Generalised Rijndael

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

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

October 27, 2012
github.Papers: 2012-10-27 (revision f43e7b4)

Abstract. [3] Here will be the abstract

Keywords: Cryptography, Symmetric, Rijndael

1 Introduction

- `TODO`: Short view on the symmetric algorithms history
  - Previous to AES was the *Data Encryption Standard*, 56 bit key length and 64 bit block size, as the standard but together with other block ciphers in 64 bit block and key length.
- `TODO`: Review on the AES contest
  - 1997 NIST announce that is seeking a replacement for the DES.
  - 15 "opponents" (plus 10 rejected due to security or efficiency reasons)
  - From the proposal on 1998 [1], [2] and the revision [3], passing by the decision in 2000 of Rijndael chosen [4].
  - and the [5] book
- `TODO`: About the future of the aes with the AESwrap (rfc3394) [6]

- `TODO`: rijndael scalability. As mentioned in [3] the standard for the AES has only 3 key sizes (128,192,256 bits), but the original specification supports blocks and keys also of lengths 160 and 224 bits. In section 12.1 the extendibility of the Rijndael is set to any multiple of 4 bytes (32 bits), with a minimum of 16 bytes (the 128 bits), but why?

- `TODO`: alternative symmetric cryptosystems. Apart than the opponents in the AES contest, thinking more in the smaller block size or, even better, variable block size.

[3] Partially founded by the Spanish project MTM20__-_____-___-__

2 Approach to the Rijndael Schema

Definition 1. *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 $m = D(k, c)$*

A pseudo-random permutation is used as a symmetric cryptosystem like Shannon have defined in [7]. Also Shannon have defined the concept of the *perfect secrecy*

Definition 2. *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 3. *For a perfect secrecy system, the distributions of the ciphers between messages in the cipher space is* computationally indistinguishable[4]*:*

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

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:

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

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 4. *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"*

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

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 [8]. 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*.

In the case of the key sizes 192 and 256 of the aes, and due to a weakness on the design of the key expansion, but this will be explained in section 3.1.

– `TODO:` What gives to Rijndael the good characteristics that it has?

Out of the standard specification [4], the revision of 1999 of the Rijndael block cipher [3] 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 2.1.

2.1 Design

– `TODO:` what is in the state matrix?
– `TODO:` Shannon: confusion & diffusion $\Rightarrow$ substitution & permutation [7] (a bit deep than what have said in the PRP, definition 2 about perfect secrecy.
– `TODO:` Explain the round transformation as one (composed) operation to provide diffusion and guarantee non-linear distribution.

3 Generalising the schema

3.1 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:

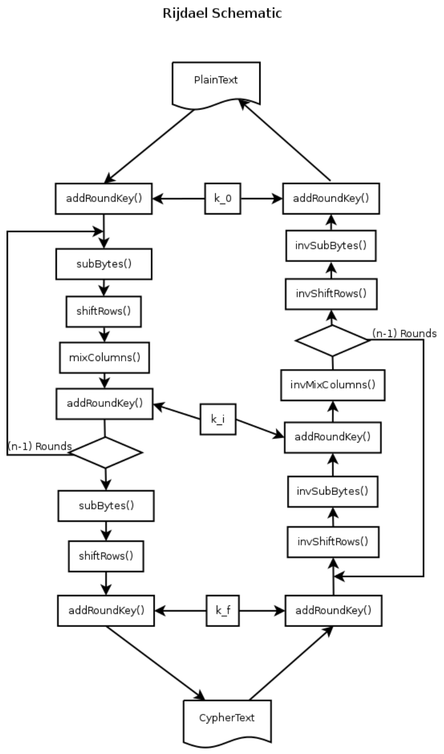


Fig. 1. rijndael diagram

Definition 5. *A* Pseudo-Random Generator *is a function that takes a seed and generates a much larger stream:*

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

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

$$\underset{k \leftarrow \mathcal{K}}{Pr}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon \qquad (2)$$

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 6 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 the 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:

$$Adv_{prg}[A,G] = \left| \underset{k \overset{R}{\leftarrow} \mathcal{K}}{Pr} [A(G(k)) = 1] - \underset{r \overset{R}{\leftarrow} \{0,1\}^n}{Pr} [A(r) = 1] \right| \in [0,1] \qquad (3)$$

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 7. *Given $G : k \to \{0,1\}^n$, is a* secure PRG *if* $\forall$ *efficient statistical test $Adv_{prg}[A,G]$ is negligible.*

Following definition 6 about the unpredictability, the ϵ is $\leqslant \frac{1}{2^{80}}$ to be considered as negligible.

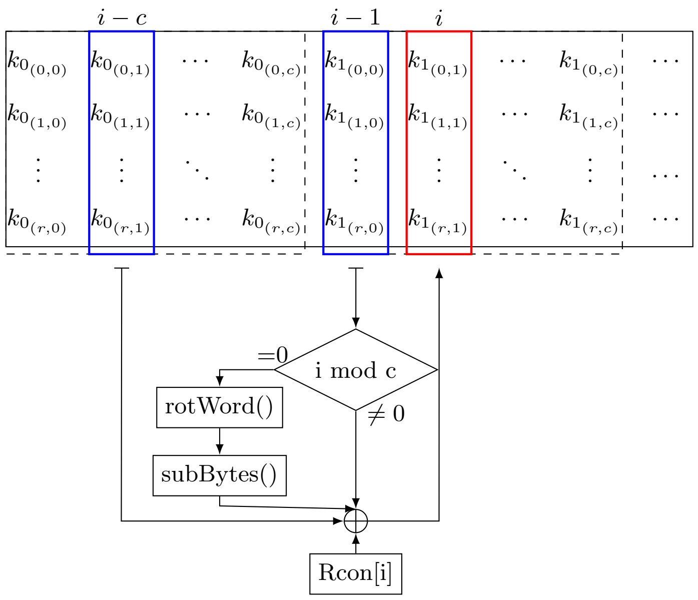
Algorithm 1 KeyExpansion

```
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 := SubWord(RotWord(temp)) $\oplus$ Rcon[i/nColumns]
10:     else
11:         temp := SubWord(temp)
12:     end if
13:     $k_{out}[i]$ := $k_{out}[i-nColumns]$ $\oplus$ temp
14:     i++
15: end while
```

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[5] takes, further than the seed key, the number of *rounds*, the number of *rows* and *columns*, and also the *word size*.

[5] This algorithm has been taken from the version 2 of the AES proposal [3] that is the same than becomes the NIST standard [4]. But is different than the book from the Rijndael authors [5], published later than the standard

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.



$i-c$
$i-1$
i
$k0_{(0,0)}$ $k0_{(0,1)}$ $\cdots$ $k0_{(0,c)}$ $k1_{(0,0)}$ $k1_{(0,1)}$ $\cdots$ $k1_{(0,c)}$ $\cdots$
$k0_{(1,0)}$ $k0_{(1,1)}$ $\cdots$ $k0_{(1,c)}$ $k1_{(1,0)}$ $k1_{(1,1)}$ $\cdots$ $k1_{(1,c)}$ $\cdots$
$k0_{(r,0)}$ $k0_{(r,1)}$ $\cdots$ $k0_{(r,c)}$ $k1_{(r,0)}$ $k1_{(r,1)}$ $\cdots$ $k1_{(r,c)}$ $\cdots$
i mod c
=0
$\neq 0$
rotWord()
subBytes()
Rcon[i]

Fig. 2. Block diagram of the iterative construction of the *Rijndael Key Expansion* as a *PseudoRandomGenerator*, PRG

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.

– `TODO:` subBytes() is used here (then the SBox) but will be explained deeper in section 3.3.

- `TODO`: What means to have different number of columns in the message than in the key matrix representation.
- `TODO`: Explain what is the proposal of the Rcon matrix (or as a recursive function).

An attack to the *PRG* of the Rijndael is described in [9] 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*[6]. The recover time of this attack is around 2^{99} that is still far away from a weakness to be worried to untrust the algorithm. Also avoiding to use related keys, this attack would not apply.

3.2 Rounds

In the AES proposal of the Rijndael [3] (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

(4)

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:

$$N_r = max(N_k, N_b) + 6 \tag{5}$$

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.

- `TODO`: But, with this, there isn't a proof of why those sizes yet.

3.3 subBytes

This transformation is a non-linear substitution of each word in the *state* matrix. In the 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 is so simple. Because the wordsize is 8 bits, by splitting the data to transform in two parts of 4 bits 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

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

	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

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. 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*).

$$g : a \to b = a^{-1} \in \frac{\mathbb{F}_{2^w}[z]}{m(z)} \tag{6}$$

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 irreductible $m(z)$, 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 $w = 8$):

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

Where b is the byte to be transformed and c is a fix value `0x63=0b01100011`. This transformation can be expressed as a matrix operation:

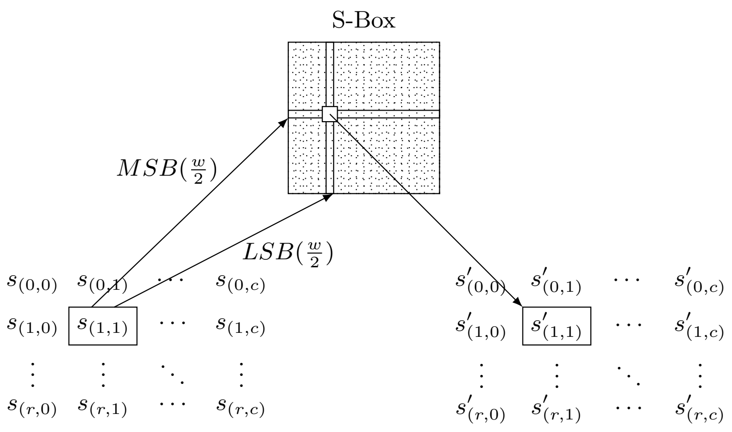
$$f : \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \\ 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{8}$$

Because all the *Rijndael* operations must be invertible, and the $g(a)$ is self-inverse ($a = (a^{-1})^{-1}$)), it is necessary to have an inverse f^{-1} by do an inverse affine transformation of the operation f described in equation 8:

$$f^{-1} : \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 0\,1\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,0\,1 \\ 1\,0\,1\,0\,0\,1\,0\,0 \end{bmatrix} \times \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{9}$$

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

How to build different SBoxes Using the same *wordsize* there are two different things that can be changed: the `0x63` and the product over the field of equation 7. 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 subparameters of the affine transformation must also be set up.



S-Box
$MSB(\frac{w}{2})$
$LSB(\frac{w}{2})$
$s_{(0,0)}$ $s_{(0,1)}$ $\cdots$ $s_{(0,c)}$
$s_{(1,0)}$ $s_{(1,1)}$ $\cdots$ $s_{(1,c)}$
$s_{(r,0)}$ $s_{(r,1)}$ $\cdots$ $s_{(r,c)}$
$s'_{(0,0)}$ $s'_{(0,1)}$ $\cdots$ $s'_{(0,c)}$
$s'_{(1,0)}$ $s'_{(1,1)}$ $\cdots$ $s'_{(1,c)}$
$s'_{(r,0)}$ $s'_{(r,1)}$ $\cdots$ $s'_{(r,c)}$

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

- `TODO`: how to build new S and S^{-1} for $w \neq 8$.
  - 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 of the equation 8 and the c (and also for the inverse)?
- `TODO`: summarize the S and S^{-1} using $w = 4$ and $w = 2$ in their S-Boxes and their inverse Boxes.

3.4 shiftColumns

- `TODO`: What this means mathematically, independently to the parameters #rows, #columns, wordsize

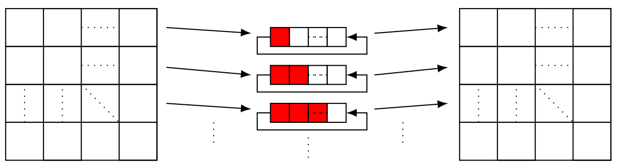
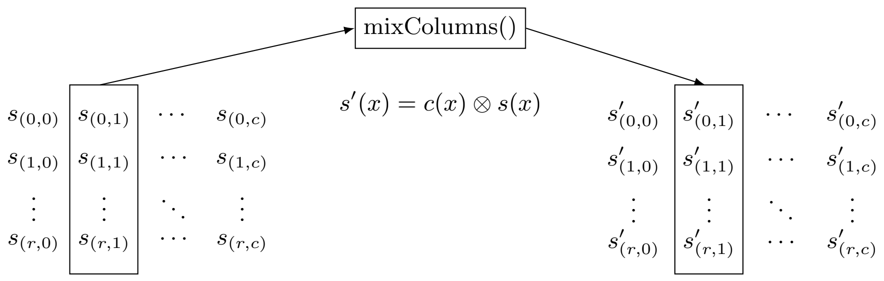


Fig. 6. Schematic diagram of the shiftColumns() transformation

3.5 mixColumns

- `TODO`: What this means mathematically? And what implies the changes on the parameters #rows, #columns, wordsize
- `TODO`: polynomial ring, where the coeficients are elements from a binary polynomial field $\frac{\mathbb{F}_{2^x}[z]}{m(z)}$, $ord(m) = \#rows$



$$\begin{array}{cccc} s_{(0,0)} & s_{(0,1)} & \cdots & s_{(0,c)} \\ s_{(1,0)} & s_{(1,1)} & \cdots & s_{(1,c)} \\ \vdots & \vdots & \ddots & \vdots \\ s_{(r,0)} & s_{(r,1)} & \cdots & s_{(r,c)} \end{array} \qquad \begin{array}{cccc} s'_{(0,0)} & s'_{(0,1)} & \cdots & s'_{(0,c)} \\ s'_{(1,0)} & s'_{(1,1)} & \cdots & s'_{(1,c)} \\ \vdots & \vdots & \ddots & \vdots \\ s'_{(r,0)} & s'_{(r,1)} & \cdots & s'_{(r,c)} \end{array}$$

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

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

3.6 Operate in a polynomial ring, with coeficients in a polynomial field

The polynomial ring used in the Rijndael schema is denoted:

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

where $m(x)$ is a composited polynomial with the same degree than the c columns parameter. This describes a polynomial ring. The coefficients of this polynomial ring are elements of a polynomial field $\mathbb{F}_{2^n}$, where this notation is a shorter of

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

and in here, the polynomial $m(z)$ is an irreductible with the same degree than the wordsize (w).

An improvement of the modular operations in the polynomial ring, in the specification of the rijndael schema [3] is proposed the use of a circulant invertible matrix. In *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:

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

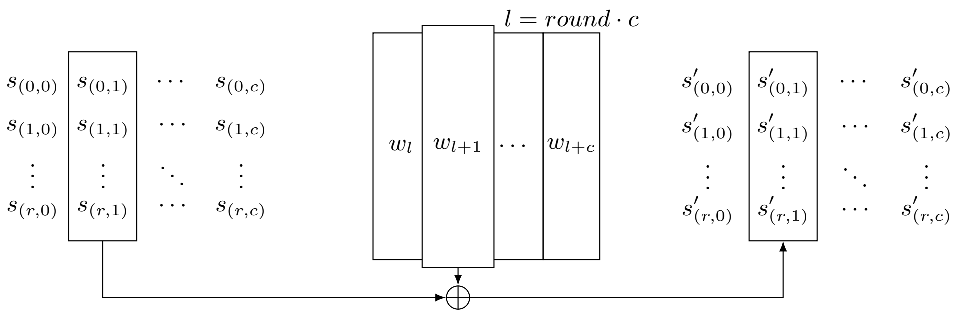
This is using the best notation to denote that the coefficients on the polynomial ring are elements of a polynomial field. The polynomial field have binary coefficients, then those polynomials can be shorted using a binary notation. Like $(z+1) = \texttt{0b11} = \texttt{0x3}$ and other like $(z^3+z+1)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

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

$$\begin{bmatrix} s'_{(0,i)} \\ s'_{(1,i)} \\ s'_{(2,i)} \\ s'_{(3,i)} \end{bmatrix} = \begin{bmatrix} z & z+1 & 1 & 1 \\ 1 & z & z+1 & 1 \\ 1 & 1 & z & z+1 \\ z+1 & 1 & 1 & z \end{bmatrix} \begin{bmatrix} s_{(0,i)} \\ s_{(1,i)} \\ s_{(2,i)} \\ s_{(3,i)} \end{bmatrix} \tag{10}$$

1. `TODO`: Does this circulant matrix works with $c \neq 4$?

3.7 addRoundKey



$l = round \cdot c$
$s_{(0,0)}$ $s_{(0,1)}$ $\cdots$ $s_{(0,c)}$
$s_{(1,0)}$ $s_{(1,1)}$ $\cdots$ $s_{(1,c)}$
$s_{(r,0)}$ $s_{(r,1)}$ $\cdots$ $s_{(r,c)}$
w_l w_{l+1} $\cdots$ w_{l+c}
$s'_{(0,0)}$ $s'_{(0,1)}$ $\cdots$ $s'_{(0,c)}$
$s'_{(1,0)}$ $s'_{(1,1)}$ $\cdots$ $s'_{(1,c)}$
$s'_{(r,0)}$ $s'_{(r,1)}$ $\cdots$ $s'_{(r,c)}$

Fig. 8. Diagram of the addRoundKey()

4 Parameter combinations

- different parameter combinations can produce the same block (and key) sizes. What can help on the option chose?

5 New useful sizes for Rijndael

– `TODO:` With the newer architectures (64bits) which parameter changes can improve the cost of the rijndael? [10]

References

1. J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1998.
2. 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.
3. J. Daemen and V. Rijmen, "Aes proposal: Rijndael version 2," 1999.
4. "Specification for the advanced encryption standard (aes)." Federal Information Processin Standards Publication 197, 2001.
5. J. Daemen and V. Rijmen, *The Design of Rijndael.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.
6. J. Schaad and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm." RFC 3394 (Informational), Sept. 2002.
7. C. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal, Vol 28, pp. 656–715*, 1949.
8. A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes." Cryptology ePrint Archive, Report 2011/449, 2011. http://eprint.iacr.org/.
9. 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/.
10. 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.