CPSC 467: Cryptography and Computer Security

Michael Fischer

Lecture 7
September 20, 2017

Thanks to Ewa Syta for the slides on AES

Multiple Encryption
    Composition
    Group property

Birthday Attack

Advanced Encryption Standard

Multiple Encryption

Composition of cryptosystems

Encrypting a message multiple times with the same or different ciphers and keys seems to make the cipher stronger, but that's not always the case. The security of the composition can be difficult to analyze.

For example, with the one-time pad, the encryption and decryption functions E_k and D_k are the same. The composition $E_k \circ E_k$ is the identity function!

Composition within practical cryptosystems

Practical symmetric cryptosystems such as DES and AES are built as a composition of simpler systems.

Each component offers little security by itself, but when composed, the layers obscure the message to the point that it is difficult for an adversary to recover.

The trick is to find ciphers that successfully hide useful information from a would-be attacker when used in concert.

Double Encryption

Double encryption is when a cryptosystem is composed with itself. Each message is encrypted twice using two different keys k' and k'', so $E^2_{(k'',k')} = E_{k''} \circ E_{k'}$ and $D^2_{(k'',k')} = D_{k'} \circ D_{k''}$.

(E, D) is the *underlying* or *base* cryptosystem and (E^2, D^2) is the *doubled* cryptosystem. $\mathcal{M}$ and $\mathcal{C}$ are unchanged, but $\mathcal{K}^2 = \mathcal{K} \times \mathcal{K}$.

The size of the keyspace is squared, resulting in an apparent doubling of the effective key length and making a brute force attack much more costly.

However, *it does not always increase the security* of a cryptosystem as much as one might naïvely think, for other attacks may become possible.

Example: Double Caesar

Consider Double Caesar, the Caesar cipher composed with itself.

It has $26^2 = 676$ possible key pairs (k'', k'). One might hope that double Caesar is more resistant to a brute force attack.

Unfortunately, still only 26 possible distinct encryption functions and only 26 possible decryptions of each ciphertext.

This is because $E^2_{(k'',k')} = E_k$ for $k = (k' + k'') \bmod 26$.

Any attack on the Caesar cipher will work equally well on the Double Caesar cipher. To the attacker, there is no difference between the two systems. Eve neither knows nor cares how Alice actually computed the ciphertext; all that matters are the probabilistic relationships between plaintexts and ciphertexts.

Group property

Let (E, D) be a cryptosystem for which $\mathcal{M} = \mathcal{C}$.
Each E_k is then a *permutation* on $\mathcal{M}$.[1]
The set of all permutations on $\mathcal{M}$ forms a *group*.[2]

Definition

(E, D) is said to have the *group property* if the set of possible encryption functions $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ is closed under functional composition $\circ$.

That is, if $k', k'' \in \mathcal{K}$, then there exists $k \in \mathcal{K}$ such that

$$E_k = E_{k''} \circ E_{k'}.$$

[1] A *permutation* is one-to-one and onto function.

[2] A *group* has an associative binary operation with an identity element, and each element has an inverse.

Cryptosystems with group property

We’ve seen that the Caesar cipher has the group property.

When $\mathcal{E}$ is closed under composition, then $(\mathcal{E}, \circ)$ is a subgroup of all permutations on $\mathcal{M}$. In this case, double encryption adds no security against a brute force attack.

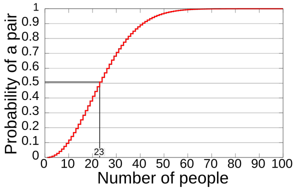
Even though the key length has doubled, the number of distinct encryption functions has not increased, and the double encryption system will fall to a brute force attack on the original cryptosystem.

Birthday Attack

Birthday Problem

The *birthday problem* is to find the probability that two people in a set of randomly chosen people have the same birthday.

This probability is greater than 50% in any set of at least 23 randomly chosen people.



From Wikipedia, "Birthday problem".

23 is far less than the 253 people that are needed for the probability to exceed 50% that at least one of them was born on a specific day, say January 1.[3]

[3] $1 - (364/365)^{252} \approx 0.49910 < 0.5$, but $1 - (364/365)^{253} \approx 0.50048 > 0.5$.

Birthday attack on a cryptosystem

A *birthday attack* is a known plaintext attack on a cryptosystem that reduces the number of keys that must be tried to roughly the square root of what a brute force attack needs.

If for example the original key length was 56 (as is the case with DES), then only about $\sqrt{2^{56}} = 2^{28}$ keys need to be tried.

Any cryptosystem with the group property is subject to a birthday attack.

How a birthday attack works

Assume (m, c) is a known plaintext-ciphertext pair, so $E_{k_0}(m) = c$ for Alice's secret key k_0.

- Choose 2^{28} random keys k_1 and encrypt m using each.
- Choose another 2^{28} random keys k_2 and decrypt c using each.
- Look for a common element u in these two sets.
- Suppose one is found for k_1 and k_2, so $E_{k_1}(m) = u = D_{k_2}(c)$. It follows that $E_{k_2}(E_{k_1}(m)) = c$, so we have succeeded in finding a key pair (k_2, k_1) that works for the pair (m, c).

By the group property, there is a key k such that $E_k = E_{k_2} \circ E_{k_1}$, so $E_k(m) = c$.

How a birthday attack works (cont.)

Alice's key k_0 also has $E_{k_0}(m) = c$. If it happens that $E_k = E_{k_0}$, then we have broken the cryptosystem.

We do not need to find k itself since we can compute E_k from E_{k_1} and E_{k_2} and D_k from D_{k_1} and D_{k_2}.

There are unlikely to be many distinct keys k such that $E_k(m) = c$, so with significant probability we have cracked the system. (For Caesar, there is only one such k.)

Using additional plaintext-ciphertext pairs, we can increase our confidence that we have found the correct key pair. Repeat this process if we have not yet succeeded.

I've glossed over many assumptions and details, but that's the basic idea.

Weakenss of the birthday attack

The drawback to the birthday attack (from the attacker's perspective) is that it requires a lot of storage in order to find a matching element.

Nevertheless, if DES were a group, this attack could be carried out in about a gigabyte of storage, easily within the storage capacity of modern workstations.

(We will see later that DES is not a group.)

Advanced Encryption Standard

New Standard

Rijndael was the winner of NIST's competition for a new symmetric key block cipher to replace DES. An open call for algorithms was made in 1997 and in 2001 NIST announced that AES was approved as FIPS PUB 197.

Minimum requirements:

- Block size of 128-bits
- Key sizes of 128-, 192-, and 256-bits
- Strength at the level of triple DES
- Better performance than triple DES
- Available royalty-free worldwide

Five AES finalists: MARS, RC6, Rijndael, Serpent, and Twofish.

Details

Rijndael was developed by two Belgian cryptographers Vincent Rijmen and Joan Daemen.

Rijndael is pronounced like *Reign Dahl*, *Rain Doll* or *Rhine Dahl*.

Name confusion

- AES is the name of the standard.
- Rijndael is the name of the cipher.
- AES is a restricted version of Rijndael which was designed to handle additional block sizes and key lengths.

More details

AES was a replacement for DES.

- Like DES, AES is an iterated block cipher.
- Unlike DES, AES is not a Feistel cipher.
- Unlike DES, AES can be parameterized.

AES supports key lengths of 128-, 192- and 256-bits.

The algorithm consists of 10 to 14 rounds.

- Number of rounds depends on the key length.
- 10 rounds for 128-bit key, 12 for 192, 14 for 256.

How does AES actually work?

3 Big Ideas:

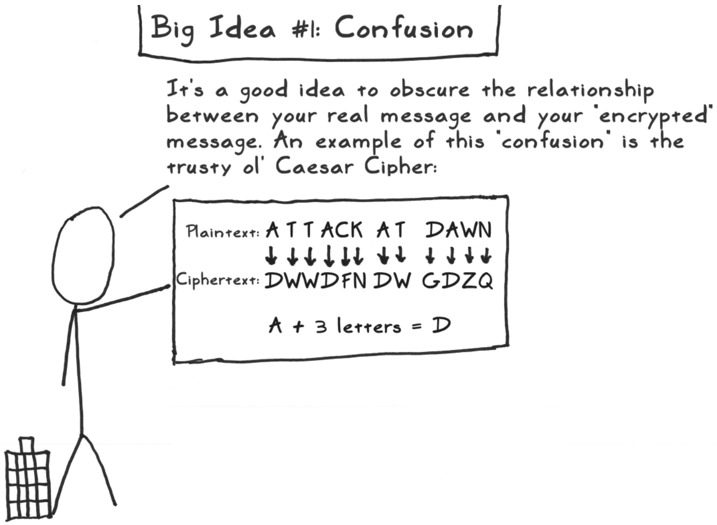
- Big Idea #1: Confusion
- Big Idea #2: Diffusion
- Big Idea #3: Key secrecy

Confusion & Diffusion

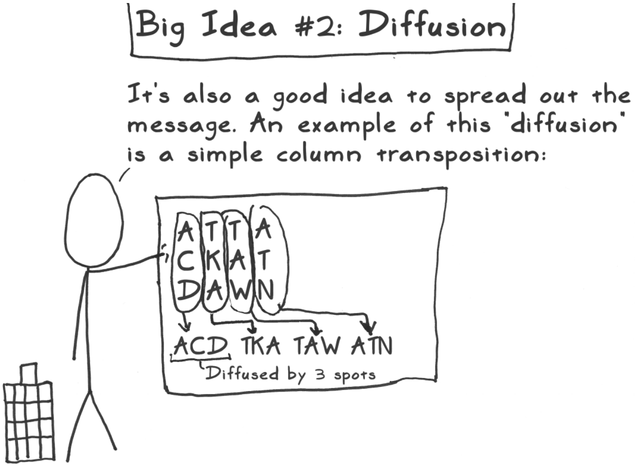
Confusion and *diffusion* are two properties of the operation of a secure cipher which were identified by Claude Shannon in his paper *Communication Theory of Secrecy Systems*[4].

DES, AES and many block ciphers are designed using Shannon's idea of confusion and diffusion.

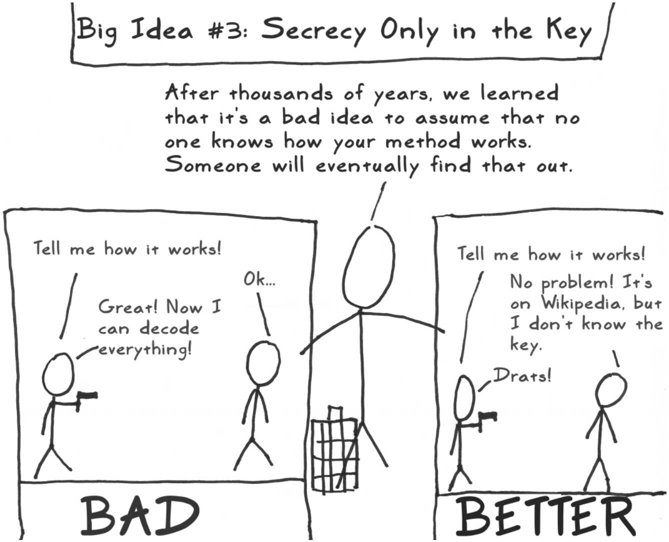
[4] `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf`


Big Idea #1: Confusion
It's a good idea to obscure the relationship between your real message and your "encrypted" message. An example of this "confusion" is the trusty ol' Caesar Cipher:
Plaintext: ATTACK AT DAWN
Ciphertext: DWWDFN DW GDZQ
A + 3 letters = D

www.moserware.com/2009/09/stick-figure-guide-to-advanced.html


Big Idea #2: Diffusion
It's also a good idea to spread out the message. An example of this "diffusion" is a simple column transposition:
A T T A
C K A T
D A W N
ACD TKA TAW ATN
Diffused by 3 spots

www.moserware.com/2009/09/stick-figure-guide-to-advanced.html


Big Idea #3: Secrecy Only in the Key
After thousands of years, we learned that it's a bad idea to assume that no one knows how your method works. Someone will eventually find that out.
Tell me how it works!
Ok...
Great! Now I can decode everything!
BAD
Tell me how it works!
No problem! It's on Wikipedia, but I don't know the key.
Drats!
BETTER

www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Avalanche Effect and Evaluation Criteria[5]

Strict avalanche criterion (SAC) states that when a single input bit i is inverted, each output bit j changes with probability $\frac{1}{2}$, for all i and j.

Bit independence criterion (BIC) states that output bits j and k should change independently when any single input bit i is inverted, for all i, j and k.

[5] Wikipedia: Avalanche Effect

Transformations

Each AES round consists of 4 transformations:

- SubBytes(State)
- ShiftRows(State)
- MixColumns(State)
- AddRoundKey(State, Key)

Each round works on a state array.

A round key is derived from the primary key using a key schedule algorithm.

All four transformations are invertible. *Q: Is it a good thing?*

Roles of the four transformations

SubBytes() replaces bytes using a fixed S-box to achieve non-linearity.

ShiftRow() and *MixColumns()* are intended to mix up bits to achieve a wider distribution of plaintext in the whole message space.

AddRoundKey() provides the necessary secret randomness.

Q: How do these transformations relate to the Big Ideas?

Roles of the four transformations

Big Idea #1 *SubBytes()* replaces bytes using a fixed S-box to achieve non-linearity. *Q: Why non-linearity?*

Big Idea #2 *ShiftRow()* and *MixColumns()* are intended to mix up bits to achieve a wider distribution of plaintext in the whole message space.

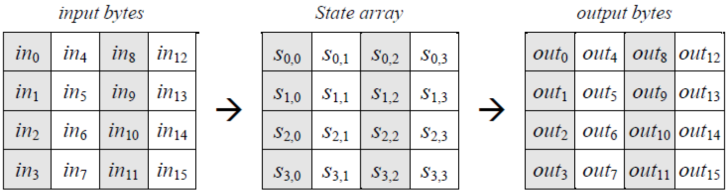
Big Idea #3 *AddRoundKey()* provides the necessary secret randomness.

Q: How do these transformations relate to the Big Ideas?

Preliminaries

We will consider the minimum case of 128-bit key.

- The input and output arrays consist of sequences of 128 bits represented by a 4 x 4 matrix of 8-bit bytes.
- The intermediate state is referred to as the state array.
- Columns and rows are also referred to as words which consist of 4 bytes.

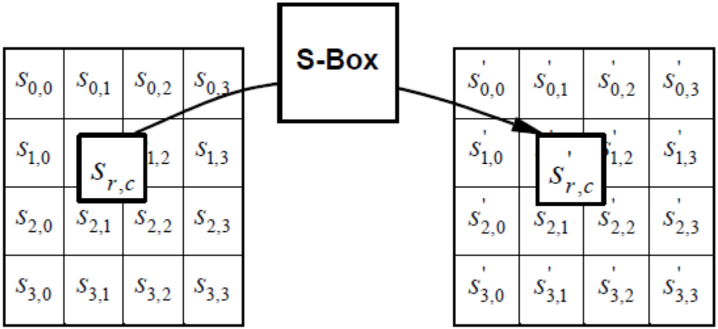


$$\begin{array}{|c|c|c|c|}\hline in_0 & in_4 & in_8 & in_{12} \\ \hline in_1 & in_5 & in_9 & in_{13} \\ \hline in_2 & in_6 & in_{10} & in_{14} \\ \hline in_3 & in_7 & in_{11} & in_{15} \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|}\hline out_0 & out_4 & out_8 & out_{12} \\ \hline out_1 & out_5 & out_9 & out_{13} \\ \hline out_2 & out_6 & out_{10} & out_{14} \\ \hline out_3 & out_7 & out_{11} & out_{15} \\ \hline \end{array}$$

All AES images come from FIPS Pub 197 available at
`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

SubBytes()

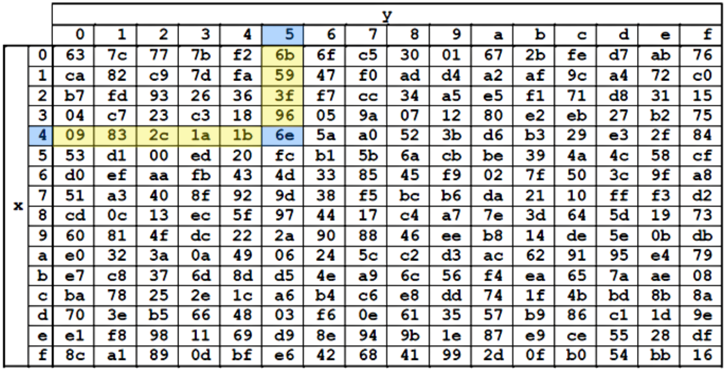
Non-linear byte substitution that operates independently on each byte of the state array using a substitution table (S-box).


S-Box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{r,c}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$
$s'_{r,c}$

SubBytes() S-box

Example: SubBytes(45) = 6e

- Rows: First 4 bits of the input byte
- Columns: Last 4 bits of input



		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

SubBytes()

Each non-zero byte x is substituted using the following transformation $y = Ax^{-1} + b$

- If x is a zero byte, then $y = b$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

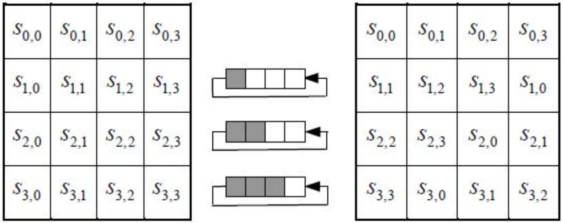
S-box is just a pre-computed table of inverses. It eliminates the possibility of a timing analysis attack:

- Observing the time difference may give out whether an operation is performed on a zero or a non-zero byte.

ShiftRows()

The bytes are cyclically shifted over by 0, 1, 2 and 3 bytes.

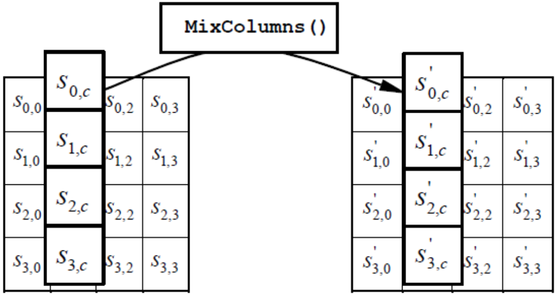
This operation works like a transposition cipher because only the positions of bytes are changed, not the bytes themselves.


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

MixColumns()

Operates on the state array column-by-column.

Each column is multiplied by a fixed array.


MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

Matrix multiplication

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

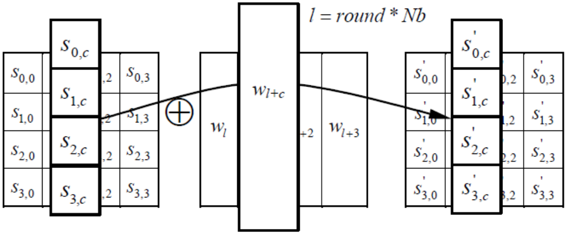
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

$\oplus$ exclusive OR operation, $\bullet$ finite field multiplication

AddRoundKey()

Each column of the state array is XORed with a word from the key schedule.

The round key is determined by the key schedule algorithm.


$l = round * Nb$
$s_{0,0}$ $s_{0,c}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,3}$
$\oplus$
w_l w_{l+c} w_{l+3}
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,3}$

Nb - number of columns, here Nb = 4

Decryption

AES is not a Fiestel cipher.

Q: How does it affect the decryption process?

Decryption

AES is not a Fiestel cipher so decryption works differently than encryption. Steps are done in reverse.

All four transformations (SR, SB, MC, ARK) are invertible.

- InvShiftRows()
- InvSubBytes()
- InvMixColumns()
- AddRoundKey()

Decryption

- InvShiftRows() - bytes in the last three rows of the state array are cyclically shifted over to the right
- InvSubBytes() - the inverse S-box is applied to each byte of the state array
- InvMixColumns() - the state array is multiplied by the matrix inverse used in MixColumns()
- AddRoundKey() is its own inverse, since it is an XOR operation

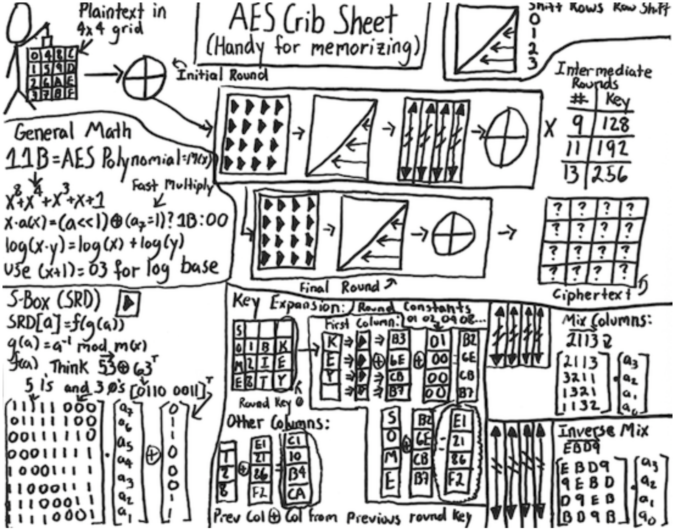
Encryption

- ARK
- BS, SR, MC, ARK
- ...
- BS, SR, MC, ARK
- BS, SR, ARK

Decryption

- ARK, ISR, IBS
- ARK, IMC, ISR, IBS
- ...
- ARK, IMC, ISR, IBS
- ARK

MixColumns() is not applied in the last round in order to make the encryption and decryption more similar in structure. This is similar to the absence of the swap operation in the last round of the DES.



www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Additional Resources

A Stick Figure Guide to AES by Jeff Moser Highly recommended!

`http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html`

AES Inspector by Enrique Zabala

`http://www.formaestudio.com/rijndaelinspector/archivos/inspector.html`

AES Animation by Enrique Zabala

`http://www.formaestudio.com/rijndaelinspector/archivos/rijndaelanimation.html`

AES Example by instructors at Massey U., New Zealand

`http://www.box.net/shared/static/uqrq0hmnb9.pdf`