Symmetric-Key Cryptography

Thomas Peyrin
Nanyang Technological University

Workshop on Mathematics for Defence
Institute for Mathematical Sciences
Singapore - April 12, 2012

Outline

Outline

What is cryptography ?

Cryptography = science of secrecy

a mix of mathematics, computer science and electronics

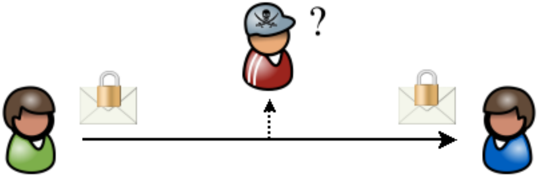
What is cryptography ?

Cryptography = science of secrecy

a mix of mathematics, computer science and electronics

Cryptography studies:

- **pure problems** such as confidentiality,



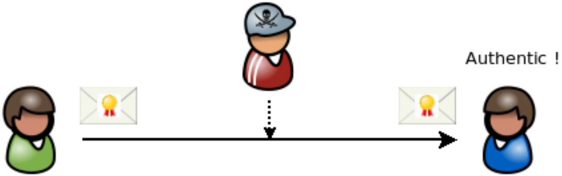
What is cryptography ?

Cryptography = science of secrecy

a mix of mathematics, computer science and electronics

Cryptography studies:

- **pure problems** such as confidentiality, authentication,



What is cryptography ?

Cryptography = science of secrecy

a mix of mathematics, computer science and electronics

Cryptography studies:

- **pure problems** such as confidentiality, authentication, integrity, etc.
- **complex protocols** such as identification, electronic voting, etc.

What is cryptography ?

Cryptography = science of secrecy

a mix of mathematics, computer science and electronics

Cryptography studies:

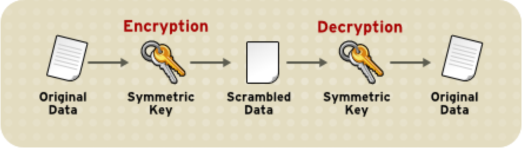
- **pure problems** such as confidentiality, authentication, integrity, etc.
- **complex protocols** such as identification, electronic voting, etc.

Cryptography is **everywhere** (security increasingly important):

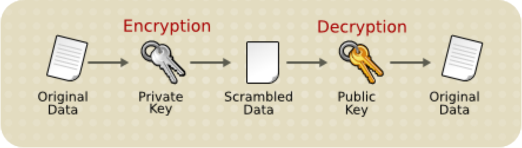
- **Industries:**
  telecommunications, banking, access control, logistic, medical, etc.
- **Applications:**
  PC, cellphones, smart-cards, Internet, supply chain, cars, etc.

What is symmetric/asymmetric-key cryptography ?

Symmetric-key cryptography: Two users A and B share the same secret key. A sends an encrypted message to B using its secret key, B deciphers using the same key.



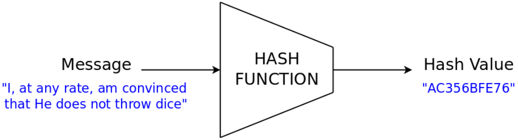
Asymmetric-key cryptography: A pair of keys private/public are given to every user. A sends an encrypted message to B using B's public key. Only B can decipher using its own private key.



Outline

What is a hash function ?

Hash function: an algorithm that transforms an arbitrary-length input message M into a fixed-length output value (hash value)

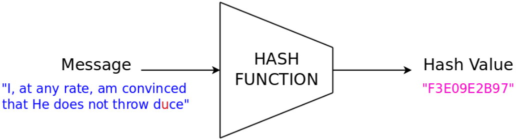


One should **NOT** be able to:

- invert the function (i.e. recover a message from the hash value)
- find two messages colliding (i.e. sharing the same hash value)

What is a hash function ?

Hash function: an algorithm that transforms an arbitrary-length input message M into a fixed-length output value (hash value)



One should **NOT** be able to:

- invert the function (i.e. recover a message from the hash value)
- find two messages colliding (i.e. sharing the same hash value)

Hash functions: applications

Many applications of hash functions:

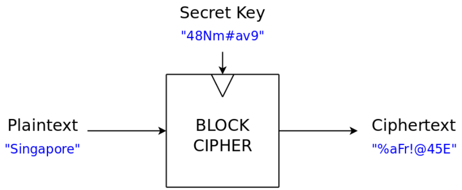
- **Signatures and Message Authentication Codes.** Allows to digitally sign a message or a file, and later verify the signature
- **Integrity check.** Used for example in most Internet protocols such as HTTP, FTP or P2P downloading
- **Passwords database protection.** Store the hash instead of the password
- **Confirmation of knowledge/commitment on a value.**
- **Pseudo-random number generator.** Allows to generate a sequence of numbers that approximates the properties of random numbers

Current status of hash functions:

- less mature field than block ciphers, very active
- most standardized hash functions got broken in 2004
- ongoing SHA-3 competition to select the future hash function standard

What is a block cipher ?

Block cipher: an algorithm that transforms a fixed-length block of plaintext P (unencrypted text) data into a block of ciphertext C (encrypted text) data of the same length, depending on a secret key K



One should NOT be able to:

- recover the secret key K faster than brute-force
- extract any information about the plaintext or the ciphertext

Block ciphers: applications

Many applications of block ciphers:

- **Confidentiality.** When used with an operating mode, it allows to securely transmit data over an insecure channel
- **Message Authentication Codes.** Allows to digitally sign a message or a file, and later verify the signature
- **Building block for other cryptography primitives.** Such as hash functions, stream-ciphers, etc.

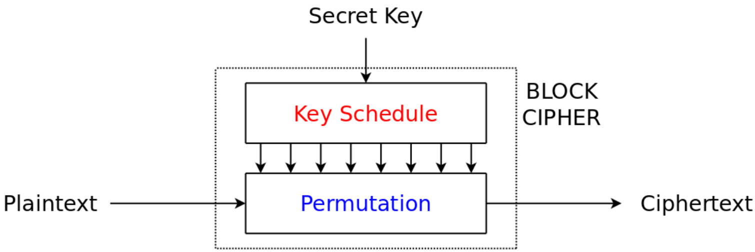
Current status of block ciphers:

- 1976-2001: DES algorithm.
- 2001-today: AES algorithm, after a 5-year competition
- very recent cryptanalysis work show some light weaknesses for AES
- many other block cipher proposals, depending on the application

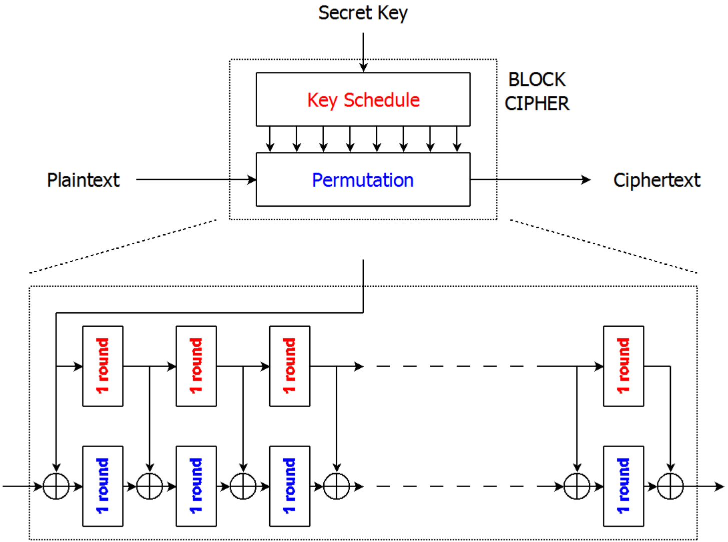
Outline

Outline

General construction of a block cipher

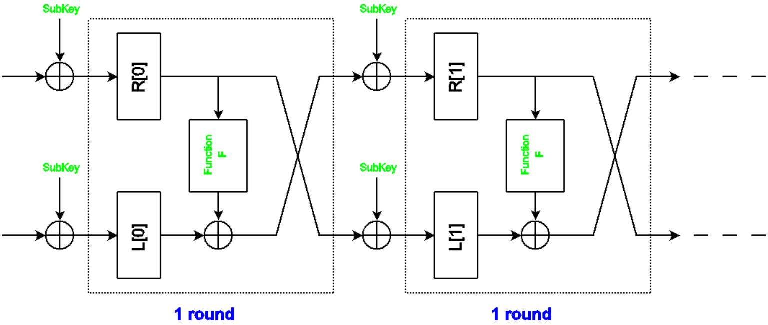


General construction of a block cipher



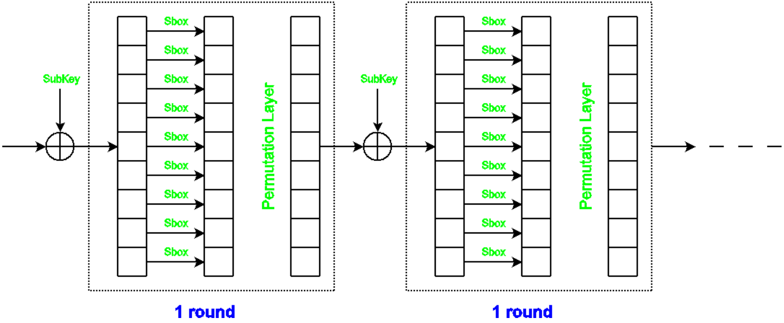
General construction of a permutation round

Feistel Network (DES)



General construction of a permutation round

Substitution-Permutation Network (AES)

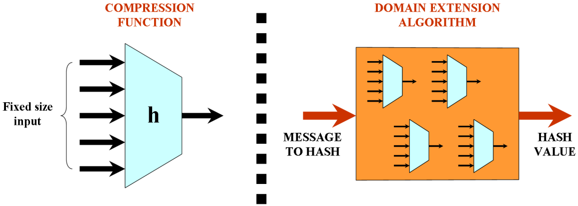


Outline

General construction of a hash function

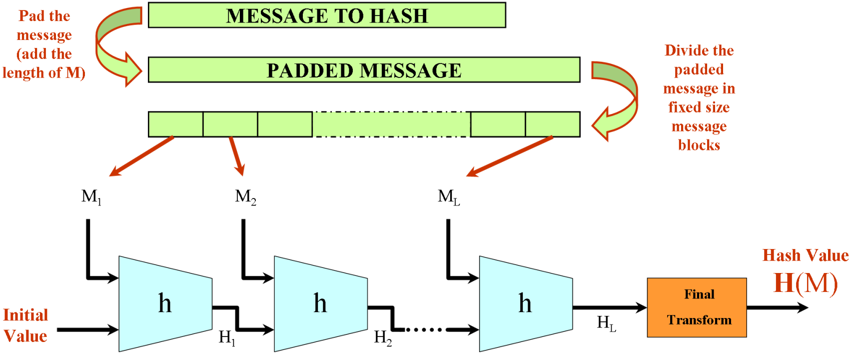
For historical reasons, most hash functions are composed of two elements:

- **a compression function h:** a function for which the input and output size is fixed.
- **a domain extension algorithm:** an iterative process that uses the compression function h so that the hash function H can handle inputs of arbitrary lenght.



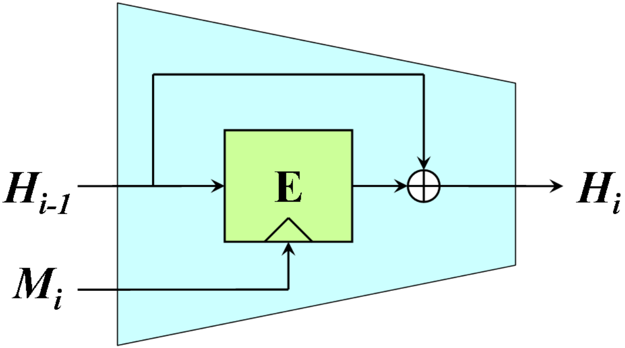
The Merkle-Damgård domain extension algorithm

The most famous domain extension algorithm used is called the **Merkle-Damgård** [Merkle Damgård-89] iterative algorithm.



The compression function

The MD/SHA family (MD4, MD5, SHA-0, SHA-1, SHA-2, ...)



Outline

Lightweight crypto ?

We expect **RFID tags** to be deployed widely (supply chain management, e-passports, contactless applications, etc.)

- we need to ensure authentication and/or confidentiality
- a basic RFID tag may have a total gate count of anywhere from 1000-10000 gates, with **only 200-2000 gates** budgeted for security
- hardware throughput and software performances are not the most important criterias, but they must be acceptable
- block ciphers and hash functions are used as basic blocks for RFID device authentication and privacy-preserving protocols.

Standard or `SHA-3` **hash functions are too big (around 10k GE)**

MDS Matrix

What is an **MDS Matrix** (“Maximum Distance Separable”) ?

- it is used as **diffusion layer** in many crypto primitives (in particular AES)
- it has excellent diffusion properties. In short, **for a d-cell vector, we are ensured that at least $d + 1$ input / output cells will be active** ...
- ... which is very good for linear / differential cryptanalysis resistance

The AES diffusion matrix can be implemented fast in software (using tables), but **the situation is not so great in hardware**. Indeed, even if the coefficients of the matrix minimize the hardware footprint, $d - 1$ **cells of temporary memory are needed for the computation**.

$$A = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as `AES` in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} =$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ \\ \vdots \\ \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as `AES` in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ \\ \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \\ \\ \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Efficient Serially Computable MDS Matrices

Idea: use a MDS matrix that can be efficiently computed in a serial way.

How to find it: build a very light matrix A and check if A^d is MDS.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ & \vdots & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \cdots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-4} \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{d-3} \\ v_{d-2} \\ v_{d-1} \\ v'_0 \end{pmatrix}$$

- we keep the same good diffusion properties since A^d is MDS
- **excellent in hardware (no additional memory cell needed)**
- **as good as** `AES` **in software**, we can use d lookup tables
- same coefficients for deciphering, so **the invert of the matrix is also excellent in hardware**

Tweaking AES for hardware: AES−HW

The smallest AES implementation requires 2400 GE with 263 GE dedicated to the MixColumns layer (the matrix A is MDS).

$$A = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}$$

A tweaked AES−HW implementation requires 2210 GE with 74 GE dedicated to the MixColumnsSerial layer (the matrix $(B)^4$ is MDS):

$$(B)^4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix}^4 = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix} \qquad B^{-1} = \begin{pmatrix} 2 & 1 & 4 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Thank you for your attention !