Generalised Rijndael

Sergi Blanch-Torné[1], Ramiro Moreno Chiral[2], Francesc Sebé Feixa[2], Magda Valls Marsal[2]

[1] Escola Politècnica Superior, Universitat de Lleida. Spain.
sblanch@alumnes.udl.es
[2] Departament de Matemàtica. Universitat de Lleida. Spain.
{ramiro,fsebe,magda}@matematica.udl.es

DRAFT
November 16, 2013

Abstract. [3]
Why to generalize? The standard AES establishes 3 key sizes and a unique block size, but starting from the original *Rijndael* there where 5 key sizes. This algorithm allows to adjust some parameters to changes those size restrictions maintaining the cryptographical properties because the mathematical background remains intact. While the mathematics remains valid against the attacks, the brute force can be fought by a key size increase, together with a block size that follows the key size, or if necessary fit the information size (for example to avoid overheads in message passing). This generalization goes further than the lightweight ciphers because it can be in this category at the same time that can play with other levels. Another reason for the generalization is to analyse if there are other configurations that can take advantage of the 64bit architectures, because the original *Rijndael* fits very well in 32 bits. What can really do a generalization is to increase the live time of an algorithm like the *Rijndael*.
Keywords: Cryptography, Symmetric, Block cipher, Rijndael

1 Introduction

In January 1997 the *National Institute for Standards and Technology* (NIST) set up a contest to replace the *Data Encryption Standard* (the DES, publish in 1975 and standardised in 1977, with last revision in 1999 [1]). This contest was to tag a modern symmetric algorithm as *Advanced Encryption Standard* (AES). The issue of the old DES was not the time, was the key size (56 bits). This has been patched to extend the live with the *triple-DES* (or 3DES) that works as: $E_{k_3}(D_{k_2}(E_{k_1}(plaintext)))$, where normally the $k_3 = k_1$, then the effective key size were $2 * 56 = 112$ bits (but having the possibility to become $3*56 = 168$). Even if the contest winner of the AES has something similar to the 3DES, called AESWrap [2], in this case the original algorithm can be extended to change the block size and the key size.

To this contest, 15 "opponents" have presented their candidacy (plus other 10 rejected on the preliminary stage due to security or efficiency reasons). In 1998 the *Rijndael* proposal [3] was sent, and in 1999 was again presented as an efficient algorithm for hardware implementations [4]. And was in August 1999 when the 5 finalists where announced: *MARS*, *RC6*, *Twofish*, *Serpent* and *Rijndael*. Then the algorithm was revised by the authors [5] and they wait until October 2000 when the announce was the choice [6]. After that, in 2002 the authors have published a book [7] with more description and detailed information about the algorithm and its internalises.

What gives the victory to the *Rijndael* was the good skills over software and hardware implementations, low memory requirements, key agility, and performance. Also it was the algorithm, on the contest, that shows a greater ability in the operations to protects itself against power and timing attacks without a significant impact on the performance.

When the *Rijndael* was chosen as the AES, the report [8], was highlighting those finalists over security and efficiency. As it is said in the report, all the finalists where tested over 32 bits architectures knowing that sooner or later the 64 bits will arrive (even the 128 bits architectures are mention as a possibility). One of the special features mention about the *Rijndael*is it flexibility to support other sizes than the proposed, even they only thought in gaps of 32 bits between those alternative block and key sizes. But it has a much bigger flexibility because other intermediate sizes can be also in the alternatives list. Is this *Rijndael* scalability the objective of this paper.

[3] Partially supported by grants MTM2010-21580-C02-01 (Spanish Ministerio de Ciencia e Innovación), 2009SGR-442 (Generalitat de Catalunya).

Paper structure:

- TODO: "*First of all introduce the maths behind the* Rijndael *in section 2.*"
- TODO: "*Design explanation in section 3.1 to explain the basic bricks and how they can be generalized in section 4.*"
- TODO: "*With the generalization there are equivalent sizes with different parameter combinations, explained in section 6.*"
- TODO: "*There is a new bunch of sizes with this to take advantage in section 5.*"

2 Polynomial binary rings and fields, the math below Rijndael

Before to enter in the design of the Rijndael schema is good to have a review over the background used. The mathematics used just above the *xor* operation in a binary group are polynomial fields defined with coefficients in this group, denoted as

$$\mathbb{F}_{2^n} = \frac{\mathbb{F}_2[z]}{m(z)} \tag{1}$$

Where the polynomial $m(z)$ is an irreducible with some given degree. As later will be used, for the notation of this degree we use w -known as *word size*- and has correspondence with the *Rijndael* schema of section 3.

This polynomial set has one operation, we can call it *addition* that describes an structure of an Abelian group $(G, +)$:

1. Closure: $\forall a(z), b(z) \in \mathbb{F}_{2^n}$ when those elements are operated $a(z) + b(z) = c(z)$, then $c(z) \in \mathbb{F}_{2^n}$
2. Associativity: $\forall a(z), b(z), c(z) \in \mathbb{F}_{2^n}$, then $[a(z) + b(z)] + c(z) = a(z) + [b(z) + c(z)]$
3. Identity: $\exists n(z) \in \mathbb{F}_{2^n} | \forall a(z), n(z) + a(z) = a(z) + n(z) = a(z)$
4. Inverse: $\forall a(z) \in \mathbb{F}_{2^n}$ must $\exists a^{-1}(z) | a(z) + a^{-1}(z) = n(z)$
5. Commutative: $\forall a(z), b(z) \in \mathbb{F}_{2^n}, a(z) + b(z) = b(z) + a(z)$

This polynomial set has a second operation, we call it *product* that has also an Abelian group structure. For convenience the *addition* operation has $0(z)$ as identity (and means the polynomial with all the coefficients at 0) and *product* has the polynomial $1(z)$ as identity (where all the coefficients are 1).

When one of those operations is an Abelian group, and the other is a non-commutative group, and together they satisfy the property:

1. '*' Distributive: $\forall a(z), b(z), c(z) \in \mathbb{F}_{2^n}, a(z) * (b(z) + c(z) = [a(z) + b(z)] * [a(z) + c(z)]$

This is called a *ring* $(R, +, *)$ as it is the basis for the operation of the `subBytes()` in section 4.3.

In the case of when those involved operations are Abelian groups, and it is also satisfied the property that the second operation is distributive respect the first, then this structure becomes what is called a *field* $(K, +, *)$.

At this point is already clear the algebraic properties of the field described in equation 1. But like this polynomial binary field uses an below a binary group, this field can be used as coefficients for an even above superstructure: a polynomial ring, where the coefficients are elements of the polynomial field:

$$(\mathbb{F}_{2^n})^l = \frac{\mathbb{F}_{2^n}[x]}{l(x)} \tag{2}$$

with $l(x)$ reducible and $gr(l(x)) = l$.

This polynomial ring is used in the Rijndael schema, specifically in section 4.5 with the `mixColumns()` operation.

The fact that in this case the modular polynomial $l(x)$ is a composited polynomial[4] with some degree (not necessarily the same than in the polynomial field, in fact for this we use c -known as *number of columns*- to denote this degree. This has correspondence with the *Rijndael* schema of section 4.5.

Remember that even if this distributive property is still valid, and because the polynomial modulo is composited, the second operation is non-commutative.

The only remaining mathematical background to explain is what means modular. Without enter in the explanation of the *equivalence classes*, the result of any operation inside the ring or the field must be

[4] say that is a *composited polynomial* is the same than say that it is reducible. Being factorizable this polynomial it has an algebraic structure of a ring. In the case of an *irreducible polynomial* what is get is an algebraic structure of a field

reduced modulo the polynomial that defines the set. That is, divide by this polynomial and use the rest as the equivalent representative in the set.

Those basic mathematical bricks of the *Rijndael* algorithm is what gives to it a genuine strongness. All the operations can be reduced to *xor*s ans bit shifts, and how they behave is regulated by mathematical elegance.

2.1 Operation example over the polynomial field

- TODO: "*Pick two "random" elements of the polynomial field of degree* 8*, using the* $m(z)$ *of the original* Rijndael*, and show their possible representations and operations like sum and product. (specially with the modular reduction). Choose this two "random" elements to have a result bigger than* $m(z)$ *that requires the reduction.*"
- TODO: "*Explain that in the lower level the addition operation can be set as a bitwise* xor *of the binary representation of the polynomial.*"

2.2 Operate in a polynomial ring, with coefficients in a polynomial field

- TODO: "*Pick two "random" elements of the polynomial ring of degree* 4*, using the* $l(x)$ *of the original* Rijndael*, and show their possible representations and operations like sum and product. (specially with the modular reduction). Like in the previous, pick this "random" elements to require reduction to get the result.*"
- TODO: "*Addition operation is not used, only the product (and don't forget it is not commutative)*".

2.3 Linear Codes

Later two concepts will be largely used and they are shortly described here:

Definition 1. *Given a set of symbols from an alphabet, the* Hamming weight *is defined as the number of symbols that are different than a symbol tagged as "zero" in the alphabet.*

In the binary alphabet this definition can be simplified as the number of $1's$ in a binary string.

Definition 2. *Given two strings of symbols from an alphabet, the* Hamming distance *is defined as the number of positions in those strings where the symbol on one string is different than the symbol in the second.*

A simplification of this definition for the binary case, is the number of *replacements* required to convert on string to the other.

3 Approach to the Rijndael Schema

Definition 3. *A Pseudo-Random Permutation (PRP) is defined as a application from the message space* $\mathcal{M}$ *and the key space* $\mathcal{K}$ *to the cipher space* $\mathcal{C}$*:*

$$PRP\text{: } \mathcal{M} \times \mathcal{K} \to \mathcal{C}$$

such that:

1. $\exists$ *"efficient"* deterministic *algorithm* $c = E(k, m)$
2. *The functions* E *is bijective*
3. $\exists$ *"efficient" inversion algorithm such that* $m = D(k, c)$

where E is a function to encrypt and D is its inverse function to decrypt.

A pseudo-random permutation is used as a symmetric cryptosystem like Shannon have defined in [9] the perfect secrecy concept in *part II, section 10.*

Definition 4. *A* cipher *has* perfect secrecy *if* $\forall m_1, m_2 \in \mathcal{M}$ *s.t.* $|m_1| = |m_2| \wedge \forall c \in \mathcal{C}$ *and* $k \in_R \mathcal{K}$ *(random and uniform distributed), the probability to that* c *comes from* m_1 *or* m_2 *are the same*

$$Pr[E(k, m_1) = c] = Pr[E(k, m_2) = c]$$

This means that c does not reveal *any* information about the original m. This can also by says like: The distribution of the cipher of a message is the same than the distribution from another message, or formally:

Definition 5. *For a perfect secrecy system, the distributions of the ciphers between messages in the cipher space is* computationally indistinguishable[5]*:*

$$\{E(k, m_1)\} \approx_p \{E(k, m_2)\}$$

Consider an scenario where an adversary has access to a random oracle where the output of this oracle can be or the output of the PRP or a truly random output, the advantage of the adversary to distinguish between if the output is get from one or the other can be described as:

$$Adv_F^{prp}(A) = Pr[Exp_F^{prp-1}(A) = 1] - Pr[Exp_F^{prp-0}(A) = 1] \tag{3}$$

where Exp_F^{PRP-1} is the probability to the adversary to win the bet that the output comes from a the PRP and Exp_F^{PRP-0} when the output comes from a truly random.

Definition 6. *A PRP is secure if for all "efficient" adversary, the advantage to distinguish if the output is from the PRP or the truly random is "negligible"*

In other words, a *PRP* is secure if the permutation given by it is indistinguishable from a truly random permutation. That means an Adversary can not take any advantage from the cipher text.

In the case of the Rijndael, the most efficient attacks on this symmetric cryptosystem, like the best key recovery attack it is *only* 4 times better than the exhaustive search using the biclique cryptoanalysis [10]. But this 4 times means that we must think in aes-128 to be like an aes-126 and this is still far, far away to an efficient break because it must be down to an attack in the order of 2^{64}. It means that this algorithm can be trusted as *still secure.*

At this point it must be mention that the key sizes of 192 and 256 bits have a weakness due to the design of the key expansion, but this will be explained later when in section 4.1 will be detailed this operation. Even that, this can be fixed because the incredibly good characteristics of the *Rijndael* are because it is absolutely based on an strong mathematical background.

Out of the standard specification [6], the revision of 1999 of the Rijndael block cipher [5] includes the section 12 about extensions. Is in this section where are mention block and key sizes different than the standardised having steps of 32 bits in between the 3 in the standard. This extensions can be because it only changes the number of columns. It already happens with the cases where the key have more columns than the block, and in a very similar way the block can also saw it number of columns increased.

From the 4 basic operations of the Rijndael this change is the one than can need less modifications in the bases. Following what has been mention about the simplicity, and the mathematical beauty and elegance of this schema, the increase of the number of columns is the parameter that causes less modifications in the design. Let see the design in more detail in next section 3.1.

3.1 The Rijndael Design

The design of the *Rijndael* starts its elegance by the data structure of the *State matrix.* This algorithm represents the data with in a matrix of a given number of rows and columns. On each cell of this matrix is allocated information in binary grouped in sets of the word size.

From the standardised *Rijndael* , 4 rows per 4 columns with words of 8 bit size.

$$\begin{bmatrix} S_{(0,0)} & S_{(0,1)} & S_{(0,2)} & S_{(0,3)} \\ S_{(1,0)} & S_{(1,1)} & S_{(1,2)} & S_{(1,3)} \\ S_{(2,0)} & S_{(2,1)} & S_{(2,2)} & S_{(2,3)} \\ S_{(3,0)} & S_{(3,1)} & S_{(3,2)} & S_{(3,3)} \end{bmatrix} \tag{4}$$

This sets the block size of the $4 \times 4 \times 8 = 128$ bits, and each cell represents an element of the polynomial field described in section 2. This representation gives a big simplicity, also because the order of the modulo (the irreducible polynomial $m(z)$ in equation 1) corresponds with 8 bits, the *word size*, a Byte: basic brick of the current computation resources.

[5] Denoted the meaning of computationally indistinguishable by the symbol $\approx_p$ because cannot be distinguished in a *polynomial time*

Also the columns are used as one element representation of the polynomial ring also described in section 2, the polynomial ring where the coefficients are elements of the mentioned polynomial field. Also the simplicity of this makes easy the operability because the chosen number of columns, 4 (the degree of the reducible polynomial $l(x)$ in equation 2) that corresponds the 4 Bytes, 32 bits, the most extended architecture size at the time of the *AES contest*.

Very smart, simple and versatile data representation.

With this matrix, the input plain text is transformed by the algorithm to the cipher text and the state matrix is the snapshot of each modification made with this transformation. Also with the decipher algorithm this state matrix contains the conversions to recover, from a cypher text the original plain text.

Understanding the *Rijndael* schema as a *Pseudo-Random Permutation*, the PRP from definition 3 on section 3, with the *State Matrix* the full *message space* $\mathcal{M}$ can be represented.

Also for the key a similar representation is used having a matrix but, because the standard allows different key sizes, in that case the number of columns is in the range between 4, 6 and 8 (for 128, 192, 256 key sizes).

What *Rijndael* does as a PRP is find a way to reach the goal of perfect secrecy (definition 4) and the best path to find that is what Shannon has defined as *diffusion & confusion* [9] (part III section 23)) by doing *substitutions & permutations*. An schematic view of the *Rijndael* algorithm can be saw in the figure 1 where the left branch shows to cipher operation and the right side the inverse or decipher. Those branches have a loop part of many rounds (the number of them will be discussed in section 4.2) proceeded and ceased by slightly different initial and final rounds. Each round is a composition of a set of operations design to give the substitutions and permutations in the given idea from Shannon's principle of diffusion and confusion.

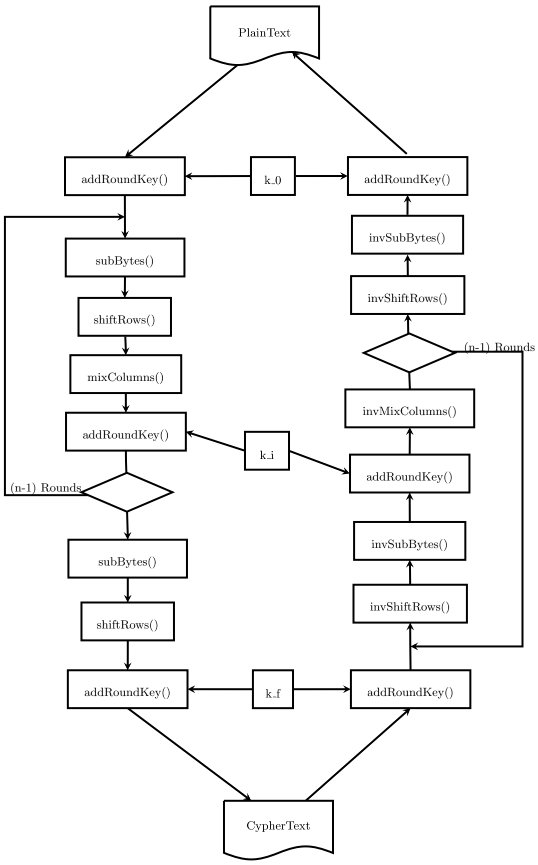

Fig. 1. rijndael diagram

4 Generalising the schema

This section has been divided in 6 parts. The first one about the *key expansion* used in the *Rijndael* to generate the subkeys needed for the algorithm rounds. The second section tries to clarify why the algorithm has been set up with this number of rounds. And the next four are dedicated to review the maths behind each of the operations in the *Rijndael* from a more generic view than the used in the official, in order to understand the steps of the next sections.

4.1 key expansion

The first important operation in the *Rijndael* schema is the *key expansion*. This operation takes the key as a seed to expand it until generate all the subkeys used on each round. In the smaller case of *AES* standard, the 128 bits become expanded to 1280 extra bits to be used like if the key was this size. But why the usage 128 bits in the key, can be trusted like having an independent key 10 times longer? This important mathematical item where this security rest is what is called *Pseudo-Random Generator*, PRG, generating a unique, and always the same, stream of bits from a much shorter seed. Lets define what PRG means:

Definition 7. *A* Pseudo-Random Generator *is a function that takes a seed and generates a much larger stream:*

$$PRG\text{: } \{0,1\}^s \rightarrow \{0,1\}^n \text{ ,where } n \ggg s$$

The goal is that the *PRG* must be efficiently computable by a deterministic algorithm and the output of it must look random and unpredictable. In fact, this is the most important property of a *PRG*, the unpredictability. But what predictability means:

Definition 8. *Given a* PRG $G : k \rightarrow \{0,1\}^n$ *is* predictable *if exist an efficient algorithm* A *such that:*

$$\Pr_{k \leftarrow \mathcal{K}}[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}] > \frac{1}{2} + \epsilon \tag{5}$$

for a non negligible ϵ

As an example, a non negligible ϵ usually is mention as a value $\geqslant \frac{1}{2^{30}}$.

The definition 8 says that for any efficient algorithm, given to this algorithm the i first bits, the probability that this algorithm predicts the next element of the stream is negligible.

The unpredictability of a *PRG* is what gives to it the quality to be *undistinguishable* from a *pure random generator*. Even when the seed space $\mathcal{K}$ of the PRG is much smaller than the space of the output $\{0,1\}^n$.

Like in the PRP, consider an scenario where the adversary has access to a random oracle where the output of this oracle can be the output of the PRG or a truly random stream. The advantage of the adversary to distinguish from where it comes any output it has take can be described as:

$$Adv_{prg}[A,G] = \left| \Pr_{k \xleftarrow{R} \mathcal{K}}[A(G(k)) = 1] - \Pr_{r \xleftarrow{R} \{0,1\}^n}[A(r) = 1] \right| \in [0,1] \tag{6}$$

That means if the probability gets close to 1, the adversary can distinguish G from random; and if it is close to 0 cannot.

Definition 9. *Given* $G : k \rightarrow \{0,1\}^n$*, is a* secure PRG *iff* $\forall$ *"efficient statistical test" the advantage* $Adv_{prg}[A,G]$ *is negligible.*

TODO: "*We shall proof this definition 9*"

Following definition 8 about the unpredictability, the ϵ is $\leqslant \frac{1}{2^{80}}$ to be considered as negligible.

Back to the *key expansion* of the *Rijndael* , and assuming that it is a *secure PRG*, it is time to take a look on the algorithm itself to "read" what can be generalized because this is the purpose of this paper. The input of the algorithm 1[6] takes, further than the seed key, the number of *rounds*, the number of *rows* and *columns*, and also the *word size*.

Algorithm 1 KeyExpansion

INPUT: nRounds, nRowns, nColumns, wSize, array of words $k_{in}[nRows * nColumns]$
OUTPUT: array of columns(array of words) $k_{out}[nColumns * (nRouns + 1)]$
1: i := 0
2: **while** $i < nColumns$ **do**
3: $k_{out}[i]$:= column($k_{in}[nRows * (i + c)$ for c in $range(nColumns)$])
4: **end while**
5: i := nColumns
6: **while** $i < nRouns * (nRows + 1)$ **do**
7: temp := $k_{out}[i - 1]$
8: **if** i mod nColumns == 0 **then**
9: temp := SubBytes(RotWord(temp)) ⊕ Rcon[i/nColumns]
10: **else**
11: temp := SubBytes(temp)
12: **end if**
13: $k_{out}[i] := k_{out}[i - nColumns]$ ⊕ temp
14: i++
15: **end while**

All those parameters are necessary to have in the output an expanded key large enough to use all the necessary subparts under each round, as it is shown in the figure 1.

Perhaps a better way to see the algorithm 1 is the figure 2. From steps 1 to 5 in the algorithm it simply "moves" the input key to the first part of the generated output. The main part of the algorithm, starts on the step 6 where each column further than the original columns of the key are generated.

With this figure seems to be easier to recognise this iterative algorithm, that is generating the new column i by taking the previous (with may be some transformations when the key size is bigger than the block size) and the one in the same relative position of the previous *subkey*, to do over it some transformations to introduce diffusion and confusion in the newer bits generation.

Each step finish with 3 *xor* operations to catch together all the partials on this step generation. The *xor* operation is the most important operation and is the most used in the lower level of the *Rijndael*. This is one of the bests characteristics of this algorithm.

Looking on the figure 2 and corresponding to the line 5 of the algorithm 1, and two extra primitive arr called over certain columns (the ones multiple of the initial number of columns), the `rotWord()` and the `Rcon` (name used in the algorithm 1 represented as $1^i \in \frac{\mathbb{F}_{2^w}[z]}{m(z)}$ in the figure 2), when the `subBytes()` is called on each iteration. The effect of these different transformations on certain columns is propagated to the next columns on the subset because the input to build a new one uses the previous. But lets seen a bit what they do.

At this point we only need to know that the `subBytes()` uses an *SBox* and the other details are explained in section 4.3. The `rotWord()` does a cyclic permutation of one step to the left. And the `Rcon` best way to define it is the formulation in the figure 2, and more formally:

$$\texttt{Rcon[n]} = x(n) = x^n = x \cdot x^{n-1} = x \cdot x(n-1) \tag{7}$$

with an initial value of $x(1) = 2$ in $\mathbb{F}_{2^8}$.

FIXME: "`Rcon` *as with* $x(1) = 2$ *in* $\mathbb{F}_{2^8}$ *can be generalized for* $\mathbb{F}_{2^w}$*?*"

This `Rcon` can be pre-calculated and stored in a table or recursively generated in each of the rounds on the loop.

- TODO: "*What means to have different number of columns in the message than in the key matrix representation.*"

An attack to the *PRG* of the Rijndael is described in [11] and affects the cases where the size of the key is not the same than the size of the block. Even that, this attack requires up to 2^{99} pairs (m, c) and 4 *related keys*[7]. The recover time of this attack is around 2^{99} that is still far away from a weakness to be worried to distrust the algorithm. Also avoiding to use related keys, this attack would not apply.

[6] This algorithm has been taken from the version 2 of the AES proposal [5] that is the same than becomes the NIST standard [6]. Different but equivalent than the algorithm give in the book from the Rijndael authors [7], published later than the standard

[7] *Related keys* means that the *Hamming* distances (definition 2) are very short and the difference between one key to another are a few bits that are flipped.

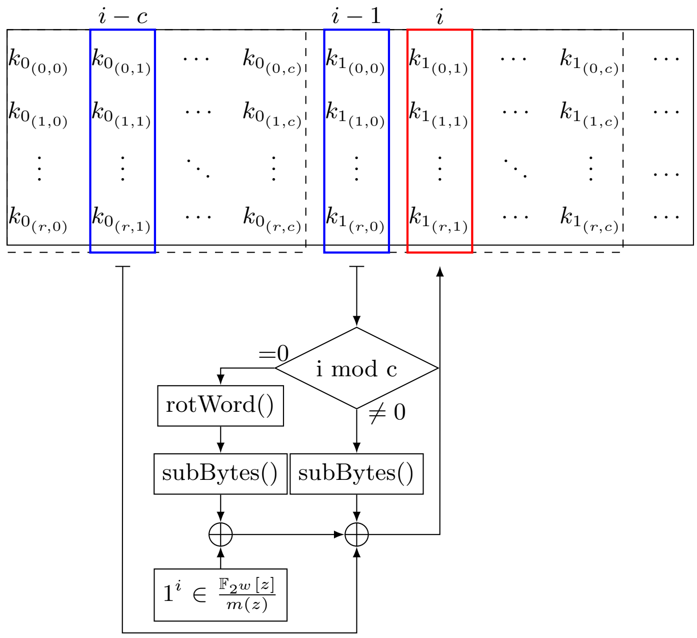

Fig. 2. Block diagram of the iterative construction of the *Rijndael Key Expansion* as a *PseudoRandomGenerator*, PRG FIXME: “*this figure has a mistake and is not corresponding with algorithm 1*”

TODO: “*Explain why the key sizes of* 192 *and* 256 *bits have a weakness on the design of the key expansion*”

4.2 Rounds

In the AES proposal of the Rijndael [5] (section 4.1) the number of rounds is described as a function of the block and the key length, followed by the table:

N_r	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	10	12	14
$N_k = 6$	12	12	14
$N_k = 8$	14	14	14

(8)

But is in section 7.6 where is said that this number has been determined by looking in to the most efficient attacks (known at that time) and adding a security margin. That is improved with in section 12.1, where the number of rounds is described by a function:

$$N_r = max(N_k, N_b) + 6 \tag{9}$$

This function means that the number of rounds is the biggest number of columns between the block and the key, plus the security margin set as 6.

There is not theoretical method to stablish which is the good number of rounds, it is only an empirical attempt base on known attacks, As this paper tries to propose different sizes for the *Rijndael*, will be in section 6 about the parameter combinations where this number will be discussed.

- TODO: “*Which is the contribution of each round to the confusion and diffusion?*”
- TODO: “*From the known cryptoanalysis, how can be set up a good number of rounds?*”

4.3 subBytes

Looking on the schema of the figure 1 the first operation is the `subBytes()`, skipping the first round where it is the `addRoundKey()` with k_0 –that will be explain later in section 4.6–. About what has been said in section 3.1 about Shannon’s perfect secrecy, this operation skill is the *confusion* because is a permutation. This transformation is a non-linear substitution of each word in the *state* matrix. In the

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x0	0x63	0x7C	0x77	0x7B	0xF2	0x6B	0x6F	0xC5	0x30	0x01	0x67	0x2B	0xFE	0xD7	0xAB	0x76
0x1	0xCA	0x82	0xC9	0x7D	0xFA	0x59	0x47	0xF0	0xAD	0xD4	0xA2	0xAF	0x9C	0xA4	0x72	0xC0
0x2	0xB7	0xFD	0x93	0x26	0x36	0x3F	0xF7	0xCC	0x34	0xA5	0xE5	0xF1	0x71	0xD8	0x31	0x15
0x3	0x04	0xC7	0x23	0xC3	0x18	0x96	0x05	0x9A	0x07	0x12	0x80	0xE2	0xEB	0x27	0xB2	0x75
0x4	0x09	0x83	0x2C	0x1A	0x1B	0x6E	0x5A	0xA0	0x52	0x3B	0xD6	0xB3	0x29	0xE3	0x2F	0x84
0x5	0x53	0xD1	0x00	0xED	0x20	0xFC	0xB1	0x5B	0x6A	0xCB	0xBE	0x39	0x4A	0x4C	0x58	0xCF
0x6	0xD0	0xEF	0xAA	0xFB	0x43	0x4D	0x33	0x85	0x45	0xF9	0x02	0x7F	0x50	0x3C	0x9F	0xA8
0x7	0x51	0xA3	0x40	0x8F	0x92	0x9D	0x38	0xF5	0xBC	0xB6	0xDA	0x21	0x10	0xFF	0xF3	0xD2
0x8	0xCD	0x0C	0x13	0xEC	0x5F	0x97	0x44	0x17	0xC4	0xA7	0x7E	0x3D	0x64	0x5D	0x19	0x73
0x9	0x60	0x81	0x4F	0xDC	0x22	0x2A	0x90	0x88	0x46	0xEE	0xB8	0x14	0xDE	0x5E	0x0B	0xDB
0xA	0xE0	0x32	0x3A	0x0A	0x49	0x06	0x24	0x5C	0xC2	0xD3	0xAC	0x62	0x91	0x95	0xE4	0x79
0xB	0xE7	0xC8	0x37	0x6D	0x8D	0xD5	0x4E	0xA9	0x6C	0x56	0xF4	0xEA	0x65	0x7A	0xAE	0x08
0xC	0xBA	0x78	0x25	0x2E	0x1C	0xA6	0xB4	0xC6	0xE8	0xDD	0x74	0x1F	0x4B	0xBD	0x8B	0x8A
0xD	0x70	0x3E	0xB5	0x66	0x48	0x03	0xF6	0x0E	0x61	0x35	0x57	0xB9	0x86	0xC1	0x1D	0x9E
0xE	0xE1	0xF8	0x98	0x11	0x69	0xD9	0x8E	0x94	0x9B	0x1E	0x87	0xE9	0xCE	0x55	0x28	0xDF
0xF	0x8C	0xA1	0x89	0x0D	0xBF	0xE6	0x42	0x68	0x41	0x99	0x2D	0x0F	0xB0	0x54	0xBB	0x16

Fig. 3. Sbox for 8 bits word size

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x0	0x52	0x09	0x6A	0xD5	0x30	0x36	0xA5	0x38	0xBF	0x40	0xA3	0x9E	0x81	0xF3	0xD7	0xFB
0x1	0x7C	0xE3	0x39	0x82	0x9B	0x2F	0xFF	0x87	0x34	0x8E	0x43	0x44	0xC4	0xDE	0xE9	0xCB
0x2	0x54	0x7B	0x94	0x32	0xA6	0xC2	0x23	0x3D	0xEE	0x4C	0x95	0x0B	0x42	0xFA	0xC3	0x4E
0x3	0x08	0x2E	0xA1	0x66	0x28	0xD9	0x24	0xB2	0x76	0x5B	0xA2	0x49	0x6D	0x8B	0xD1	0x25
0x4	0x72	0xF8	0xF6	0x64	0x86	0x68	0x98	0x16	0xD4	0xA4	0x5C	0xCC	0x5D	0x65	0xB6	0x92
0x5	0x6C	0x70	0x48	0x50	0xFD	0xED	0xB9	0xDA	0x5E	0x15	0x46	0x57	0xA7	0x8D	0x9D	0x84
0x6	0x90	0xD8	0xAB	0x00	0x8C	0xBC	0xD3	0x0A	0xF7	0xE4	0x58	0x05	0xB8	0xB3	0x45	0x06
0x7	0xD0	0x2C	0x1E	0x8F	0xCA	0x3F	0x0F	0x02	0xC1	0xAF	0xBD	0x03	0x01	0x13	0x8A	0x6B
0x8	0x3A	0x91	0x11	0x41	0x4F	0x67	0xDC	0xEA	0x97	0xF2	0xCF	0xCE	0xF0	0xB4	0xE6	0x73
0x9	0x96	0xAC	0x74	0x22	0xE7	0xAD	0x35	0x85	0xE2	0xF9	0x37	0xE8	0x1C	0x75	0xDF	0x6E
0xA	0x47	0xF1	0x1A	0x71	0x1D	0x29	0xC5	0x89	0x6F	0xB7	0x62	0x0E	0xAA	0x18	0xBE	0x1B
0xB	0xFC	0x56	0x3E	0x4B	0xC6	0xD2	0x79	0x20	0x9A	0xDB	0xC0	0xFE	0x78	0xCD	0x5A	0xF4
0xC	0x1F	0xDD	0xA8	0x33	0x88	0x07	0xC7	0x31	0xB1	0x12	0x10	0x59	0x27	0x80	0xEC	0x5F
0xD	0x60	0x51	0x7F	0xA9	0x19	0xB5	0x4A	0x0D	0x2D	0xE5	0x7A	0x9F	0x93	0xC9	0x9C	0xEF
0xE	0xA0	0xE0	0x3B	0x4D	0xAE	0x2A	0xF5	0xB0	0xC8	0xEB	0xBB	0x3C	0x83	0x53	0x99	0x61
0xF	0x17	0x2B	0x04	0x7E	0xBA	0x77	0xD6	0x26	0xE1	0x69	0x14	0x63	0x55	0x21	0x0C	0x7D

Fig. 4. Inverse Sbox for 8 bits word size

original Rijndael it is used a substitution table called *S-Box*. This S-Box is represented in the figure 3 and there is also an inverse of it in figure 4.

From the programmatic point of view the use of those boxes makes it so simple. Because the wordsize is 8 bits, by splitting the data to transform in two parts of 4 bits (1 hexadecimal digit) you can get the row and the column, taking the value in the cell as the value of the substitution. In the decipher operation, is used the inverse of the box, and with the same procedure of split the word and find the coordinates, but now with the inverse S-Box, the value you get back is the original data.

As an example, to transform the data `0x39` localise the cell in: row `0x3`, column `0x9`; and change the state matrix value with `0x12`. In the decipher procedure the transformation will be from the value `0x12`, reading: the row `0x1`, column `0x2`; the cell have the value `0x39`, the original of this example. Check any other example using figures 3 and 4 to do it and undo.

A schematic of how this step can be visualized if in the figure 5, but this drawing (and the *S-Box* way) can be only used in the case that the word size w has an even number of bits.

But this tool of the *S-Box* is a faster way to compose two transformations in one and with not much computation (the big thing is pre-computed). There can be implementations that have more memory limitations than cpu, and can make this transformation analytically better than maintain those tables (the *S-Box* and its inverse), but for the generalization of the *Rijndael* for new word sizes, at least, we must be capable to build those boxes for those new sizes.

The first transformation (called g) is to compute the multiplicative inverse in the polynomial field $\mathbb{F}_{2^w}$, where w is the wordsize ($w = 8$ in the original *Rijndael* that has become *AES*).

$$g : x \to y = x^{-1} \in \frac{\mathbb{F}_{2^w}[z]}{m(z)} \tag{10}$$

Note that the inverse operation $g^{-1}$of the function g is itself.

Using the polynomial representation of the word on each cell of the state matrix (that is considering those elements of the word, as coefficients in the field $\mathbb{F}_2$ on a polynomial field where those polynomials are modulo an irreducible $m(z)$, in the original *Rijndael*: $m(z) = z^8 + z^4 + z^3 + z + 1$ with binary representation `0b100011011=0x11B`).

The second transformation (called f) is an affine transformation over the polynomial field $\mathbb{F}_{2^w}$. In the original Rijndael is (where $w = 8$):

$$b'_i = (b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8}) \oplus c_i \tag{11}$$

Where b is the byte to be transformed, denoting the vector a as the chosen elements from b with the value `0xF1=0b11110001` and c is a fix value `0x63=0b01100011`. This transformation can be expressed as a matrix operation as an affine transformation of a polynomial product followed by an `xor` with a constant:

$$b' = f(b) \Longleftrightarrow \begin{bmatrix} b'_7 \\ b'_6 \\ b'_5 \\ b'_4 \\ b'_3 \\ b'_2 \\ b'_1 \\ b'_0 \end{bmatrix} = \begin{bmatrix} 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix} \times \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{12}$$

where the vector a has been converted to a A as a *Maximum Separable Distance* (MDS) matrix.

1. TODO: "*From where a and c comes from? From where they have been selected? Which properties they have and give is crucial to determine the ones for other word sizes and to analyse if any other proposal is good enough.*"

Because all the *Rijndael* operations must be invertible, and the $g(x)$ is self-inverse ($x = (x^{-1})^{-1}$)), it is necessary to have an inverse f^{-1} by do an inverse affine transformation of the operation f described in equation 13 and also expressed as matrix operation in 12:

$$b_i'^{-1} = (b^{-1}_{(i+1)mod8} \oplus b^{-1}_{(i+3)mod8} \oplus b^{-1}_{(i+6)mod8}) \oplus c_i^{-1} \tag{13}$$

For the inverse operation their inverse a^{-1} and c^{-1} are:

$$\begin{aligned} a^{-1} &= \texttt{0b01010010}{=}\texttt{0x52} \\ c^{-1} &= \texttt{0b00000101}{=}\texttt{0x05} \end{aligned} \tag{14}$$

Having then the inverse to the equation 12 in the matrix notation:

$$f^{-1} : \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0\,1\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,0\,1 \\ 1\,0\,1\,0\,0\,1\,0\,0 \end{bmatrix} \times \begin{bmatrix} b'_7 \\ b'_6 \\ b'_5 \\ b'_4 \\ b'_3 \\ b'_2 \\ b'_1 \\ b'_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{15}$$

The *S-Box* can be build, then, from $S(z) = f(g(z))$ and the inverse $S^{-1}(z) = g^{-1}(f^{-1}(z)) = g(f^{-1}(z))$

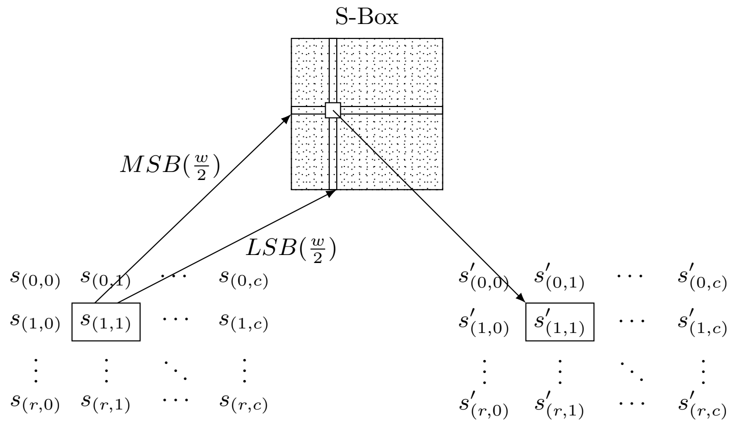

Fig. 5. Schematic diagram of the `subBytes()` transformation

4.3.1 How to build different SBoxes Using the same *wordsize* there are two different things that can be changed: b and c. The $c =$`0x63` and the product over the field of equation 11. If the option is to use another wordsize this is the unique main parameter of the original Rijndael to set a different. With a wordsize of 4 the operations will be defined over $\frac{\mathbb{F}_{2^4}[z]}{m(z)}$, over 16 the field will be $\frac{\mathbb{F}_{2^{16}}[z]}{m(z)}$, and the sub-parameters of the affine transformation must also be set up.

- `TODO:` "*how to build new S and S^{-1} for $w \neq 8$.*"
  - Why was chosen for $w = 8 \Rightarrow m(z) = z^8 + z^4 + z^3 + z + 1$?
    - Would be that `0b100011011` is the first (bigger than z^8) that has a Hamming weight of the half (by excess) of the length.
    - `TODO:` "*If it's the case, reference to the Hamming weight definition 1*"
  - Irreducible polynomial with *wordsize* degree, the firsts bigger than x^w with a Hamming weight above the half of the length:
    - $w = 2 \to m(z) = z^2 + z + 1$: Non other below but bigger than z^2 is irreducible.
    - $w = 3 \to m(z) = z^3 + z + 1$: Hamming weight of `0b1011` is 3 (above 2).
    - $w = 4 \to m(z) = z^4 + z + 1$: Hamming weight of `0b10011` is 3 (above 3).
    - $w = 5 \to m(z) = z^5 + z^2 + 1$: Hamming weight of `0b100101` is 3 (above 3).
    - $w = 6 \to m(z) = z^6 + z^4 + z^2 + z + 1$: Hamming weight of `0b1010111` is 5 (above 4).
    - $w = 7 \to m(z) = z^7 + z^4 + z^3 + z^2 + 1$: Hamming weight of `0b10011101` is 5 (above 4).
    - `TODO:` "*Rule to chose the others, specially odds wordsizes but also bigger than* 8".
  - build $g(z)$ in $\frac{\mathbb{F}_{2^w}[z]}{m(z)}$
  - build $f(z)$ and $f^{-1}(z)$ in $\frac{\mathbb{F}_{2^w}[z]}{m(z)}$
    - How to chose the circulant matrix from b of equation 11 used in equation 12 and the c (and also for the inverse)?
- `TODO:` "*summarize the S and S^{-1} using $w = 2$, $w = 4$ and $w = 16$ in their* S-Box*es and their inverse Boxes.*"

4.4 shiftRows

Now we are over the second operation of a *Rijndael* round –remember figure 1–. This operation is a transposition, and again, in Shannon's terms this provides *diffusion* to the schema. This operation takes the state matrix in rows to do a cyclic shift to the left over its cells, each row as many times as its index. Row n cyclically shifted its cells n times as the figure 6 represents for 4 rows –as the classic *Rijndael* is–. As mention in [7] (section 3.4.2) the assignment of number of offsets per rows is arbitrary, and they are done in this order for simplicity.

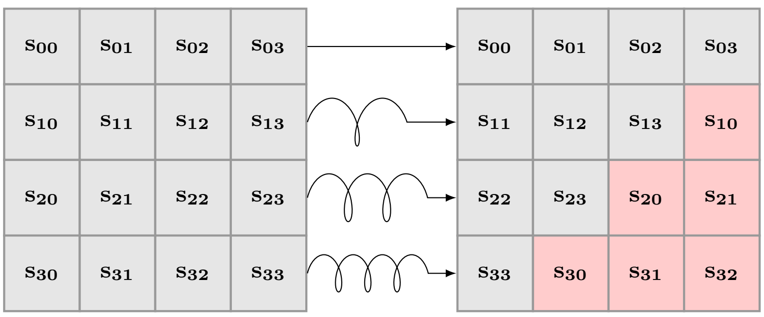

Fig. 6. Schematic diagram of the `shiftRows()` transformation simplified to four rows and four columns.

Its inverse operation is quite simple as the operation itself is, only that this time the cyclic shift is to the right the same number of times that was before to the left.

About the generalization of the *Rijndael* schema, this operation does not show special hassles. There is only one restriction that makes a constrain in the generalization: there must not be two different rows that does the same number of shifts. And to complain that the number of rows must be equally or greater than the number or columns.

4.5 mixColumns

This is the third operation of the *Rijndael* round, and as it is a permutation in terms of Shannon this means that the step provides *confusion* to the schema. In this case –and this is different than `subBytes()` from section 4.3– this transformation is lineal.

This would the most complicated step in the *Rijndael* under mathematical terms. This operation takes the columns of the states matrix and interprets them as elements in a polynomial ring. This mathematics has been introduced in section 2 with an example in 2.2. A columns represents an element of a polynomial ring $\frac{\mathbb{F}_{2^w}[x]}{l(x)}$ (with degree the number of rows: $gr(l(x)) = \#rows$), where each of its cells are elements of a polynomial field $\mathbb{F}_{2^w} = \frac{\mathbb{F}_2[z]}{m(z)}$ (with degree the word size: $gr(m(z)) = wordsize$), and all the columns were operated with another element of the ring. This element, denoted by $c(x)$ in figure 7 must be invertible in the ring –remember that in a ring, not all the elements has inverse–.

$$\boxed{\text{mixColumns()}}$$

$$\begin{array}{c|c|ccc} s_{(0,0)} & s_{(0,1)} & \cdots & s_{(0,c)} \\ s_{(1,0)} & s_{(1,1)} & \cdots & s_{(1,c)} \\ \vdots & \vdots & \ddots & \vdots \\ s_{(r,0)} & s_{(r,1)} & \cdots & s_{(r,c)} \end{array} \qquad s'(x) = c(x) \otimes s(x) \qquad \begin{array}{c|c|ccc} s'_{(0,0)} & s'_{(0,1)} & \cdots & s'_{(0,c)} \\ s'_{(1,0)} & s'_{(1,1)} & \cdots & s'_{(1,c)} \\ \vdots & \vdots & \ddots & \vdots \\ s'_{(r,0)} & s'_{(r,1)} & \cdots & s'_{(r,c)} \end{array}$$

$$s(x) = s_{(0,1)}x^{c-1} + s_{(1,1)}x^{c-2} + \cdots + s_{(r,1)}x^{c-c+1}$$
$$s(x), c(x), s'(x) \in \frac{\mathbb{F}_{2^w}[x]}{m'(x)} \text{ with } m'(x) \text{ reductible and degree } c$$
$$s_{(i,j)} \in \frac{\mathbb{F}_2[z]}{m(z)} \text{ with } m(z) \text{ irreductible and degree } w$$

Fig. 7. Diagram of the `mixColumns()` operation over the polynomial ring with coefficients in a polynomial field. Invert the `mixColumns()` is operate with $c^{-1}(x) = d(x)$ in the polynomial ring

The inverse of this operation requires to inverse the polynomial $c(x)$ to do the same operation in the polynomial ring to reverse the column on the state matrix. As long as this $c(x)$ polynomial is an element of a ring, it must be chosen carefully because not all the elements in the ring are invertible.

4.5.1 Speeding the polynomial ring product operation In section 2.2 the operation between elements of this kind of polynomial ring that has coefficients in a binary polynomial field, has been explained. But there is an specific improvement of the *product* operation that gives a valuable advantage to speed up this step of the process.

In the specification of the *Rijndael* schema [5] is proposed the use of a circulant invertible matrix. In the `mixColumns()` operation, is set one fix element of the ring to be operated with each of the columns of the state matrix (in the interpretation of the column where each cell is one coefficient of this polynomial).

Then the fix polynomial element in the ring have set in the standard:

$$c(x) = (z+1)x^3 + (1)x^2 + (1)x + (z)$$

This is using the best notation to denote that the coefficients on the polynomial ring are elements of a polynomial field. The polynomial field have binary coefficients, then those polynomials can be shorted using a binary notation. Like $(z+1) = \mathtt{0b11} = \mathtt{0x3}$ and other like $(z^3+z+1) = \mathtt{0b1011} = \mathtt{0xB}$. Then this $c(x)$ can be shorted represented by: $c(x) = \mathtt{0x3}x^3 + \mathtt{0x1}x^2 + \mathtt{0x1}x + \mathtt{0x2}$. This polynomial element is co-prime to the modulo (x^4+1) and therefore has an inverse in the ring used to revert the `mixColumns()`: $c^{-1}(x) = \mathtt{0xB}x^3 + \mathtt{0xD}x^2 + \mathtt{0x9}x + \mathtt{0xE} = d(x)$.

The matrix multiplication of this polynomial ring operation can be written as:

$$\begin{bmatrix} s'_{(0,i)} \\ s'_{(1,i)} \\ s'_{(2,i)} \\ s'_{(3,i)} \end{bmatrix} = \begin{bmatrix} z & z+1 & 1 & 1 \\ 1 & z & z+1 & 1 \\ 1 & 1 & z & z+1 \\ z+1 & 1 & 1 & z \end{bmatrix} \begin{bmatrix} s_{(0,i)} \\ s_{(1,i)} \\ s_{(2,i)} \\ s_{(3,i)} \end{bmatrix} \tag{16}$$

4.5.2 mixColumns with different number of rows

- TODO: "*Different number of rows means different degree on the polynomial ring than a column means*"
- TODO: "*How to choose a composited polynomial of the wanted degree? How to rule this to have it standardised and no need to deal with it to advice on the decryption*"
- TODO: "*How to choose an invertible element* $c(x)$*? How to standardised to avoid extra information requirement to the decipher?*"

4.6 addRoundKey

Even this is the first operation in in the *Rijndael* schema (review figure 1), it is considered the last of the four of one round. This operation is a substitution of the elements in the cells of the state matrix, and because of that, on the Shannon's view this provides *confusion* to the schema. This is the only operation in the *Rijndael* that interacts with the expanded key, and the operation is a *bitwise xor*, the addition of the very basic bricks below *Rijndael* maths, the binary field $\mathbb{F}_2$.

- TODO: "*Explain why the state matrix is taken by columns if the* xor *is for bit elements? It's nothing more than simplicity with the common 32 bit architecture*"

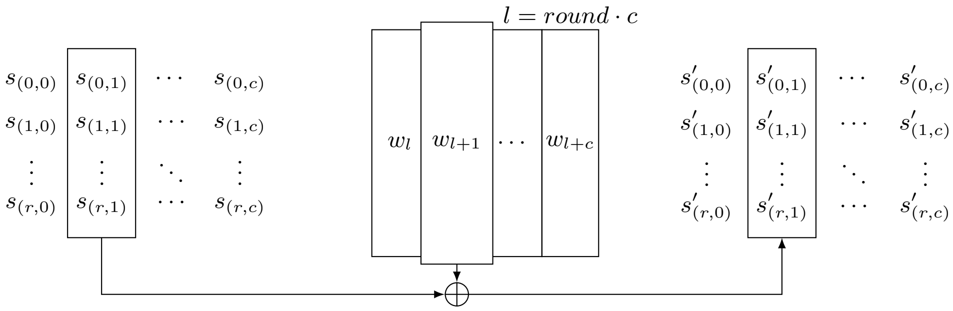

Fig. 8. Diagram of the addRoundKey()

5 New useful sizes for Rijndael

- TODO: "*With the newer architectures (64bits) which parameter changes can improve the cost of the* Rijndael *? [12] In* 64 *bits arch, a block size doubles to* 256 *in two ways:*"
  - *nRows=8* $\Rightarrow (\mathbb{F}_{2^8})^8$
  - *wSize=16* $\Rightarrow (\mathbb{F}_{2^{16}})^4$
- Small sizes of *Rijndael* vs. lightweight alternatives (like *Present*: $64b$ block with keys between $80b$ and $128b$.
- Check the AES contest tests with those new sizes

6 Parameter combinations

- TODO: "*different parameter combinations*" can produce the same block (and key) sizes. What can help on the option chose?
- TODO: "*Remember, from* `shiftRows()` *section 4.4, the number of rows must be equal or greater than number of columns.*"
- TODO: "*Can the wordsize be smaller than #rows? It looks that no restriction in this. There is no relation between the degree of the elements in of the polynomial ring* $\frac{\mathbb{F}_{2^w}[x]}{l(x)}$*, with the degree of the coefficients that are elements on a polynomial field* $\frac{\mathbb{F}_2[z]}{m(z)}$"
- TODO: "*Number of Rounds: security and performance*"

7 Conclusions

- TODO: "*This paper has proposed a new way to use the* Rijndael *, but it is not the last word as never is said in cryptography.*"
- TODO: "*The implications of the known attacks to the* Rijndael *has been studied in section 5 in the sense that was studied the original algorithm, but it is has not yet review the degree to which will affect other attacks proposed over the time.*"
- TODO: "*This proposal, with the required calm and leisurely revision that cryptography needs, can be used to extend the live of this algorithm over the time like in asymmetric algorithms the increase of key size is used instead of discard completely the algorithm. This as long as the maths behind keep valid.*"
- TODO: "*Block encryption modes*"

References

1. P. FIPS, "46-3: Data encryption standard (des)," *National Institute of Standards and Technology*, vol. 25, no. 10, 1999.
2. J. Schaad and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm." RFC 3394 (Informational), Sept. 2002.
3. J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1998.
4. J. Daemen and V. Rijmen, "The block cipher rijndael," in *Proceedings of the The International Conference on Smart Card Research and Applications*, CARDIS '98, (London, UK, UK), pp. 277–284, Springer-Verlag, 2000.
5. J. Daemen and V. Rijmen, "Aes proposal: Rijndael version 2," 1999.
6. "Specification for the advanced encryption standard (aes)." Federal Information Processin Standards Publication 197, 2001.
7. J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.
8. J. Nechvatal, E. Barker, L. Bassham, W. Burr, and M. Dworkin, "Report on the development of the advanced encryption standard (aes)," tech. rep., DTIC Document, October 2000.
9. C. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal, Vol 28, pp. 656–715*, 1949.
10. A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes." Cryptology ePrint Archive, Report 2011/449, 2011. http://eprint.iacr.org/.
11. A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full aes-192 and aes-256." Cryptology ePrint Archive, Report 2009/317, 2009. http://eprint.iacr.org/.
12. J. Daemen and V. Rijmen, "Efficient block ciphers for smartcards," in *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 1999.