Seguridad y Protección de Sistemas Informáticos

Fco. Javier Lobillo Borrero

Departamento de Álgebra, Universidad de Granada

Curso 2017/2018

Índice

Índice

Sustitución monoalfabética

Denotamos por $\mathcal{A}$ un alfabeto, y $\mathcal{A}^*$ el conjunto de las cadenas sobre el alfabeto de longitud arbitraria. Una sustitución monoalfabética es una aplicación biyectiva

$$e : \mathcal{A} \to \mathcal{A},$$

que se extiende a las cadenas de forma natural

$$e : \mathcal{A}^* \to \mathcal{A}^*, [e(x_0 x_1 \cdots) = e(x_0)e(x_1)\cdots]$$

Cifrado de César

$\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$, $e : \mathcal{A} \to \mathcal{A}$ un desplazamiento cíclico a la derecha de tres posiciones. Matemáticamente

$$e : \mathbb{Z}_{23} \to \mathbb{Z}_{23}, [e(x) = x + 3 \pmod{23}].$$

Sustitución monoalfabética: Ejemplos I

Criptosistema de desplazamiento

```
sage: A = AlphabeticStrings()
sage: S = ShiftCryptosystem(A)
sage: K = 7
sage: P = A.encoding("El criptosistema de desplazamiento generaliza al cifrado de César"); P
ELCRIPTOSISTEMADEDESPLAZAMIENTOGENERALIZAALCIFRADODECSAR
sage: C = S.enciphering(K,P); C
LSJYPWAVZPZALTHKLKLZWSHGHTPLUAVNLULYHSPGHHSJPMYHKVKLJZHY
sage: S.deciphering(K,C)
ELCRIPTOSISTEMADEDESPLAZAMIENTOGENERALIZAALCIFRADODECSAR
sage: pdict = S.brute_force(C); pdict
{0: LSJYPWAVZPZALTHKLKLZWSHGHTPLUAVNLULYHSPGHHSJPMYHKVKLJZHY,
1: KRIXOVZUYOYZKSGJKJKYVRGFGSOKTZUMKTKXGROFGGRIOLXGJUJKIYGX,
2: JQHWNUYTXNXYJRFIJIJXUQFEFRNJSYTLJSJWFQNEFFQHNKWFITIJHXFW,
...
6: FMDSJQUPTJTUFNBEFEFTQMBABNJFOUPHFOFSBMJABBMDJGSBEPEFDTBS,
7: ELCRIPTOSISTEMADEDESPLAZAMIENTOGENERALIZAALCIFRADODECSAR,
8: DKBQHOSNRHRSDLZCDCDROKZYZLHDMSNFDMDQZKHYZZKBHEQZCNCDBRZQ,
... }
```

Sustitución monoalfabética: Ejemplos II

Sustitución monoalfabética

```
sage: E = SubstitutionCryptosystem(A)
sage: P = E.encoding('Este es un ejemplo de una sustitucion monoalfabetica general. Como pretendemos realizar un
primer ejemplo de ataque encaminado a la ruptura vamos a utilizar un ejemplo de cierto tamano.')
sage: K = A('JUCFLPKQWGVTXYBRMDZIEOAHSN'); K
JUCFLPKQWGVTXYBRMDZIEOAHSN
sage: e = E(K)
sage: C = e(P); C
LZILLZEYLGLXRTBFLEYJZEZIWIECWBYXBYBJTPJULIWCJKLYLDJTCBXBRDLILYFLXBZDLJTW
NJDEYRDWXLDLGLXRTBFLJIJMELLYCJXWYJFBJTJDERIEDJOJXBZJEIWTWNJDEYLGLXRTBFLC
WLDIBIJXJYB
sage: E.deciphering(K,C)
ESTEESUNEJEMPLODEUNASUSTITUCIONMONOALFABETICAGENERALCOMOPRETENDEMOSREALI
ZARUNPRIMEREJEMPLODEATAQUEENCAMINADOALARUPTURAVAMOSAUTILIZARUNEJEMPLODEC
IERTOTAMANO
```

Sustitución monoalfabética: Ejemplos III

```
sage: C.frequency_distribution()
Discrete probability space defined by {P: 0.00645161290322581, W: 0.0580645161290323,
B: 0.0838709677419355, X: 0.0645161290322581, C: 0.0322580645161290, I: 0.0645161290322581,
D: 0.0645161290322581, J: 0.122580645161290, E: 0.0645161290322581, K: 0.00645161290322581,
F: 0.0322580645161290, L: 0.148387096774194, G: 0.0193548387096774, R: 0.0387096774193548,
M: 0.00645161290322581, N: 0.0129032258064516, Y: 0.0709677419354839, T: 0.0516129032258065,
O: 0.00645161290322581, Z: 0.0387096774193548, U: 0.00645161290322581}
```

Sustitución polialfabética

Dadas t biyecciones en el alfabeto $\mathcal{A}$

$$e_0, \ldots, e_{t-1} : \mathcal{A} \to \mathcal{A},$$

se define una sustitución polialfabética como la aplicación biyectiva

$$e : \mathcal{A}^* \to \mathcal{A}^*, [e(x_0 \ldots x_{t-1} x_t \ldots x_{2t-1} x_{2t} \ldots) = e_0(x_0) \ldots e_{t-1}(x_{t-1}) e_0(x_t) \ldots e_{t-1}(x_{2t-1}) e_0(x_{2t}) \ldots].$$

Cifrado de Vigenere

Las biyecciones $e_i : \mathcal{A} \to \mathcal{A}$ son cifrados de desplazamiento, determinados por las posiciones, empezando por 0, que ocupan las letras de una palabra clave. Por ejemplo, una palabra clave `ABCBA` se corresponde con las funciones

$$
\begin{aligned}
e_0(x) &= x \pmod{26},\\
e_1(x) &= e_3(x) = x + 1 \pmod{26},\\
e_2(x) &= x + 2 \pmod{26},\\
e_3(x) &= x + 1 \pmod{26}\\
e_4(x) &= x \pmod{26}.
\end{aligned}
$$

Cifrado de Vigenere: Ejemplo I

```
sage: A = AlphabeticStrings()
sage: E = VigenereCryptosystem(A,6)
sage: K = A('JAVIER')
sage: P = A.encoding("Rugby union, or simply rugby, is a contact team sport which originated in England in the first
half of the 19th century.[3] One of the two codes of rugby football, it is based on running with the ball in hand. In
its most common form, a game is between two teams of 15 players (two more than rugby league) using an oval-shaped
ball on a rectangular field with H-shaped goalposts on each try line. In 1845, the first football laws were written
by Rugby School pupils; other significant events in the early development of rugby include the Blackheath Club's
decision to leave the Football Association in 1863 and the split between rugby union and rugby league in 1895.
Historically an amateur sport, in 1995 restrictions on payments to players were removed, making the game openly
professional at the highest level for the first time.[4]")
sage: C = E.enciphering(K,P); C
AUBJCLWIJVSIBIHXPPAUBJCZBAXWRKJCOBIRVSKWVKFHDKLFAIBQRRCEYQRVWGGIRURNOPIWRRNBLRUFJNXYNTCK
IECUMGSENOABLVCWJKSUNSJNVLPBTNSFCBVTPZCINJEJNDJVVLWNDVKNRTCBLVKAGTMEQAILMERTNUSJCCJUQFWF
JZQRPAHMMJKEOEIVWTRWXVJMNWJGUATMVJCWJUSINTCIRIDGWGPVJGPMYJRNBIRFEAGALRYEYJECUOIIVVLTVVKL
UAMNMVUDRQXYQSCITVMGJIPGXSOASENAXPXIHLDVIZWTCMJZASONSFCBVTPCJWNEIINWMQXKNNWGVLPBTAGYXOGX
YGRLNWXYNRNQKERFDKEECEQMRKBIIBLVNAMTCUNVZTSGVEIBSWAUBJCZWCGCHVCHZJPRLKCMEKQCGCFJMEXQWZXN
OWPVJVZBLVOOJBFRULVAWFLIVBMFWIIIRUCHZATCRTWMXNNEIZYXKYPVMFWAILVLPBTTIRPUZQRYRSOWVZLAGTCR
WAHIXVDRNXSICIIZIJCRDKXZXNNWRGJYHMRKBTJXPRHEMAAVAEMMQFEEYUEBRNBBLVPAHMSGNNGGTIXFZAWZXNVT
EKCHZPMXQENBPVEEGNSICHZNMIBTOQQV
sage: E.deciphering(K,C)
RUGBYUNIONORSIMPLYRUGBYISACONTACTTEAMSPORTWHICHORIGINATEDINENGLANDINTHEFIRSTHALFOFTHETHC
ENTURYONEOFTHETWOCODESOFRUGBYFOOTBALLITISBASEDONRUNNINGWITHTHEBALLINHANDINITSMOSTCOMMONF
ORMAGAMEISBETWEENTWOTEAMSOFPLAYERSTWOMORETHANRUGBYLEAGUEUSINGANOVALSHAPEDBALLONARECTANGU
LARFIELDWITHHSHAPEDGOALPOSTSONEACHTRYLINEINTHEFIRSTFOOTBALLLAWSWEREWRITTENBYRUGBYSCHOOLP
```

Cifrado de Vigenere: Ejemplo II

```
UPILSOTHERSIGNIFICANTEVENTSINTHEEARLYDEVELOPMENTOFRUGBYINCLUDETHEBLACKHEATHCLUBSDECISION
TOLEAVETHEFOOTBALLASSOCIATIONINANDTHESPLITBETWEENRUGBYUNIONANDRUGBYLEAGUEINHISTORICALLYA
NAMATEURSPORTINRESTRICTIONSONPAYMENTSTOPLAYERSWEREREMOVEDMAKINGTHEGAMEOPENLYPROFESSIONAL
ATTHEHIGHESTLEVELFORTHEFIRSTTIME
sage: C.frequency_distribution()
Discrete probability space defined by V: 0.0648148148148148, X: 0.0370370370370370,
J: 0.0493827160493827, L: 0.0354938271604938, N: 0.0632716049382716, Y: 0.0216049382716049,
P: 0.0324074074074074, B: 0.0432098765432099, D: 0.0138888888888889, F: 0.0246913580246914,
Q: 0.0262345679012346, H: 0.0200617283950617, S: 0.0293209876543210, U: 0.0293209876543210,
W: 0.0462962962962963, I: 0.0632716049382716, K: 0.0308641975308642, M: 0.0416666666666667,
O: 0.0200617283950617, Z: 0.0308641975308642, A: 0.0432098765432099, C: 0.0540123456790123,
E: 0.0462962962962963, G: 0.0370370370370370, R: 0.0570987654320988, T: 0.0385802469135802
sage: aux = A('')
sage: for ii in range(len(C)):
... if ii% 4 == 0:
... aux *= C[ii]
...
sage: aux.frequency_distribution()
Discrete probability space defined by V: 0.0740740740740741, X: 0.0493827160493827,
J: 0.0617283950617284, L: 0.0432098765432099, N: 0.0864197530864197, Y: 0.0308641975308642,
P: 0.0370370370370370, B: 0.0370370370370370, D: 0.0246913580246914, Q: 0.0308641975308642,
H: 0.0246913580246914, S: 0.0185185185185185, U: 0.0246913580246914, W: 0.0493827160493827,
I: 0.0493827160493827, K: 0.0123456790123457, M: 0.0555555555555556, O: 0.0308641975308642,
Z: 0.0370370370370370, A: 0.0246913580246914, C: 0.0493827160493827, E: 0.0185185185185185,
```

Cifrado de Vigenere: Ejemplo III

```
G: 0.0246913580246914, R: 0.0802469135802469, T: 0.0246913580246914
sage: aux = A('')
sage: for ii in range(len(C)):
... if ii% 6 == 0:
... aux *= C[ii]
...
sage: aux.frequency_distribution()
Discrete probability space defined by P: 0.0555555555555556, V: 0.0185185185185185,
A: 0.0555555555555556, W: 0.0925925925925926, B: 0.0462962962962963, X: 0.0555555555555556,
C: 0.138888888888889, D: 0.0185185185185185, J: 0.0555555555555556, E: 0.0277777777777778,
K: 0.0277777777777778, F: 0.00925925925925926, Q: 0.0370370370370370, L: 0.0370370370370370,
R: 0.0925925925925926, M: 0.0185185185185185, H: 0.0185185185185185, N: 0.120370370370370,
Y: 0.00925925925925926, O: 0.00925925925925926, U: 0.0555555555555556
```

Criptosistema de Hill

Como antes identificamos $\mathcal{A} \approx \mathbb{Z}_{26}$. Dada una matriz $M \in \mathcal{M}_t(\mathbb{Z}_{26})$ con inversa, la aplicación

$$e : \mathbb{Z}_{26}^t \to \mathbb{Z}_{26}^t, [e(x_0 \dots x_{t-1}) = (x_0 \dots x_{t-1})M]$$

es biyectiva, y proporciona una función de cifrado

$$e : \mathbb{Z}_{26}^* \to \mathbb{Z}_{26}^*,$$

definida dividiendo cada cadena en bloques de longitud t y multiplicando por M cada bloque.

Cifrado de Hill: Ejemplo I

```
sage: A = AlphabeticStrings()
sage: E = HillCryptosystem(A,4)
sage: K = random_matrix(IntegerModRing(26),4,4)
sage: while not(K.is_invertible()):
... K = random_matrix(IntegerModRing(26),4,4)
sage: P = A.encoding("Rugby union, or simply rugby, is a contact team sport which originated in England in the first
half of the 19th century. One of the two codes of rugby football, it is based on running with the ball in hand. In
its most common form, a game is between two teams of 15 players (two more than rugby league) using an oval-shaped
ball on a rectangular field with H-shaped goalposts on each try line. In 1845, the first football laws were written
by Rugby School pupils; other significant events in the early development of rugby include the Blackheath Club's
decision to leave the Football Association in 1863 and the split between rugby union and rugby league in 1895.
Historically an amateur sport, in 1995 restrictions on payments to players were removed, making the game openly
professional at the highest level for the first time.")
sage: C = E.enciphering(K,P); C
KAZERQLZDXAXCSXRZCVSHDLYIQYARBRVHJBFHWFNFJCAJCRRSEUFNODMXUTAFUROFXWYFTMKRLYNBWIVSJLWELAZIP
KRWMPGBEWWZGXAWYTDOQHXKAZELNZILHQQJCTIZDBWRHKDZQYZLTZYEJZCBUDKHXSCTCOXLPAPCSYEQQCPDOAMDFZC
CQAQPQHXYUSDVRTBAOSHPMAKCWIVNEPKEYBVURQDHRRPPMRWIISKUGTFBMTVJSTONMJGNTWYJUDQIOVSPFFGNGWNVU
SABASYTGLGTWHSDQSEVCBNZJRCWZFMHXSPPKOCTWXQTQIFFSYAQVPCCQYIZFNKVKKJKDQBKAZEQILVPCEGNPMDPKUM
VVDYDNMFKSVNHMDCNJJRIVSKNQOYZHINTTMPFICQQJILHDLYFQRSJCITQJOPHWXRYBIBDMREKMUVUYDTNCRGNNCNXE
BEQDARLVFNQUGOHRLCFSFQMPPZTYGKOPGZYUSDHRRPEYRILGJLKPMOJHAFICKEHUGAFMTLACAZXXDZMASBGMVQHFQZ
VIAFBGQURUFRFTRMBRBVFICQTHCLCWIVEGORNUBUTEHTKYGJJPKRGUEFMUDXUDSRZZMWUUCALPOXWZZSLLBSWCNUVS
TGNROCFTMKRLYNUIST
sage: E.deciphering(K,C)
RUGBYUNIONORSIMPLYRUGBYISACONTACTTEAMSPORTWHICHORIGINATEDINENGLANDINTHEFIRSTHALFOFTHETHCEN
TURYONEOFTHETWOCODESOFRUGBYFOOTBALLITISBASEDONRUNNINGWITHTHEBALLINHANDINITSMOSTCOMMONFORMA
```

Cifrado de Hill: Ejemplo II

```
GAMEISBETWEENTWOTEAMSOFPLAYERSTWOMORETHANRUGBYLEAGUEUSINGANOVALSHAPEDBALLONARECTANGULARFIE
LDWITHHSHAPEDGOALPOSTSONEACHTRYLINEINTHEFIRSTFOOTBALLLAWSWEREWRITTENBYRUGBYSCHOOLPUPILSOTH
ERSIGNIFICANTEVENTSINTHEEARLYDEVELOPMENTOFRUGBYINCLUDETHEBLACKHEATHCLUBSDECISIONTOLEAVETHE
FOOTBALLASSOCIATIONINANDTHESPLITBETWEENRUGBYUNIONANDRUGBYLEAGUEINHISTORICALLYANAMATEURSPOR
TINRESTRICTIONSONPAYMENTSTOPLAYERSWEREREMOVEDMAKINGTHEGAMEOPENLYPROFESSIONALATTHEHIGHESTLE
VELFORTHEFIRSTTIME
sage: C.frequency_distribution()
Discrete probability space defined by V: 0.0370370370370370,
X: 0.0277777777777778, J: 0.0308641975308642, L: 0.0354938271604938,
N: 0.0401234567901235, Y: 0.0370370370370370, P: 0.0401234567901235,
B: 0.0324074074074074, D: 0.0401234567901235, F: 0.0462962962962963,
Q: 0.0493827160493827, H: 0.0354938271604938, S: 0.0432098765432099,
U: 0.0401234567901235, W: 0.0324074074074074, I: 0.0370370370370370,
K: 0.0385802469135802, M: 0.0385802469135802, O: 0.0262345679012346,
Z: 0.0416666666666667, A: 0.0385802469135802, C: 0.0509259259259259,
E: 0.0308641975308642, G: 0.0324074074074074, R: 0.0540123456790123,
T: 0.0432098765432099
sage: aux = A('')
sage: for ii in range(len(C)):
... if ii% 4 == 0:
... aux *= C[ii]
...
sage: aux.frequency_distribution()
Discrete probability space defined by V: 0.0432098765432099,
```

Cifrado de Hill: Ejemplo III

```
X: 0.0185185185185185, J: 0.0432098765432099, L: 0.0493827160493827,
N: 0.0740740740740741, Y: 0.0185185185185185, P: 0.0432098765432099,
B: 0.0432098765432099, D: 0.0370370370370370, F: 0.0740740740740741,
Q: 0.0432098765432099, H: 0.0802469135802469, S: 0.0123456790123457,
U: 0.0370370370370370, W: 0.0308641975308642, I: 0.0370370370370370,
K: 0.0493827160493827, M: 0.0185185185185185, O: 0.0123456790123457,
Z: 0.0493827160493827, A: 0.0123456790123457, C: 0.0370370370370370,
E: 0.0308641975308642, G: 0.0123456790123457, R: 0.0370370370370370,
T: 0.0555555555555556
```

Criptosistema de permutación

Una permutación de t elementos $\sigma \in S_t$, es decir, una aplicación biyectiva $\sigma : \{0, \ldots, t-1\} \to \{0, \ldots, t-1\}$, induce otra biyección

$$e : \mathcal{A}^t \to \mathcal{A}^t, \left[e(x_0 \ldots x_{t-1}) = x_{\sigma(0)} \ldots x_{\sigma(t-1)} \right],$$

que se extiende de la manera usual a una función de cifrado

$$e : \mathcal{A}^* \to \mathcal{A}^*$$

descomponiendo cada cadena en bloques de tamaño t.

Cifrado de permutación: Ejemplo I

```
sage: A = AlphabeticStrings()
sage: E = TranspositionCryptosystem(A,10)
sage: K = E.random_key(); K
(1,9,6,8,5,10)(2,4)(3,7)
sage: P = A.encoding("Rugby union, or simply rugby, is a contact team sport which originated in England in the first
half of the 19th century. One of the two codes of rugby football, it is based on running with the ball in hand. In
its most common form, a game is between two teams of 15 players (two more than rugby league) using an oval-shaped
ball on a rectangular field with H-shaped goalposts on each try line. In 1845, the first football laws were written
by Rugby School pupils; other significant events in the early development of rugby include the Blackheath Club's
decision to leave the Football Association in 1863 and the split between rugby union and rugby league in 1895.
Historically an amateur sport, in 1995 restrictions on payments to players were removed, making the game openly
professional at the highest level for the first time.")
sage: while not(len(P) % 10 == 0):
... P *= A('X')
sage: len(P)
650
sage: C = E.enciphering(K,P); C
OBNUNIGYURRILRUYSMPONICBTOYSAGPTMCOSTEAARHHTIOWICRNADIEINTEGTAIGHNLNDNLRHFFAISTEEHHFNCTE
TOFYEUTORONTEWOESDTOCHOUYFOFRGBOSLTBBIALITNDRSNUEONAHWHNETGITINLHADALINBTTONCSISMIMOOMAR
MNFOTEBAWEMISGATTEMENWOERPYOSEFLASHMEWATOORTAGLRGEUBYNNSGEOAUINUDSPABELHAVCORLTELNAAIURN
EFGLAAHIHDASWTHLOGLESPDOAPTNCSRHOEATHNNLETIEIYTSOIBORTFFELSLRWLAWABIEWYNRTTEOBCUOHGYSRTP
SPHOUILLIIIRCFSGNESENNITTVEAYERTDLHEANNLMVTEEOPENUYFCIRGBOLEEUABDTHLLEHKUCHATCOESSNIDCIB
HEEOETLAVTATLOSLOBAFNIIOIOCATSPDEALSNTHNNEETREBTWINYIGAOBUNUEUYDALRGBNTIIUOSENHGNAYIAACL
LROESARPTURMIRTICRNESTANNIYPOSOTLTOEAPNSTMRSREEERWEYIEAONKVDMMOEMTPEHGAGEYONSFLPRETNAIHT
OALSLGSHETIHEEEFTEFHLORVXTMRXESTII
```

Cifrado de permutación: Ejemplo II

```
sage: E.deciphering(K,C)
RUGBYUNIONORSIMPLYRUGBYISACONTACTTEAMSPORTWHICHORIGINATEDINENGLANDINTHEFIRSTHALFOFTHETHC
ENTURYONEOFTHETWOCODESOFRUGBYFOOTBALLITISBASEDONRUNNINGWITHTHEBALLINHANDINITSMOSTCOMMONF
ORMAGAMEISBETWEENTWOTEAMSOFPLAYERSTWOMORETHANRUGBYLEAGUEUSINGANOVALSHAPEDBALLONARECTANGU
LARFIELDWITHHSHAPEDGOALPOSTSONEACHTRYLINEINTHEFIRSTFOOTBALLLAWSWEREWRITTENBYRUGBYSCHOOLP
UPILSOTHERSIGNIFICANTEVENTSINTHEEARLYDEVELOPMENTOFRUGBYINCLUDETHEBLACKHEATHCLUBSDECISION
TOLEAVETHEFOOTBALLASSOCIATIONINANDTHESPLITBETWEENRUGBYUNIONANDRUGBYLEAGUEINHISTORICALLYA
NAMATEURSPORTINRESTRICTIONSONPAYMENTSTOPLAYERSWEREREMOVEDMAKINGTHEGAMEOPENLYPROFESSIONAL
ATTHEHIGHESTLEVELFORTHEFIRSTTIMEXX
sage: P.frequency_distribution()
Discrete probability space defined by V: 0.00923076923076923,
X: 0.00307692307692308, L: 0.0584615384615384, N: 0.0784615384615385,
Y: 0.0276923076923077, P: 0.0230769230769231, B: 0.0292307692307692,
D: 0.0215384615384615, F: 0.0261538461538461, H: 0.0476923076923077,
S: 0.0584615384615384, U: 0.0307692307692308, W: 0.0184615384615385,
I: 0.0738461538461539, K: 0.00307692307692308, M: 0.0246153846153846,
O: 0.0784615384615385, A: 0.0753846153846154, C: 0.0261538461538461,
E: 0.106153846153846, G: 0.0323076923076923, R: 0.0553846153846154,
T: 0.0923076923076924
sage: C.frequency_distribution()
Discrete probability space defined by V: 0.00923076923076923,
X: 0.00307692307692308, L: 0.0584615384615384, N: 0.0784615384615385,
Y: 0.0276923076923077, P: 0.0230769230769231, B: 0.0292307692307692,
D: 0.0215384615384615, F: 0.0261538461538461, H: 0.0476923076923077,
```

Cifrado de permutación: Ejemplo III

```
S: 0.0584615384615384, U: 0.0307692307692308, W: 0.0184615384615385,
I: 0.0738461538461539, K: 0.00307692307692308, M: 0.0246153846153846,
O: 0.0784615384615385, A: 0.0753846153846154, C: 0.0261538461538461,
E: 0.106153846153846, G: 0.0323076923076923, R: 0.0553846153846154,
T: 0.0923076923076924
```

One-time-pad

Consiste en un cifrado de Vigenere con una clave “aleatoria” de la misma longitud, al menos, que el texto. Una versión eléctrica binaria fue desarrollada por Vernam y Mauborgne en la compañía AT&T. Matemáticamente, el mensaje y la clave son cadenas en $\mathbb{Z}_{26}^*$, siendo la clave una sucesión aleatoria. Si $m = m_0 m_1 \cdots$ y $k = k_0 k_1 \cdots$, la función de cifrado es

$$e(m) = (m_0 + k_0 \pmod{26})(m_1 + k_1 \pmod{26}) \cdots$$

Por qué un solo uso.

Si $m^{(1)} + k = c^{(1)}$ y $m^{(2)} + k = c^{(2)}$, $c^{(2)} - c^{(1)} = m^{(2)} - m^{(1)}$. Desaparece la clave y por tanto la aleatoriedad.

WWII

- Máquinas electromecánicas: Enigma, Purple, Typex, SIGABA,
- Libros de códigos.
- One-time-pad.

Índice

Definición formal I

Dados conjuntos

- $\mathcal{M}$ el conjunto se los mensajes, textos en claro o *plaintexts*,
- $\mathcal{C}$ el conjunto de los criptogramas o *cyphertexts*,
- $\mathcal{K}$ el espacio de claves o *key space*,

un cripsosistema viene definido por dos aplicaciones

$$e : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$
$$d : \mathcal{K} \times \mathcal{C} \to \mathcal{M},$$

tales que para cualquier clave $k \in \mathcal{K}$ y cualquier mensaje $m \in \mathcal{M}$,

$$d(k, e(k, m)) = m. \tag{1}$$

Definición formal II

Fijada una clave $k \in \mathcal{K}$, se suele utilizar la notación

$$\begin{aligned} e_k &: \mathcal{M} \to \mathcal{C}, \\ d_k &: \mathcal{C} \to \mathcal{M}, \end{aligned}$$

para las funciones de cifrado y descifrado. La propiedad (1) se transforma en

$$d_k(e_k(m)) = m.$$

Consideraciones

Si partimos de un texto cifrado, ¿cómo es posible asegurarnos de que nuestro criptosistema es seguro? Vamos a hacer dos consideraciones de cara a realizar un pequeño estudio de la seguridad de un criptosistema

- *"el criptosistema no debe dar más información que la estrictamente necesaria".*
- *"cada posible texto cifrado y cada posible texto sin cifrar son equiprobables".*

Si tenemos en cuenta estas dos reglas nos aseguramos de que nuestro criptosistema no contiene información redundante, y por tanto todo criptoanálisis que se realice se debe centrar únicamente en los métodos matemáticos usados en su diseño o en la potencia de cálculo que se pueda desarrollar.

Aproximación de Shannon

En su artículo “Communication Theory of Secrecy Systems”, C. E. Shannon destaca dos características que un criptosistema debe tener para no ser vulnerable a ataques estadísticos y de frecuencias:

Difusión La estructura estadística del mensaje, que produce su redundancia, se disipa en la estructura estadística de grandes combinaciones de letras del criptograma. Informalmente, un cambio en un carácter del texto en claro provocará muchos cambios en el criptograma (idealmente en la mitad sus caracteres).

Confusión La relación entre el criptograma y la clave es compleja y enmarañada. Esto quiere decir que cada carácter del criptograma depende de varios caracteres de la clave.

Desde su publicación, se ha buscado garantizar estas características en los criptosistemas propuestos.

Lo importante, desde nuestro punto de vista, es que existen herramientas para analizar la bondad de un criptosistema.

Criptografía digital

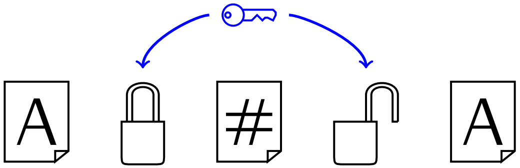
Con el desarrollo de la sociedad de la información y la digitalización de los contenidos, los criptosistemas modernos se diseñan sobre cadenas de bits, y no sobre cadenas de caracteres. Fijamos en consecuencia la siguiente notación. $\mathbb{B} = \mathbb{F}_2 = \mathbb{Z}_2$ es el álgebra de Boole con dos elementos, y en él tenemos las operaciones $\{\wedge, \vee, \oplus, \cdot, +\}$.

Los mensajes, criptogramas y claves son cadenas de bits, es decir, $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{B}^*$.

Criptosistemas simétricos

Las claves de cifrado y descifrado son “computacionalmente” equivalentes, es decir, si conocemos la clave utilizada para cifrar el mensaje, podemos descifrar el criptograma. Todos los criptosistemas clásicos son simétricos. En la actualidad los hay de dos clases:

1. Cifrados de bloque.
2. Cifrados de flujo.



Índice

Definición

Para un criptosistema por bloques, limitamos los mensajes, criptogramas y claves a cadenas de una longitud fija,

$$e : \mathbb{B}^K \times \mathbb{B}^N \to \mathbb{B}^M,$$
$$d : \mathbb{B}^K \times \mathbb{B}^M \to \mathbb{B}^N,$$

o para cada clave $k \in \mathbb{B}^K$,

$$e_k : \mathbb{B}^N \to \mathbb{B}^M,$$
$$d_k : \mathbb{B}^M \to \mathbb{B}^N.$$

Lo usual es que $N = M$, hipótesis que asumiremos mientras no indiquemos lo contrario. A este valor común se le conoce como tamaño del bloque y a K como el tamaño de la clave.

Cómo se extiende e de $\mathbb{B}^N$ a $\mathbb{B}^*$ es competencia de los modos de operación. Estos modos dependen del tamaño del bloque, no de la clave.

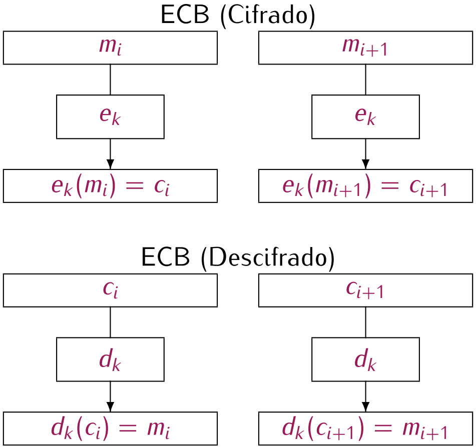
Electronic Code Book (ECB)

El mensaje se divide en bloques de N bits, es decir, $m = m_0||m_1||\cdots$ donde cada m_i tiene longitud N. Cada bloque se cifra de manera independiente. Formalmente,

$$c_i = e_k(m_i)$$

Difusión y confusión se mantienen localmente en cada bloque. Un atacante podría sustituir un bloque del criptograma por otro de texto en claro conocido.

ECB: descripción gráfica



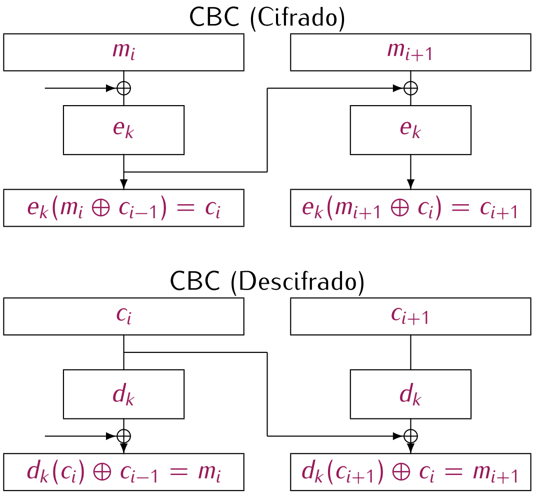
Cipher Block Chaining (CBC)

En este modo, la salida del cifrado de un bloque se suma (XOR) a la entrada del siguiente bloque:

$$c_{i+1} = e(m_{i+1} \oplus c_i)$$

Empleamos un vector de inicialización $c_0 = \mathtt{IV}$ que no es necesario mantener en secreto. Su integridad es esencial para garantizar el correcto descifrado de c_1. Si $\mathtt{IV}$ es aleatorio, un mismo texto en claro corresponderá con diferentes criptogramas en cada cifrado.

CBC: descripción gráfica



Cipher FeedBack (CFB)

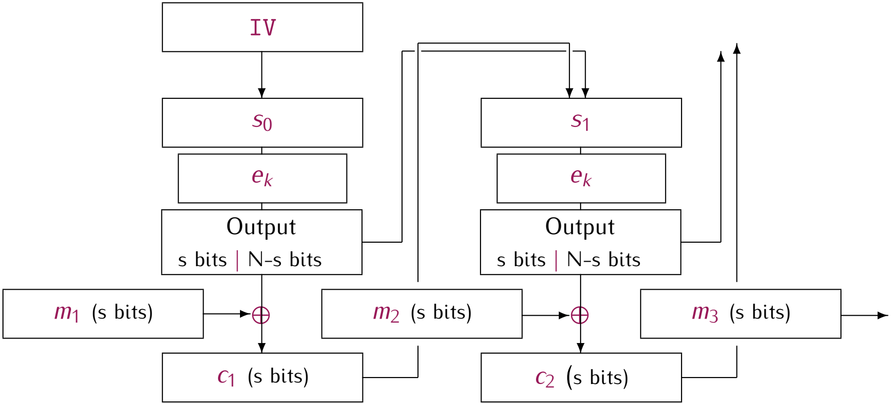
- Recordemos que N es el tamaño del bloque. Sea s un entero tal que $1 \leq s \leq N$.
- El mensaje se divide en bloques de tamaño s, que llamamos m_i, $i \geq 1$.
- Tomamos un $s_0 = \mathtt{IV}$ de longitud N. Las funciones LSB_{N-s} y MSB_s devuelven los $N - s$ bits menos significativos (por la derecha) y los s más significativos (por la izquierda).
- Una vez que hemos generado un bloque s_{i-1} lo ciframos. La parte correspondiente del criptograma es $c_i := m_i \oplus \mathrm{MSB}_s(e_k(s_{i-1}))$. Además $s_i = (\mathrm{LSB}_{N-s}(s_{i-1}), c_i)$.

Es decir,

$$
\begin{aligned}
&s_0 \\
c_1 &:= m_1 \oplus \mathrm{MSB}_s(e_k(s_0)) \\
s_1 &:= (\mathrm{LSB}_{N-s}(e_k(s_0)), c_1) \\
c_i &:= m_i \oplus \mathrm{MSB}_s(e_k(s_{i-1})) \quad \text{for } i = 2, \ldots \\
s_i &:= (\mathrm{LSB}_{N-s}(e_k(s_{i-1})), c_i) \quad \text{for } i = 2, \ldots.
\end{aligned}
$$

Cuando $s = N$, MSB_s es la identidad y LSB_{N-s} es el bloque vacío.

CFB: descripción gráfica del cifrado



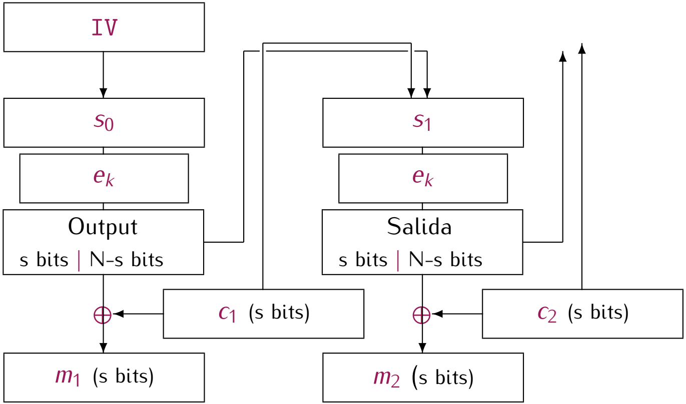
CFB: descifrado

- Se emplea el mismo vector de inicialización $s_0 = \mathtt{IV}$.
- Una vez calculado s_{i-1}, podemos calcular el siguiente bloque del mensaje mediante $m_i := c_i \oplus \mathrm{MSB}_s(e_k(s_0))$.
- De nuevo $s_i = (\mathrm{LSB}_{N-s}(s_{i-1}), c_i)$.

Formalmente:

$$
\begin{aligned}
&s_0 \\
m_1 &:= c_1 \oplus \mathrm{MSB}_s(e_k(s_0)) \\
s_1 &:= (\mathrm{LSB}_{b-s}(e_k(s_0)), c_1) \\
m_i &:= c_i \oplus \mathrm{MSB}_s(e_k(s_{i-1})) \quad \text{for } i = 2, \dots, n \\
s_i &:= (\mathrm{LSB}_{N-s}(e_k(s_{i-1})), c_i) \quad \text{for } i = 2, \dots, n.
\end{aligned}
$$

CFB: descripción gráfica del descifrado

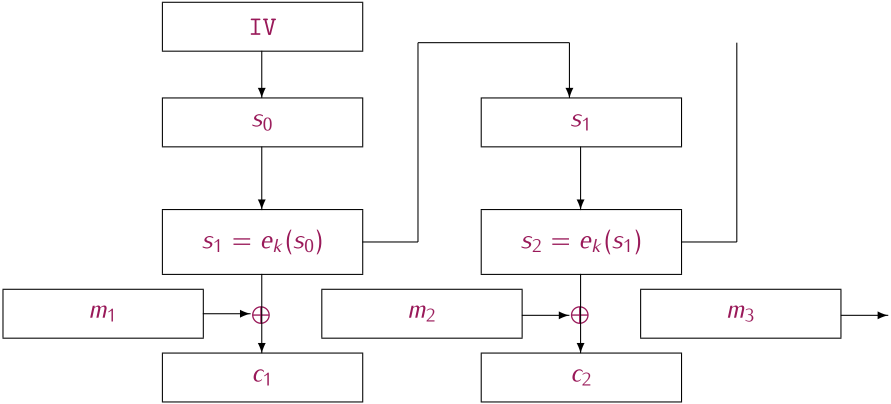


Output FeedBack (OFB)

Partimos de nuevo de un vector de inicialización $s_0 = \mathtt{IV}$. El cifrado por bloques actúa como la función generadora de un cifrado de flujo síncrono, sistema que estudiaremos más adelante. Formalmente:

$$
\begin{aligned}
& s_0 \\
& c_i := m_i \oplus e_k(s_{i-1}) \quad \text{for } i = 1, \dots, n \\
& s_i := e_k(s_{i-1}) \qquad \text{if } i = 1, \dots, n.
\end{aligned}
$$

OFB: descripción gráfica del cifrado

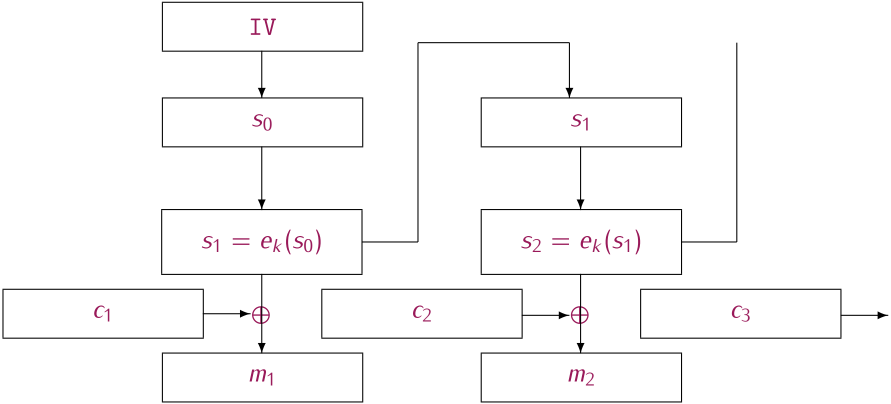


OFB: descifrado

El proceso para descifrar es exactamente el mismo, empleando el mismo vector de inicialización:

$$
\begin{aligned}
&s_0 \\
&m_i := c_i \oplus e_k(s_{i-1}) \quad \text{for } i = 1, \ldots, n \\
&s_i := e_k(s_{i-1}) \qquad \text{if } i = 1, \ldots, n.
\end{aligned}
$$

OFB: descripción gráfica del descifrado



Counter Mode (CTR)

Este modo parte de un vector inicial $\mathtt{IV}$, llamado aquí valor de un solo uso (*nonce* en inglés), y una función contador que genera una sucesión

$$h_0 = \mathtt{IV}, \dots, h_{i+1} = \mathrm{CTR}(h_i), \dots, h_t$$

de manera que si el mensaje se descompone en bloques como $m = m_0 || m_1 || \cdots || m_t$ el cifrado se produce de la forma

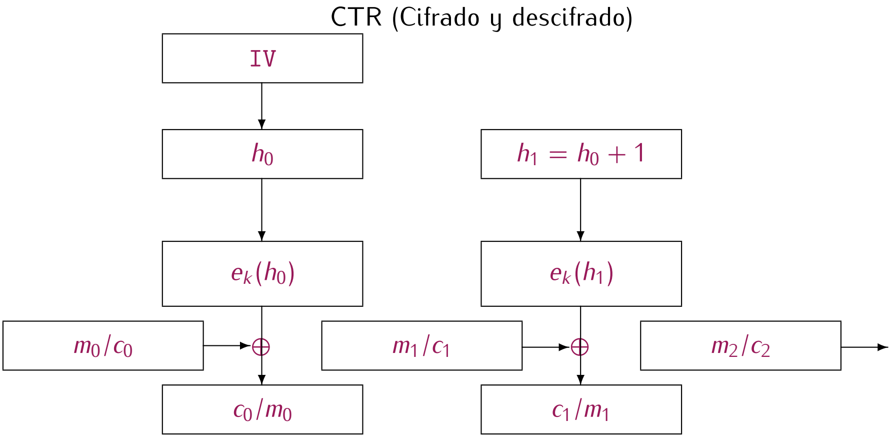
$$c_i = m_i \oplus e_k(h_i) \quad \text{para } i = 0, \dots, t.$$

El descifrado actúa de la misma manera.

$$m_i = c_i \oplus e_k(h_i) \quad \text{para } i = 1, \dots, t.$$

Es otra forma de utilizar el criptosistema por bloques como un cifrado de flujo.
La función contador debe garantizar que no se emplea el mismo valor para cifrar dos bloques diferentes. Suele emplearse el incremento en una unidad, es decir, $\mathrm{CTR}(h_{i+1}) = h_i + 1$.

CTR: descripción gráfica



Índice

Red de Feistel: Cifrado

El proceso de cifrado funciona de la siguiente forma:

1. La entrada es un bloque de longitud N.
2. Este bloque se divide en dos partes de longitud $N/2$, llamadas L_0 y R_0.
3. Durante n rondas, es decir, moviendo i de 1 a n,

   $$L_i = R_{i-1} \qquad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

   donde K_i es la subclave usada en cada ronda.
4. La salida es el bloque $R_n || L_n$.

Red de Feistel: Descifrado

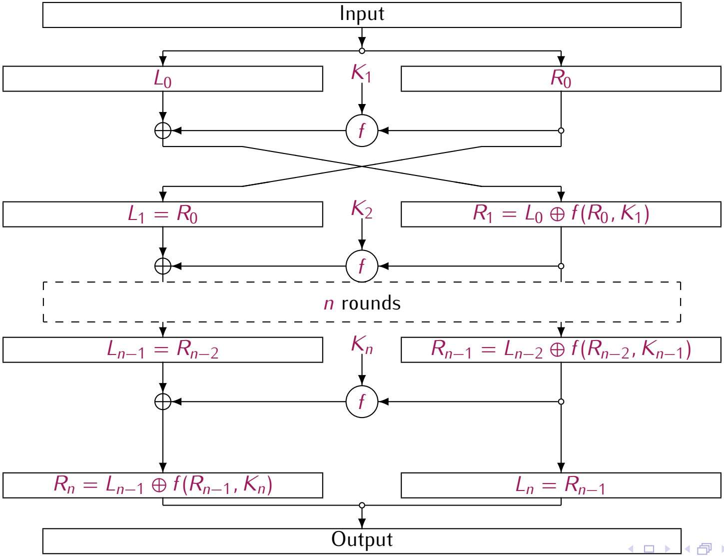
El proceso de descifrado es simétrico

1. La entrada es un bloque de longitud N.
2. Este bloque se divide en dos de longitud $N/2$, llamados R_n y L_n
3. Para cada i entre $n-1$ y 0,

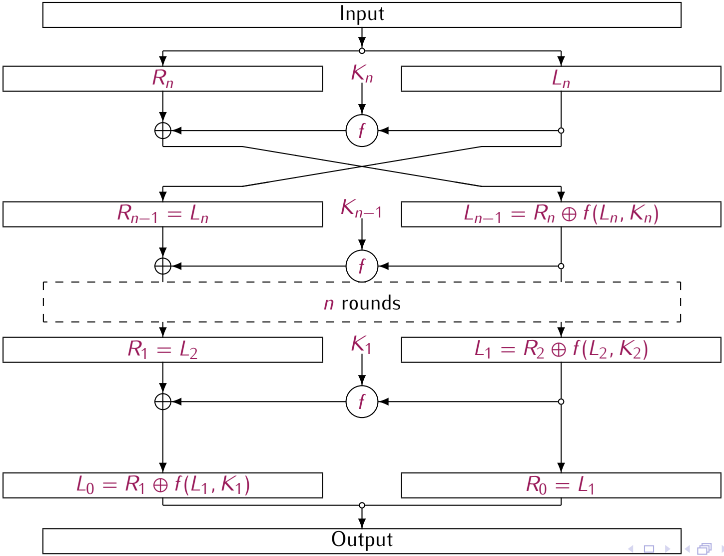
$$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus f(L_{i+1}, K_{i+1})$$

   donde K_i es la subclave usada en cada ronda.
4. La salida es el bloque $L_0 || R_0$.

Red de Feistel: Descripción gráfica del cifrado



Red de Feistel: Descripción gráfica del descifrado



DES: Punto de partida

DES fue un desarrollo realizado a partir de una solicitud del organismo americano entonces llamado NBS (National Boureau of Standards) —hoy rebautizado como NIST (National Institute of Standards and Technology)— por IBM, basado en un criptosistema previo llamado LUCIFER. Debía satisfacer los siguientes requisitos:

- El algoritmo debe proporcionar un alto nivel de seguridad.
- Debe ser fácil de describir y completamente especificado.
- La seguridad debe recaer en la clave, no en el secreto del algoritmo.
- Totalmente accesible a todos los usuarios.
- Fácil de adaptar a diferentes aplicaciones.
- Fácil de implementar en hardware.
- Debe ser eficiente..

DES: Observaciones sobre su seguridad

1. La longitud de la clave, 56 bits, fue considerada pequeña demasiado pronto.
2. Los criterios de diseño de las S-cajas fue mantenido en secreto hasta 1994, lo que hizo creer a mucha gente que tenían una puerta secreta.

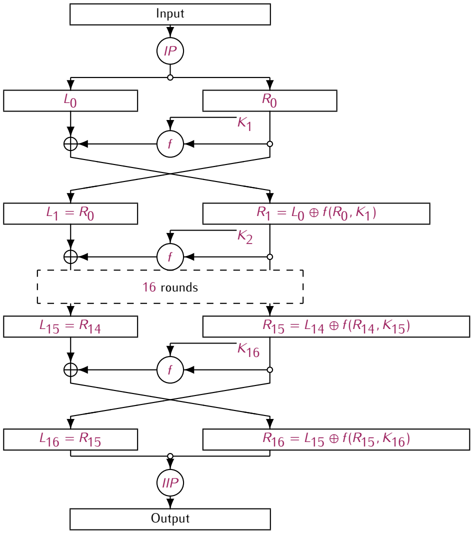
Hoy en día una clave puede obtenerse en 24 horas.

DES: Descripción

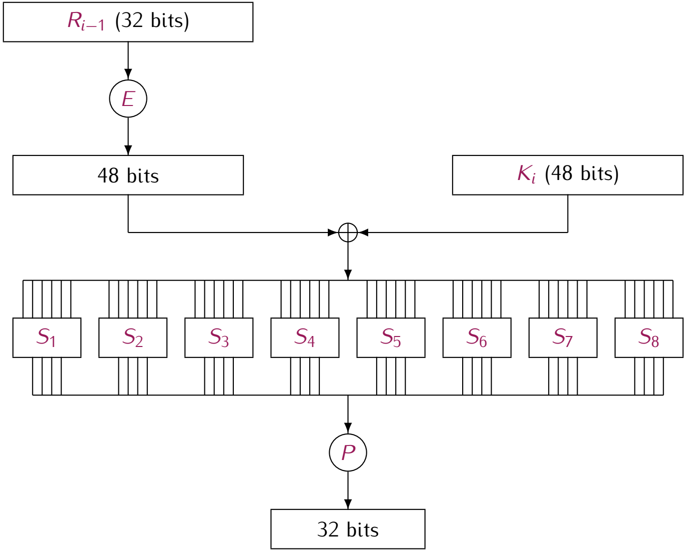
DES es un cifrado por bloques.

1. La entrada es un bloque de 64 bits, y la salida tiene la misma longitud.
2. DES es una red de Feistel de 16 rondas.
3. Cifrado y descifrado son idénticos excepto en el orden de las subclaves.
4. La clave tiene una longitud de 56 bits, aunque se presenta como un bloque de 64 donde los bits 8, 16, 24, 32, 40, 48, 56 y 64 son bits de paridad.
5. Hay claves débiles y semidébiles, pero son conocidas y fácilmente eliminadas.
6. Las primeras implementaciones fueron en hardware. No es muy rápido en sus versiones software.

DES: Descripción gráfica del cifrado



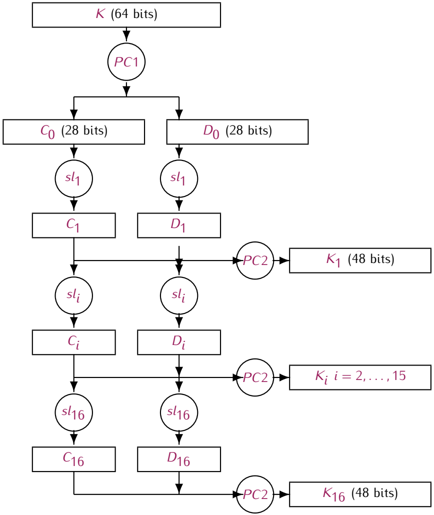
DES: Función f



DES: Expansión de la clave

1. Los bits de paridad se desechan y se permutan los demás bits mediante una función llamada *Permutation Choice* 1 $PC1$.
2. Este bloque de 56 bits se divide en dos bloque de 28 bits. Si $K = \langle k_1, \ldots, k_{28}, k_{29}, \ldots, k_{56} \rangle$, llamamos $C = \langle k_1, \ldots, k_{28} \rangle$ and $D = \langle k_{29}, \ldots, k_{56} \rangle$.
3. Realizamos un desplazamiento cíclico a la izquierda de una o dos posiciones, dependiendo de la ronda, en los bloques C y D.
4. En cada ronda seleccionamos 48 bits usando la *Permutation Choice 2* $PC2$.

DES: Descripción gráfica de la expansión de la clave



DES: Expansión y permutación E

Esta función transforma un bloque de longitud 32 en otro de 48 bits duplicando la mitad de ellos. Todos son reordenados.

- Los 48 bits son sumados a la clave de ronda.
- Los bits duplicados serán comprimidos más adelante.
- Esta técnica aumenta la aleatoriedad entre la entrada y la salida.

S-cajas

- Las funciones de sustitución empleadas son 8 llamadas S-cajas.
- Cada bloque de 48 bits se divide en 8 subbloques de 6 bits, siendo operado cada uno de ellos por una S-caja distinta.
- Las S- cajas se representan por una matriz 4×16, con filas numeradas de 0 a 3 y columnas de 0 a 15.
- De cada entrada $\langle b_0 b_1 b_2 b_3 b_4 b_5 \rangle$ ontenemos dos números que representan una fila y una columna. La fila es $r = b_0 2 + b_5$ y la columna $c = b_1 2^3 + b_2 2^2 + b_3 2 + b_4$.
- El valor en binario correspondiente a la posición $\langle r, c \rangle$ nos da una salida de 4 bits.
- La concatenación de las 8 salidas nos da un nuevo bloque de longitud 32.

Claves débiles

En 1992 Campbell y Wiener demostraron que DES está muy lejos de ser un grupo, es decir, existen claves K^1 y K^2 tales que para cualquier otra clave K^3

$$\mathtt{DES}_{K^2} \circ \mathtt{DES}_{K^1} \neq \mathtt{DES}_{K^3}.$$

Esta falta de estructura hace que el criptoanálisis sea más difícil.

Una clave es *débil* si todas las subclaves de ronda son iguales. Es *semidébil* si solo genera dos o cuatro claves de ronda diferentes y alternadas. Hay 4 claves débiles y 60 (12 con dos claves de ronda y 48 cuatro) claves semidébiles. Todas están tabuladas y no representan un problema de seguridad por su pequeña cantidad. La débiles son en hexadecimal

```
0x0101010101010101
0x1F1F1F1F0E0E0E0E
0xE0E0E0E0F1F1F1F1
0xFEFEFEFEFEFEFEFE
```

Índice

AES: Introducción

Es el nuevo estándar diseñado para reemplazar a DES. Se empieza a gestar el 12 de septiembre de 1997, cuando el departamento de comercio del *National Institute of Standards and Technology (NIST)* hace un llamamiento público para la presentación de algoritmos.

Los requisitos mínimos son:

1. El algoritmo debe ser simétrico de clave secreta.
2. El algoritmo debe ser un algoritmo de bloque.
3. El algoritmo debe ser capaz de soportar las combinaciones clave-bloque de tamaños 128-128, 192-128 y 256-128.

AES: Evaluación

Los criterios de evaluación de los proyectos presentados fueron los siguientes:

Seguridad. El factor más importante en la evaluación de los candidatos.

Coste.
1. El algoritmo debe ser accesible a todo el mundo y de libre distribución.
2. El algoritmo debe ser computacionalmente eficiente tanto en hardware como en software.
3. El algoritmo debe utilizar la menor memoria posible tanto en hardware como en software.

AES: Características de implementación del algoritmo.

1. El algoritmo debe ser fácilmente implementable en distintas plataformas tanto en hardware como en software.
2. El algoritmo debe acomodarse a diferentes combinaciones clave-bloque además de las mínimas requeridas.
3. El algoritmo debe ser de diseño simple.

AES: candidatos

A modo de curiosidad veamos algunos de los candidatos que se presentaron y sus promotores.

CAST-256	Entust Technologies, Inc. (C. Adams).
CRYPTON	Future Systems, Inc. (Chae Hoon Lim).
DEAL	L. Knudsen, R. Outerbridge.
DFC	CNRS-Ecole Normale Superiere (S. Vaudenay).
E2	NTT Nippon Telegraph and Telephone Corporation (M. Kanda).
FROG	TecApro International S.A. (D. Georgoudis, D. Leroux, B. S. Chaves).
HPC	R. Schoeppel.
LOKI97	L. Brown, J. Pieprzyk, J.Seberry.
MAGENTA	Deutshe Telekom AG (K. Huber).
MARS	IBM (N. Zunic).
RC6	RSA Laboratories (Rivest, M. Robshaw, Sidney, Yin).
RIJNDAEL	J. Daemen, V. Rijmen.
SAFER+	Cylink Corporation (L. Chen).
SERPENT	R. Anderson, E. Biham, L. Knudsen.
TWOFISH	B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson.

AES: RIJNDAEL

- Es un algoritmo simétrico de bloque de 128 bits y clave de 128, 192 o 256 bits.
- Para representar los bloques usamos polinomios de grado tres con coeficientes en $\mathbb{F}_{2^8} = \mathbb{F}_{256}$, esto es el cuerpo finito de 256 elementos.
- La representación del cuerpo $\mathbb{F}_{256}$ se realiza mediante el polinomio $x^8 + x^4 + x^3 + x + 1$, que es un polinomio irreducible sobre el cuerpo $\mathbb{F}_2$, el cuerpo finito de dos elementos.
- El grupo multiplicativo de $\mathbb{F}_{256}$ es un grupo cíclico, generado por la clase de $x + 1$. Esto es, los elementos de $\mathbb{F}_{256}$ se pueden escribir como potencias de $x + 1$.

AES: elementos de $\mathbb{F}_{256}$

Cada elemento de $\mathbb{F}_{256}$ es la clase de un polinomio de grado menor que ocho, por tanto, escribiéndolo como una lista de sus coeficientes, sería

$$a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \leftrightarrow a_7a_6a_5a_4a_3a_2a_1a_0.$$

Esto es, una 8-upla de elementos de $\mathbb{F}_2 = \{0, 1\}$.

Se puede identificar pues con un *byte*. Los bytes, se pueden escribir bien en modo hexadecimal o binario. De esta forma tenemos la correspondencia

byte	$\rightarrow$	hexadecimal	$\rightarrow$	polinomio
10100101	$\rightarrow$	`0xA5`	$\rightarrow$	$x^7 + x^5 + x^2 + 1.$

AES: elementos de $\mathbb{F}_{256}$ no nulos

El grupo multiplicativo de $\mathbb{F}_{256}$ es un grupo cíclico, de orden 255. De entre sus generadores el, posiblemente, más sencillo es el `0x03`, que corresponde al polinomio $x+1$. Todas las potencias de `0x03`, escritas en hexadecimal, son:

03	05	0F	11	33	55	FF	1A	2E	72	96	A1	F8	13	35	5F	16
E1	38	48	D8	73	95	A4	F7	02	06	0A	1E	22	66	AA	E5	32
34	5C	E4	37	59	EB	26	6A	BE	D9	70	90	AB	E6	31	53	48
F5	04	0C	14	3C	44	CC	4F	D1	68	B8	D3	6E	B2	CD	4C	64
D4	67	A9	E0	3B	4D	D7	62	A6	F1	08	18	28	78	88	83	80
9E	B9	D0	6B	BD	DC	7F	81	98	B3	CE	49	DB	76	9A	B5	96
C4	57	F9	10	30	50	F0	0B	1D	27	69	BB	D6	61	A3	FE	112
19	2B	7D	87	92	AD	EC	2F	71	93	AE	E9	20	60	A0	FB	128
16	3A	4E	D2	6D	B7	C2	5D	E7	32	56	FA	15	3F	41	C3	144
5E	E2	3D	47	C9	40	C0	5B	ED	2C	74	9C	BF	DA	75	9F	160
BA	D5	64	AC	EF	2A	7E	82	9D	BC	DF	7A	8E	89	80	9B	176
B6	C1	58	E8	23	65	AF	EA	25	6F	B1	C8	43	C5	54	FC	192
1F	21	63	A5	F4	07	09	1B	2D	77	99	B0	CB	46	CA	45	208
CF	4A	DE	79	8B	86	91	A8	E3	3E	42	C6	51	F3	0E	12	224
36	5A	EE	29	7B	8D	8C	8F	8A	85	94	A7	F2	0D	17	39	240
4B	DD	7C	84	97	A2	FD	1C	24	6C	B4	C7	52	F6	01		255

AES: suma

Es un XOR a nivel de bits. Por ejemplo

$$(x^4 + x^3 + 1) + (x^7 + x^6 + x^4 + x^2 + 1) = x^7 + x^6 + x^3 + x^2$$
$$00011001 \oplus 11010101 = 11001100$$
$$\mathtt{0x19} \oplus \mathtt{0xD5} = \mathtt{0xCC}$$

AES: multiplicación

Producto de polinomios y reducción módulo $x^8+x^4+x^3+x+1$. Por ejemplo

$$
\begin{aligned}
&(x^6+x^4+x^2+x+1)(x^7+x+1)\\
&\quad= (x^{13}+x^{11}+x^9+x^8+x^7)+(x^7+x^5+x^3+x^2+x)+(x^6+x^4+x^2+x+1)\\
&\quad= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1\\
&\quad\equiv x^7+x^6+1 \pmod{x^8+x^4+x^3+x+1}.
\end{aligned}
$$

En hexadecimal y a nivel de bits

$$\texttt{0x57}\bullet\texttt{0x83}=\texttt{0xC1} \qquad 01010111 \bullet 10000011 = 11000001$$

¿Cómo hacer esto de forma eficiente?

AES: multiplicación por recurrencia

Para hacer la multiplicación basta multiplicar por x, y después hacer la suma, XOR. El producto $(x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1)$ es la suma

$$(x^6 + x^4 + x^2 + x + 1) \cdot x^7 + (x^6 + x^4 + x^2 + x + 1) \cdot x + (x^6 + x^4 + x^2 + x + 1)$$

También se puede realizar como:

$$(x^6 + x^4 + x^2 + x + 1) \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x + (x^6 + x^4 + x^2 + x + 1) \cdot x + (x^6 + x^4 + x^2 + x + 1)$$

y vamos haciendo por recurrencia las multiplicaciones.

AES: multiplicación con logaritmos

Recordemos que los elementos no nulos de $\mathbb{F}_{256}$ forman un grupo cíclico de orden 255 generado por $x+1 = \texttt{0x03} = 00000011$. Esto significa que todo elemento distinto de cero es una potencia de $x+1$, por ejemplo

$$11001100 = \texttt{0xAA} = x^7 + x^6 + x^3 + x^2 = (x+1)^{31},$$

por lo que

$$\log_{\texttt{0x03}} \texttt{0xAA} = 31.$$

La multiplicación se realiza empleando tablas de logaritmos. Así, si $a, b \in \mathbb{F}_{256}^*$

$$ab = \texttt{0x03}^{\log_{\texttt{0x03}} a + \log_{\texttt{0x03}} b \pmod{255}}.$$

Por ejemplo

$$\texttt{0x57} \bullet \texttt{0x83} = \texttt{0x03}^{\log_{\texttt{0x03}} \texttt{0x57} + \log_{\texttt{0x03}} \texttt{0x83} \pmod{255}} = \texttt{0x03}^{98+80 \pmod{255}} = \texttt{0x03}^{178} = \texttt{0xC1}.$$

Las tablas de logaritmos se tabulan.

AES: multiplicación extendida I

Para algunos procesos AES emplea subbloques de 32 bits, es decir, necesitamos extender la aritmética a bloques de 4 bytes. Podemos representar estos bloque como polinomios de grado menor o igual que 3 con coeficientes en $\mathbb{F}_{256}$.
La suma sigue realizándose a nivel de bits mediante XOR. Vamos a multiplicar dos polinomios de grado menor que 4 con coeficientes en $\mathbb{F}_{256}$, si $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ y $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, entonces el producto es un polinomio de hasta grado 6. Sea $c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$, siendo

$$
\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3
\end{aligned}
$$

AES: multiplicación extendida II

para tener un polinomio de grado menor o igual que 3 basta reducir módulo un polinomio de grado 4. En este caso el polinomio $x^4 + 1$. Resulta entonces la relación:

$$x^h \equiv x^{h \pmod 4} \pmod{x^4 + 1}.$$

Al aplicar esto a la relación anterior resulta que tenemos que sumar c_0 y c_4, c_1 y c_5 y c_2 y c_6. Si llamamos

$$d(x) = a(x)b(x) \pmod{x^4 + 1},$$

y si $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$, tenemos:

$$\begin{aligned}
d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\
d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3,
\end{aligned}$$

AES: multiplicación extendida III

Esta multiplicación admite una representación matricial

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Esta transformación se puede invertir cuando $a(x)$ es un polinomio invertible módulo $x^4 + 1$.
(Observemos que $x^4 + 1$ no es irreducible; y que necesitamos que $a(x)$ sea primo relativo con $x^4 + 1$.)
En el caso particular en que $a(x) = x$, que es evidentemente primo relativo con $x^4 + 1$, la transformación es:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} b_3 \\ b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

Tenemos entonces una permutación que mueve los elementos una unidad a la derecha.

Parámetros

Tenemos tres números que determinan el funcionamiento del criptosistema, estos son:

- $\mathbf{N_b}$ es el número de bits del bloque dividido por 32. En Rijndael sus valores son: 4, 6 u 8 según que el bloque tenga longitud 128, 192 o 256. El estándar AES sólo admite bloques de tamaño 128, por lo que el valor es 4
- $\mathbf{N_k}$ es el número de bits de la clave dividido por 32. Sus valores son: 4, 6 u 8 según que la clave tenga longitud 128, 192 o 256.
- $\mathbf{N_r}$ el *número de rondas* determinado por los dos anteriores según el cuadro:

$\mathbf{N_r}$	$N_b = 4$	$N_b = 6$	$N_b = 8$
$N_k = 4$	**10**	12	14
$N_k = 6$	**12**	12	14
$N_k = 8$	**14**	14	14

Estado o *state*

Se actúa sobre un resultado intermedio, al que llamamos *state*. Éste es una estructura del siguiente tipo:

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$\cdots$	$\cdots$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\cdots$	$\cdots$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$\cdots$	$\cdots$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$\cdots$	$\cdots$

Donde cada $m_{i,j}$ son 8 bits, esto es, un byte, o si se quiere un elemento de $\mathbb{F}_{256}$.
El número de columnas es N_b, esto es, 4, 6 u 8 según el tamaño del bloque.

Primer estado

El primer *state* se forma, por columnas, dividiendo el bloque (por ejemplo de 128 bits, o equivalentemente 16 bytes)

$$m_{0,0}m_{1,0}m_{2,0}m_{3,0}m_{0,1}m_{1,1}m_{2,1}m_{3,1}m_{0,2}m_{1,2}m_{2,2}m_{3,2}m_{0,3}m_{1,3}m_{2,3}m_{3,3}$$

en los subbloques correspondientes según en siguiente esquema:

$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$\cdots$	$\cdots$
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\cdots$	$\cdots$
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$\cdots$	$\cdots$
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$\cdots$	$\cdots$

Rondas

La primera ronda, ronda 0, consiste en

1. $\texttt{AddRoundKey}(state, RoundKey_0)$

A continuación se realizan $N_r - 1$ rondas con el siguiente esquema: $\texttt{Round}(state, RoundKey_i)$, $i = 1, \ldots, N_r - 1$

1. $\texttt{SubBytes}(state)$
2. $\texttt{ShiftRows}(state)$
3. $\texttt{MixColumns}(state)$
4. $\texttt{AddRoundKey}(state, RoundKey_i)$

Y finalmente la última ronda con el esquema: $\texttt{FinalRound}(state, RoundKey_{N_r})$

1. $\texttt{SubBytes}(state)$
2. $\texttt{ShiftRows}(state)$
3. $\texttt{AddRoundKey}(state, RoundKey_{N_r})$

AddRoundKey

La función

$$\mathtt{AddRoundKey}(state, RoundKey_i),$$

consiste en un XOR entre *state* y $RounKey_i$ bit a bit; ($\oplus$ en la notación anterior).

$$state \oplus RounKey_0$$

SubBytes

Cada uno de los $m_{i,j}$ es un byte (8 bits), entonces:

1. Se calcula el inverso de $m_{i,j}$ en $\mathbb{F}_{256}$; la imagen de 00 es él mismo, obteniendo otro byte (8 bits) $(x_0 \cdots x_7)$,
2. A este byte se le aplica la transformación afín (invertible) definida por:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

en donde la columna de la derecha es el número `0x63`, (hexadecimal).

ShiftRows

Las filas de *state* se desplazan a la izquierda cíclicamente.
La primera fila permanece en su posición, la segunda se desplaza C1 posiciones a la izquierda, la tercera C2 y la cuarta C3, según la siguiente tabla que depende de N_b.

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Por ejemplo en el caso de $N_b = 4$ tenemos:

$$\begin{array}{|c|c|c|c|}\hline m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ \hline m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} \\ \hline m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} \\ \hline m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} \\ \hline \end{array} \quad \rightsquigarrow \quad \begin{array}{|c|c|c|c|}\hline m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} \\ \hline m_{1,1} & m_{1,2} & m_{1,3} & m_{1,0} \\ \hline m_{2,2} & m_{2,3} & m_{2,0} & m_{2,1} \\ \hline m_{3,3} & m_{3,0} & m_{3,1} & m_{3,2} \\ \hline \end{array}$$

MixColumns

Las columnas de *state* se consideran polinomios, hasta grado tres, sobre $\mathbb{F}_{256}$ y se multiplican, módulo el polinomio $x^4 + 1$, por el polinomio con coeficientes en $\mathbb{F}_{256}$:
$c(x) = \mathtt{0x03}x^3 + \mathtt{0x01}x^2 + \mathtt{0x01}x + \mathtt{0x02}$, (los coeficientes están escritos en hexadecimal).
Como esta operación es siempre la misma, podemos dar una fórmula genérica. Se tiene que el producto $b(x) = c(x) \otimes a(x)$ está dado por la multiplicación matricial:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Formación de las claves de ronda

- La clave *Key* se extiende a una lista de palabras de 4 bytes que llamaremos **W** y que contiene exactamente $N_b(N_r + 1)$ palabras.
- Los primeros N_k elementos de **W**, esto es, $W[0], \ldots, W[N_k - 1]$, corresponden a partes de la clave rellenas por columnas y de arriba a abajo
- El resto se definen de forma recursiva utilizando:
  1. la función `SubBytes` actuando sobre palabras, a la que llamaremos `SubWord`
  2. desplazamientos cíclicos y
  3. la operación $\oplus$.
- La recurrencia depende de la longitud de la clave.

Formación de las claves de ronda: Funciones

`SubWord` consiste en aplicar la S-box `SubBytes` a cada uno de los cuatro bytes de una palabra.

`RotWord` aplicado a una palabra de 4 bytes, devuelve una palabra cuyos bytes se han desplazado cíclicamente una posición a la izquierda, por ejemplo `RotWord`$(a, b, c, d) = (b, c, d, a)$. A veces también se llama `RotByte`.

`Rcon` `Rcon`$[i] = (RC[i], \texttt{0x00}, \texttt{0x00}, \texttt{0x00})$ siendo $RC[i]$ un elemento de $\mathbb{F}_{256}$ definido recursivamente mediante:

$$
\begin{aligned}
RC[1] &= \texttt{0x01},\\
RC[i] &= \texttt{0x02} \bullet RC[i-1],
\end{aligned}
$$

que escrito como clases de polinomios es:

$$
\begin{aligned}
RC[1] &= 1,\\
RC[i] &= x \bullet RC[i-1]
\end{aligned}
$$

Formación de las claves de ronda: $N_k = 4$ o 6

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)])

   for(i=0; i<Nk; i++)
      W[i]=(Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
   end for
   for(i=Nk; i<Nb*(Nr + 1); i++)
      temp=W[i-1];
      if (i mod Nk == 0)
         temp=SubWord(RotWord(temp)) XOR Rcon[i/Nk];
      W[i]=W[i-Nk] XOR temp;
   endfor
```

Formación de las claves de ronda: $N_k = 8$

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)])

   for(i=0; i<Nk; i++)
      W[i]=(key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
   end for
   for(i=Nk; i<Nb*(Nr+1); i++)
      temp=W[i - 1];
      if (i mod Nk == 0)
         temp=SubWord(RotWord(temp)) XOR  Rcon[i/Nk];
      else if (i mod Nk == 4)
         temp=SubWord(temp); W[i]=W[i-Nk] XOR temp;
      end if
   end for
```

La diferencia con la situación anterior es que cuando $i - 4$ es un múltiplo de N_k, entonces aplicamos `SubWord` a $W[i - 1]$ antes de hacer XOR.

Descifrado

Falta ver como se realiza el descifrado. Para esto se realizan las operaciones inversas de las realizadas y en orden inverso al realizado, es decir:
La primera ronda, ronda N_r, consiste en $\texttt{Round}(state, RoundKey_{N_r})$

1. $\texttt{InvAddRoundKey}(state, RoundKey_{N_r})$
2. $\texttt{InvShiftRows}(state)$
3. $\texttt{InvSubBytes}(state)$

A continuación se realizan $N_r - 1$ rondas con el siguiente esquema: $\texttt{Round}(state, RoundKey_i)$, $i = N_r - 1, \dots, 1$

1. $\texttt{InvAddRoundKey}(state, RoundKey_i)$
2. $\texttt{InvMixColumns}(state)$
3. $\texttt{InvShiftRows}(state)$
4. $\texttt{InvSubBytes}(state)$

Y finalmente la última ronda con el esquema:

1. $\texttt{InvAddRoundKey}(state, RoundKey_0)$

Funciones inversas

`InvAddRoundKey` esta función es la misma que `AddRoundKey`, ya que el XOR es su propio inverso

`InvShiftRows` El mismo desplazamiento que `ShiftRows` pero a la derecha.

`InvSubBytes` primero se realiza la inversa de la transformación afín y después se calculan los inversos en $\mathbb{F}_{256}$.

`InvMixColumns` multiplicar por la matriz inversa de la usada en `MixColumn`:

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$$

Optimizaciones

Dos aspectos a considerar:

1. Las transformaciones SubBytes y ShiftRows (y en consecuencia sus inversas) conmutan, es decir, podemos cambiarlas de orden sin que afecte al resultado final
2. La multiplicación matricial es lineal, es decir,

$$\mathtt{InvMixColumns}(state \oplus RoundKey_i) = \mathtt{InvMixColumns}(state) \oplus \mathtt{InvMixColumns}(RoundKey_i)$$

Definimos por este motivo las siguientes nuevas claves de ronda:

$$\begin{aligned} InvRoundKey_0 &= RoundKey_{N_r}, \\ InvRoundKey_i &= \mathtt{InvMixColumn}(RoundKey_{N_r - i}) \text{ cuando } i = 1, \ldots, N_r - 1, \\ InvRoundkey_{N_r} &= RoundKey_0. \end{aligned}$$

Descifrado optimizado

El proceso de descifrado queda por tanto con la siguiente estructura

La ronda 0 consiste en

1. $\texttt{AddRoundKey}(state, InvRoundKey_0)$

A continuación se realizan $N_r - 1$ rondas con el siguiente esquema:

$\texttt{Round}(state, InvRoundKey_i)$, $i = 1, \ldots, N_r - 1$

1. $\texttt{InvSubBytes}(state)$
2. $\texttt{InvShiftRows}(state)$
3. $\texttt{InvMixColumns}(state)$
4. $\texttt{AddRoundKey}(state, InvRoundKey_i)$

Y finalmente la última ronda con el esquema:

$\texttt{FinalRound}(state, InvRoundKey_{N_r})$

1. $\texttt{InvSubBytes}(state)$
2. $\texttt{InvShiftRows}(state)$
3. $\texttt{AddRoundKey}(state, InvRoundKey_{N_r})$

Índice

One-time pad

Ya hemos hablado del *one-time pad*. Para mensajes digitales, es decir cadenas en el alfabeto $\mathbb{B} = \mathbb{Z}_2 = \mathbb{F}_2 = \{0, 1\}$, el sistema es conocido como cifrado de Vernam, ya que fue patentado por G. Vernam en 1919. Es el único criptosistema seguro de acuerdo con los criterios de Shannon. La clave es una secuencia aleatoria en $\mathbb{B}$ de longitud mayor que el mensaje, y la función de cifrado es la suma (XOR) del mensaje y la clave bit a bit.

Los cifrados de flujo emplean sucesiones pseudoaleatorias, conocidas aquí por el nombre de *sucesiones criptográficas*, en lugar de aleatorias. Estas sucesiones son generadas a partir de una *semilla*, tienen periodos muy grandes y es computacionalmente imposible averiguar un bit a partir de los anteriores, siempre que nos mantengamos por debajo de periodo. Las sucesiones pseudoaleatorias deben pasar algunos test estadísticos de aleatoriedad.

Postulados de Golomb I

Definición

- Sea $s = s_0 s_1 s_2 \ldots$ una sucesión infinita en $\mathbb{F}_2$. La cadena consistente en los primeros n términos de s se denota por $s^n = s_0 s_1 \ldots s_{n-1}$.
- Una sucesión s se dice N-periódica si $s_i = s_{i+N}$ para cualquier $i \geq 0$. Si s es N-periódica, el *ciclo* de s es s^N.
- Una racha de s es una subcadena de s consistente en 0s o 1s consecutivos delimitada por el otro símbolo. Una racha de 0s se llama hueco, y una racha de 1s bloque.
- Sea s una sucesión N-periódica. La función de autocorrelación de s es

$$C(t) = \sum_{i=0}^{N-1} (-1)^{s_{i+t}+s_i}, \text{ for } 0 \leq t \leq N-1.$$

Postulados de Golomb II

Definición

Sea s una sucesión N-periódica. Los postulados de Golomb son:

1. En el ciclo s^N, el número de 0s difiere del número de 1s en una unidad como máximo.
2. En el ciclo s^N, al menos la mitad de las rachas tienen longitud 1, al menos un cuarto tienen longitud 2, un octavo longitud 3, etc. Además, para cada longitud el número de huecos es (casi) igual al número de bloques.
3. La función de autocorrelación $C(t)$ fuera de fase es constante, i.e. existe $K \in \mathbb{Z}$ tal que para cualquier $0 < t \leq N - 1$
$$C(t) = K.$$

Cifrados de flujo síncronos

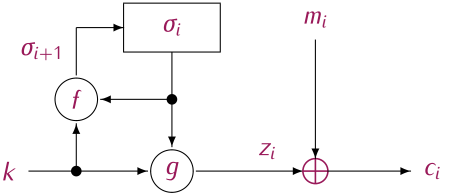
La sucesión criptográfica se genera independientemente del mensaje y del criptograma. Satisface las ecuaciones:

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i, k), \\ z_i &= g(\sigma_i, k), \\ c_i &= z_i \oplus m_i, \end{aligned}$$

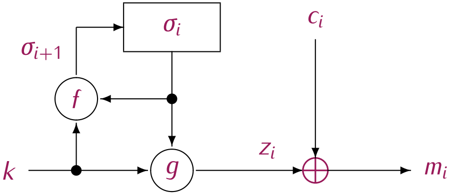
donde σ_0 es el estado inicial, k es la clave, f es la función *siguiente estado*, g produce la sucesión criptográfica z_i que es sumada (XOR) con mensaje m_i para generar el criptograma c_i.

Cifrados de flujo síncronos: representación gráfica

Encryption



Decryption



Cifrados de flujo síncronos: propiedades

- Sincronización obligatoria.
- La inserción o borrado de bits se detecta fácilmente.
- La alteración de bits es difícil de detectar.

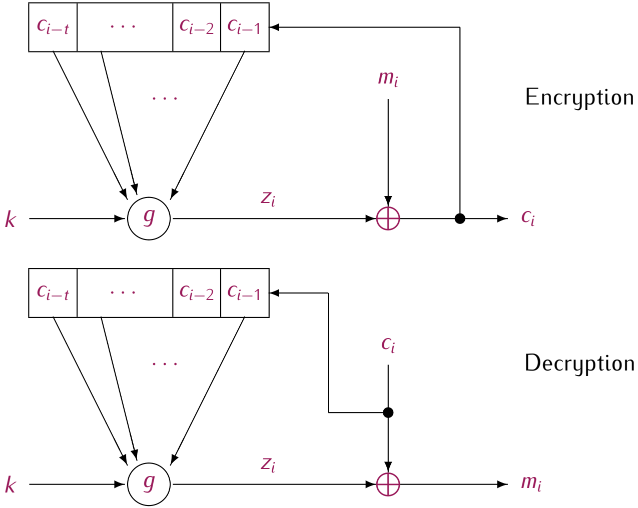
Cifrados de flujo autosincronizables

La sucesión criptográfica se genera a partir de la clave y de una cantidad fija de bits en el criptograma, matemáticamente

$$\begin{aligned} \sigma_i &= (c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}), \\ z_i &= g(\sigma_i, k) \\ c_i &= z_i \oplus m_i, \end{aligned}$$

donde $\sigma_0 = (c_{-t}, c_{-t+1}, \ldots, c_{-1})$ es el estado inicial, k es la clave, g es la función que genera la sucesión criptográfica z_i que es sumada (XOR) con mensaje m_i para generar el criptograma c_i.

Cifrados de flujo autosincronizables: representación gráfica



Cifrados de flujo autosincronizables: características

- Sincronización automática.
- Inserción o borrado de bits más difícil de detectar.
- Alteración de bits más fácil de detectar.

Índice

Feedback shift registers

Registros lineales de desplazamiento retroalimentados (Linear feedback shift registers, LFSR)) son empleados usualmente para definir las funciones f y g. Algunas razones son las siguientes:

1. se implementan muy fácilmente en hardware,
2. producen sucesiones de periodo largo,
3. producen sucesiones con buenas propiedades estadísticas,
4. pueden ser analizados mediante técnicas algebraicas.

LFSR: estructura

Un LFSR se modela a partir de un polinomio $C(D) = 1 + c_1 D + c_2 D^2 + \cdots + c_l D^l \in \mathbb{F}_2[D]$. La sucesión se genera recursivamente

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_l s_{j-l}) \pmod{2} \quad \text{for } j \geq l$$

donde el estado inicial es $[s_0, s_1, \ldots, s_{l-1}]$.
Si $C(D)$ es un polinomio primitivo[1], cada uno de los $2^l - 1$ estados iniciales no nulos genera una sucesión del máximo periodo $2^l - 1$. Estas sucesiones satisfacen los postulados de Golomb y su complejidad lineal[2] es también alta.

[1] el polinomio mínimo de un elemento primitivo en la extensión de Galois $\mathbb{F}_2 \subseteq \mathbb{F}_{2^l}$.
[2] Un índice asociado a una sucesión y calculado por el algoritmo de Berlekamp-Masey.

LFSR: Problemas y soluciones

Como las funciones generadas por un LFSR son lineales, son vulnerables a ataques de texto en claro conocido. Hay varias soluciones:

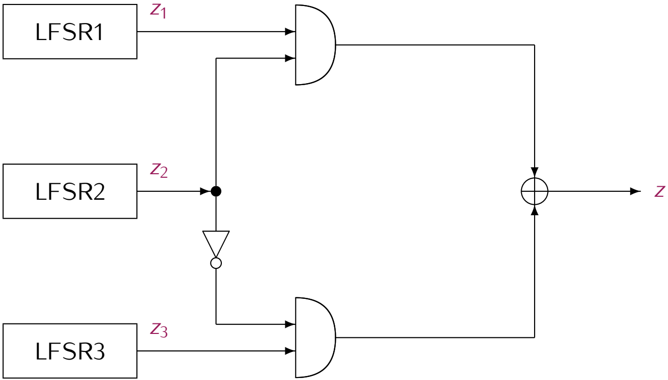
1. Podemos reemplazarlos por registros de desplazamiento retroalimentados no lineales (Non-linear feedback shift registers, NFSR),
2. podemos hacer una combinación no lineal de varios LFSRs,
3. podemos usar un filtro no lineal dentro de un LFSR,
4. un LFSR puede controlar el reloj que alimenta a varios LFSRs.

Generador de Geffe

Se utilizan tres LFSRs de longitudes l_1, l_2 y l_3 primos relativos. Las salidas z_1, z_2 y z_3 se combinan mediante la función no lineal

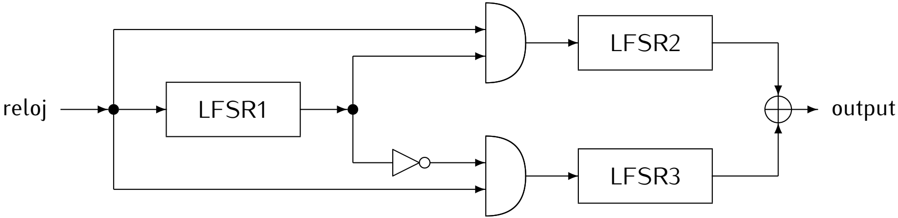
$$f(z_1, z_2, z_3) = z_1 z_2 \oplus \overline{z_2} z_3$$

El periodo global es $(2^{l_1} - 1)(2^{l_2} - 1)(2^{l_3} - 1)$, y la complejidad lineal $l_1 l_2 + l_2 l_3 + l_3$. Gráficamente



Generadores de paso alternado

Gráficamente:



Índice

eSTREAM

eSTREAM fue un proyecto, auspiciado por la red ECRYPT de la Unión Europea, desarrollado entre los años 2004 y 2008, y cuyo objetivo fue promover el diseño de cifrados de flujo compactos y eficientes adecuados para un uso generalizado. Como resultado se publicó en Abril de 2008, con una revisión en Septiembre, un portafolio que contiene siete cifrados de flujo adecuados para su uso en software o hardware.

Perfil 1 (SW)

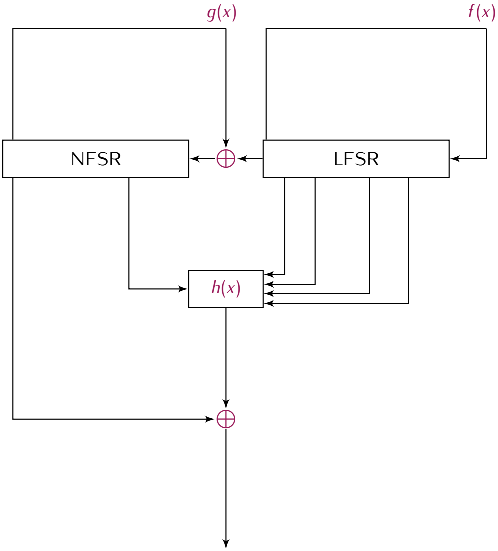
- HC-128
- Rabbit
- Salsa 20/12
- SOSEMANUK

Perfil 2 (HW)

- Grain v1
- MICKEY 2.0
- Trivium

`http://www.ecrypt.eu.org/stream/`

Grain v1: esquema gráfico



Grain v1: detalles

- $f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$
- $g(x) = 1 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}$
- $h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4$
- $x_0, x_1, x_2, x_3, x_4 = s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}$

Resumen de características

Confidencialidad Es la principal característica, sólo la entidad autorizada tiene acceso a la información.

Autenticidad Sólo el poseedor de la clave puede haber cifrado la información.

Integridad La alteración de la información se descubre al descifrar el criptograma, aunque algunos criptosistemas permiten reemplazar bloques completos de información.

No repudio No se puede garantizar, dos entidades comparten la clave.

Bibliografía I

- Hans Delfs and Helmut Knebl.
  Introduction to Cryptography. Principles and Applications.
  Information Security and Cryptography. Springer, third edition, 2015.

- National Institute of Standards and Technology (NIST).
  DATA ENCRYPTION STANDARD (DES), October 1999.

- National Institute of Standards and Technology (NIST).
  ADVANCED ENCRYPTION STANDARD (AES), November 2001.