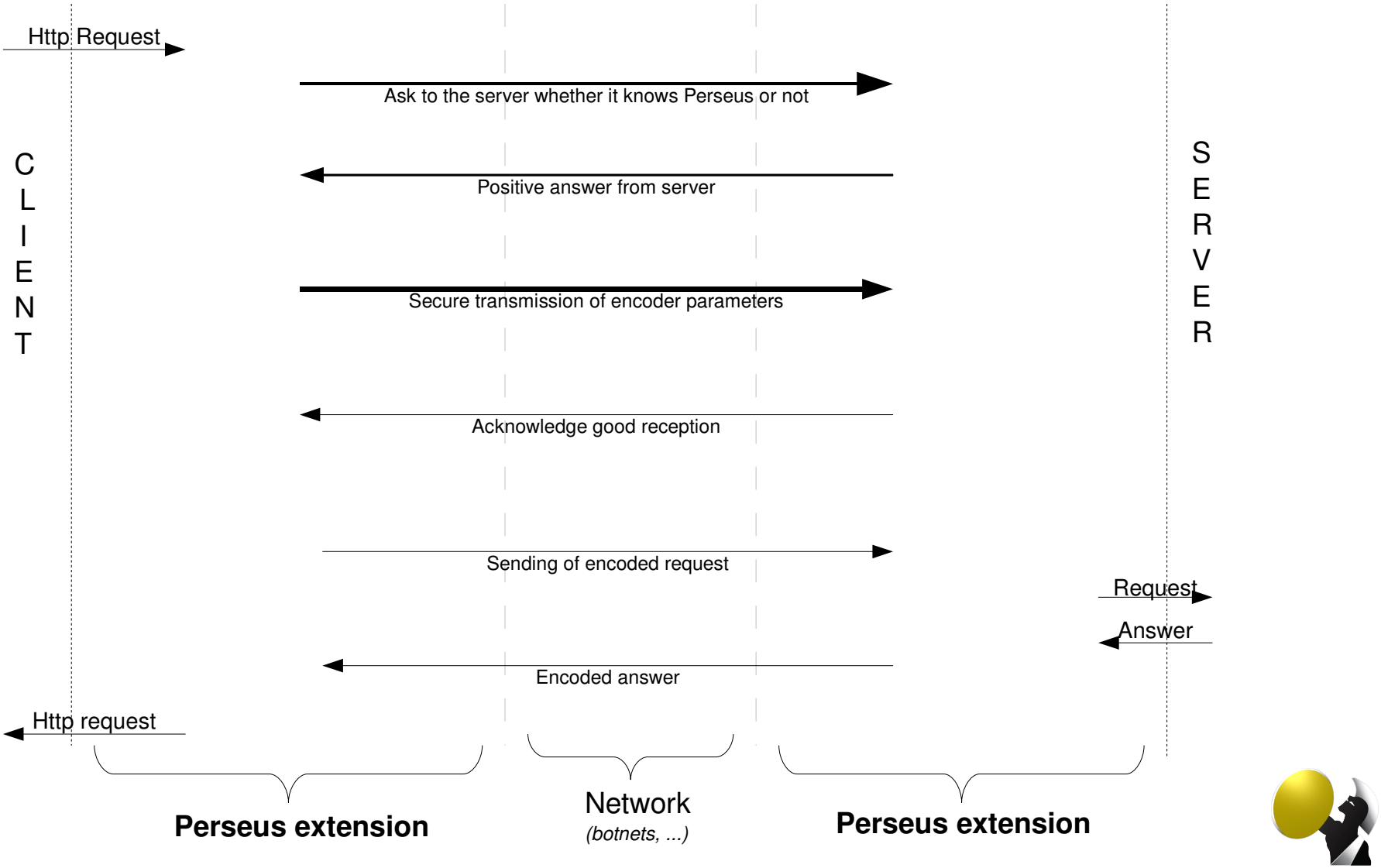
Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

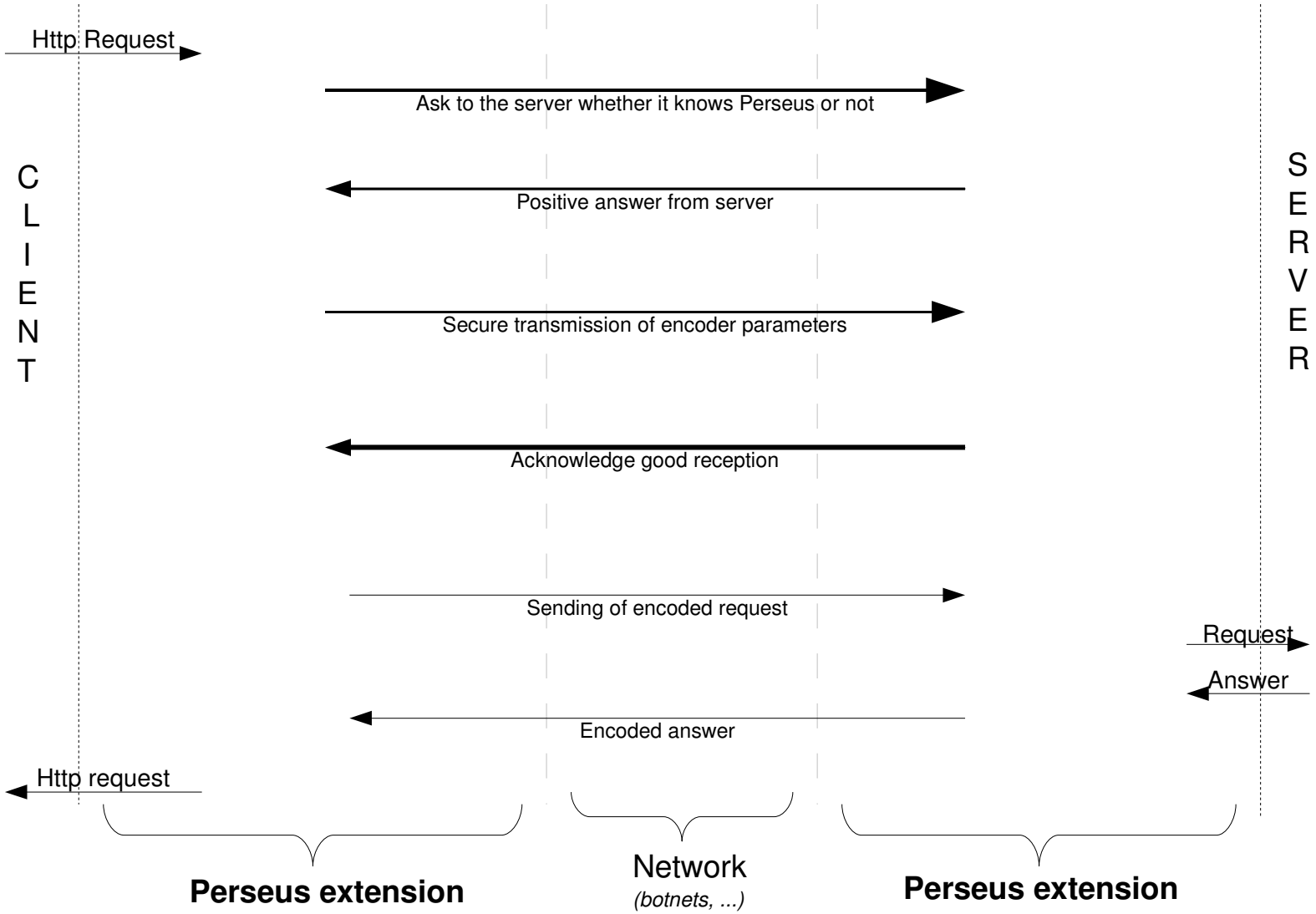
Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

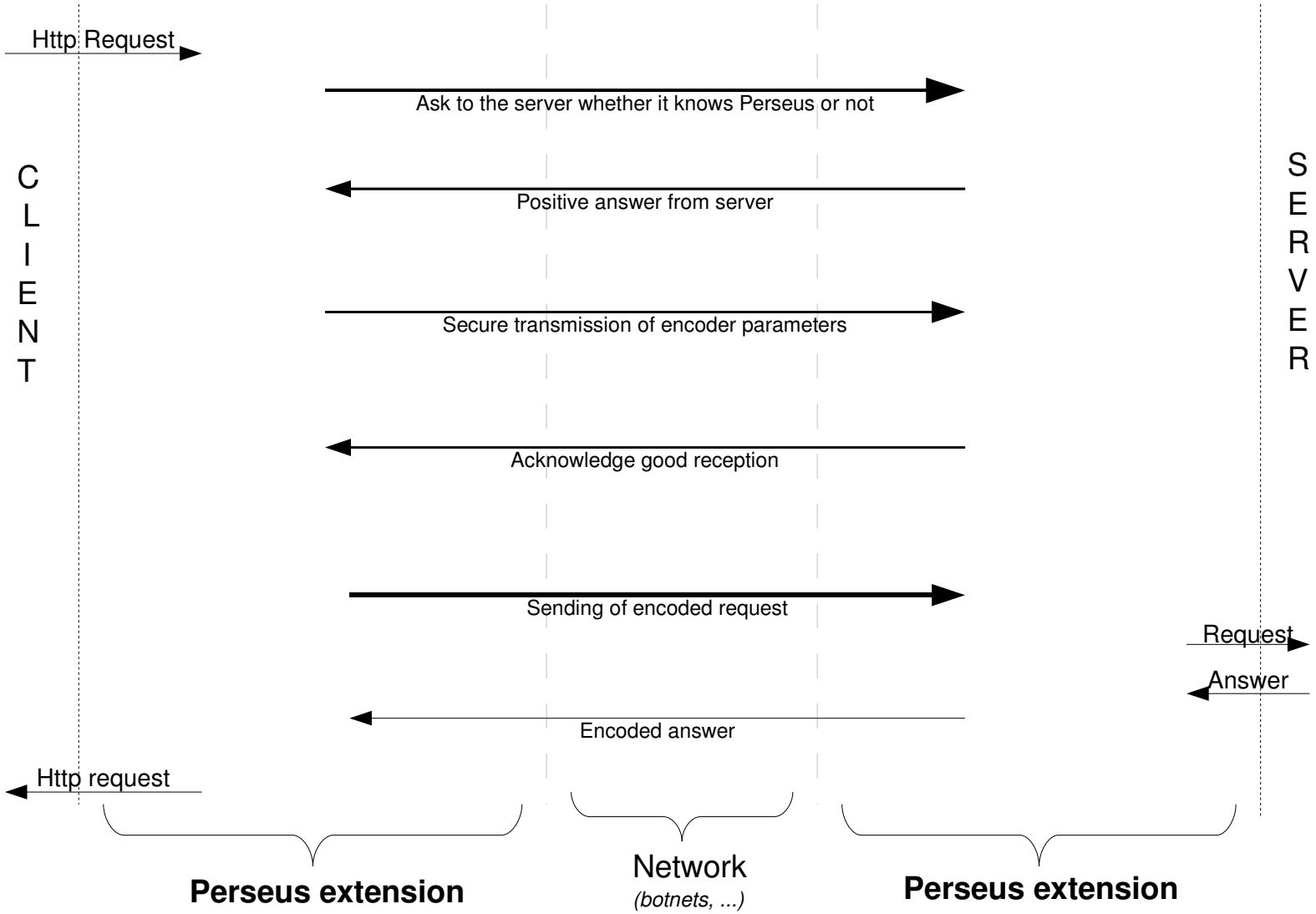
Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

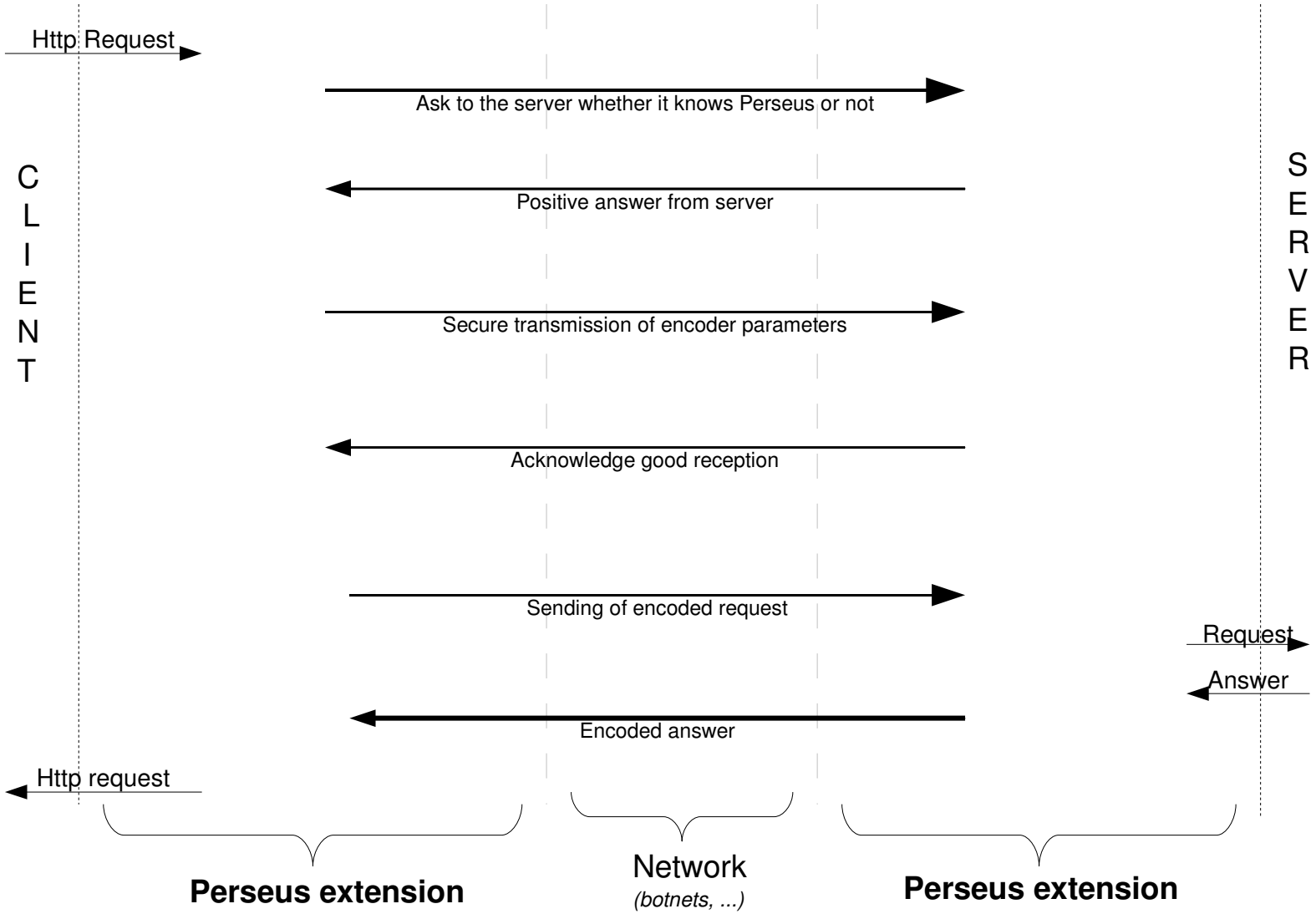
Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

Scenarii (2)


Http Request
Ask to the server whether it knows Perseus or not
CLIENT
Positive answer from server
SERVER
Secure transmission of encoder parameters
Acknowledge good reception
Sending of encoded request
Request
Answer
Encoded answer
Http request
Perseus extension
Network
(botnets, ...)
Perseus extension

Encoder	Machine	Encoding Time (μs)		Decoding Time (μs)		Data Exchange (s)
		Noise	Total	Noise	Total	
Size of data sent to the server : 231 octets (HTTPS :$0,050s$)						
$[4, 3, 18, 3, 2]$	Client Server	3460 4429	4463 5591	4370 3167	18362 11520	0,076138
$[2, 1, 28, 8, 7]$	Client Server	3151 4437	3983 5411	4625 3188	10296 5087	0,075423
$[4, 1, 22, 9, 8]$	Client Server	8713 12314	10245 13923	12513 8764	20348 11759	0,103056
$[11, 1, 18, 11, 10]$	Client Server	28892 39911	33535 43751	40169 28674	145110 89893	0,352391
$[8, 3, 21, 10, 9]$	Client Server	6737 9463	8897 12091	9375 6771	34189 22753	0,117641
Size of data sent to the server : 1 Ko (HTTPS :$0,050s$)						
$[6, 3, 17, 10, 9]$	Client Server	27717 29975	36701 37262	29672 28685	92395 80740	0,289190
$[3, 2, 23, 7, 6]$	Client Server	17803 17061	23420 21851	13984 17602	87633 88181	0,252633

TAB.: Processing and communication times with PERSEUS.

- The noise generator represents about 75% of the processing time.
- Current development of an optimized noise generator is bound to significantly reduce it.

Implementation

PERSEUS

Written in C++.
Under the triple MPL/GPL/LGPL licence.
Compliant with the Mozilla Function programming standards.
Source, documentation and binaries (XPI file) available at `http://www.mozdev.org/source/browse/perseus`.
Feedback, bug report and contributions requested (contact authors).
Project encouraged by the French DoD.

Future Work and Improvements

Management of UTF-16 encoding.
Management of FTP and FILE protocols.
The PERSEUS module will include a server as well.

Implementation

PERSEUS

Written in C++.
Under the triple MPL/GPL/LGPL licence.
Compliant with the Mozilla Function programming standards.
Source, documentation and binaries (XPI file) available at `http://www.mozdev.org/source/browse/perseus`.
Feedback, bug report and contributions requested (contact authors).
Project encouraged by the French DoD.

Future Work and Improvements

Management of UTF-16 encoding.
Management of FTP and FILE protocols.
The PERSEUS module will include a server as well.

Demos

PERSEUS Demo 1

Using PERSEUS.

PERSEUS Demo 2

Wireshark analysis without and with PERSEUS.

Demos

PERSEUS Demo 1

Using PERSEUS.

PERSEUS Demo 2

Wireshark analysis without and with PERSEUS.

Conclusion

- PERSEUS 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 ?