Lecture (09)
Advanced Encryption Standard

Dr. Ahmed M. ElShafee

Introduction

- The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001.
- AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications.
- The AES cipher (& other candidates) form the latest generation of block ciphers, and now we see a significant increase in the block size - from the old standard of 64-bits up to 128-bits; and keys from 128 to 256-bits.
- Whilst triple-DES is regarded as secure and well understood, it is slow, especially in s/w.

- In a first round of evaluation, 15 proposed algorithms were accepted.
- A second round narrowed the field to 5 algorithms.
- NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001.
- NIST selected Rijndael as the proposed AES algorithm.
- The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr.Vincent Rijmen.

AES selection, the competition

- private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- stronger & faster than Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST SuiteB suggests AES128 for secret data and AES256 for top secret data

AES Evaluation Criteria

- When NIST issued its original request for candidate algorithm nominations in 1997, the request stated that candidate algorithms would be compared based on the factors shown in Stallings Table5.1, which were used to evaluate field of 15 candidates to select shortlist of 5. initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics

- The final criteria evolved during the evaluation process, and were used to select Rijndael from that short-list,
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Shortlist

- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- Notice the mix of commercial (MARS, RC6, Twofish) verses academic (Rijndael, Serpent) proposals, sourced from various countries.
- All were thought to be good – it came down to the best balance of attributes to meet criteria, in particular the balance between speed, security & flexibility.

The AES Cipher - Rijndael

- The Rijndael has block length and the key length can be independently specified to be 128,192,or 256 bits, while the AES specification uses the same three key size alternatives but limits the block length to 128 bits.
- Rijndael is an academic submission, based on the earlier Square cipher, from Belgium academics Dr Joan Daemen and Dr Vincent Rijmen.
- It is an **iterative** cipher (operates on entire data block in every round) rather than feistel (operate on halves at a time)

- It was designed to have characteristics of:
  - Resistance against all known attacks,
  - Speed and code compactness on a wide range of platforms, &
  - Design simplicity.

RIJNDAEL, the AES structure

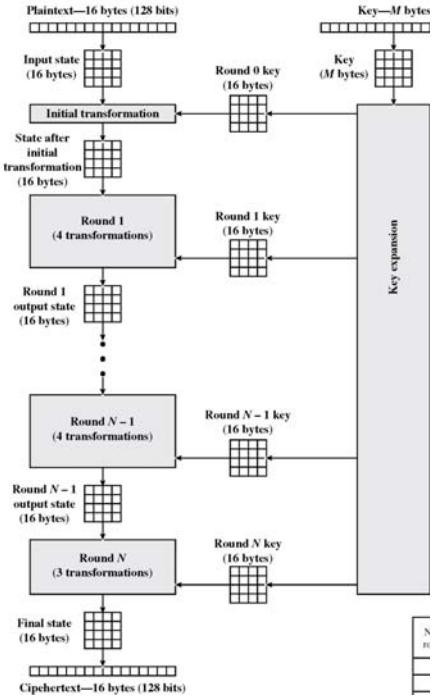
Block length/ key length

- Multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits

- Block length = 128, 192, and 256 bits.
- Key lengths = 128, 192, and 256 bits.

- Due to fixed block length of 128 bits = 16 bytes, Rijndael uses 4×4 matrix calls state
- Rijndael versions of larger key length, adds columns to state.

Overall structure



The state, the cipher key and the number of rounds

- Block mapped to array of 4 rows × N_b columns.
- Key is mapped to array of 4 rows × N_k columns.

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	$A_{0,4}$	$A_{0,5}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$	$A_{1,4}$	$A_{1,5}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$	$A_{2,4}$	$A_{2,5}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$	$A_{3,4}$	$A_{3,5}$

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

- example of state (with $N_b = 6$) and cipher key (with $N_k = 4$) layout
- State may be considered as 4 bytes vectors (a ,b ,c ,d).

Rijndael input/ output

- Input is an array of bytes, length = ((4 × *Nb*)-1)
- Output is the same length as input.

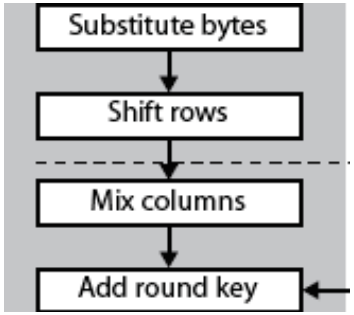
Rijndael rounds (N_r)

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

The round transformation

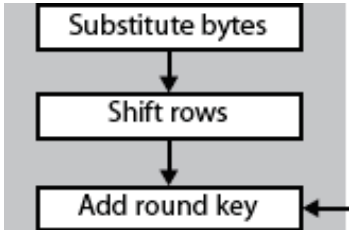
- **All rounds**

Round (*State, RoundKey*)
{
ByteSub (*State*);
ShiftRow (*State*);
MixColumn (*State*);
AddRoundKey (*State, RoundKey*);
}



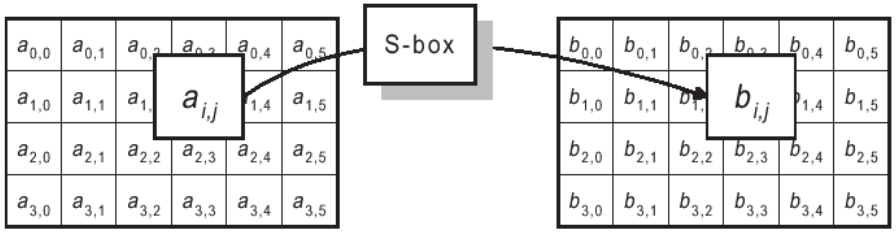
- **Last round**

FinalRound (*State, RoundKey*)
{
ByteSub (*State*);
ShiftRow (*State*);
AddRoundKey (*State, RoundKey*);
}



The ByteSub transformation

- a simple substitution of each byte
- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
- S-box may be constructed using defined transformation of values in GF(28)
- designed to be resistant to all known attacks



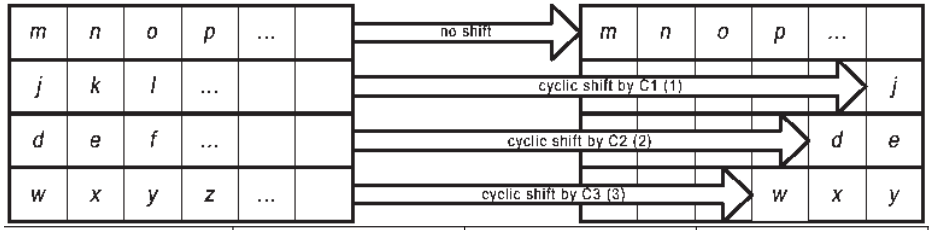
x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	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
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

(a) S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

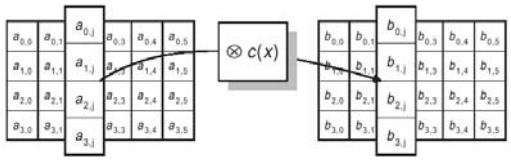
(b) Inverse S-box

The ShiftRow Transformation

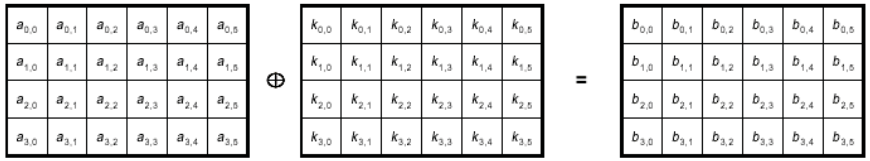


N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

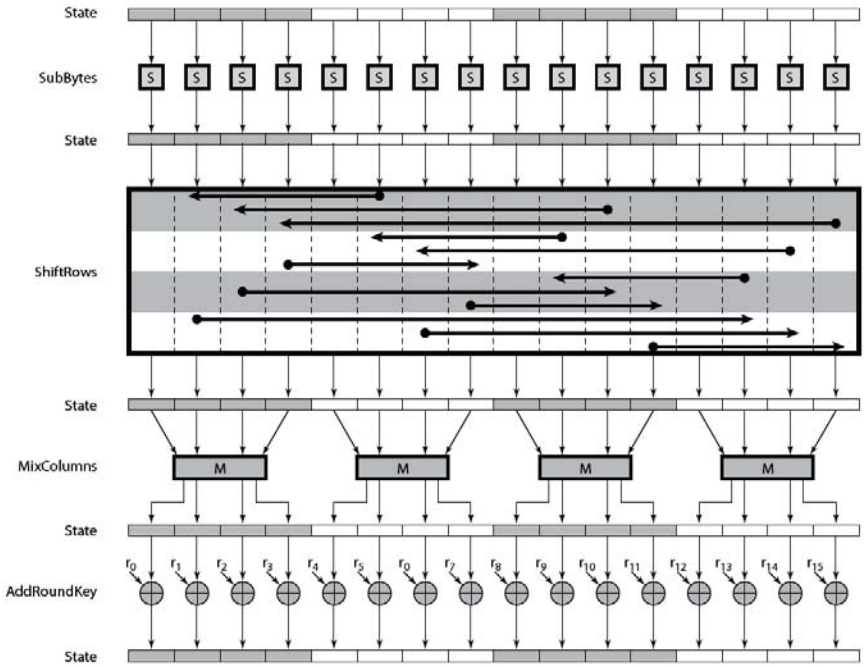
The MixColumn Transformation

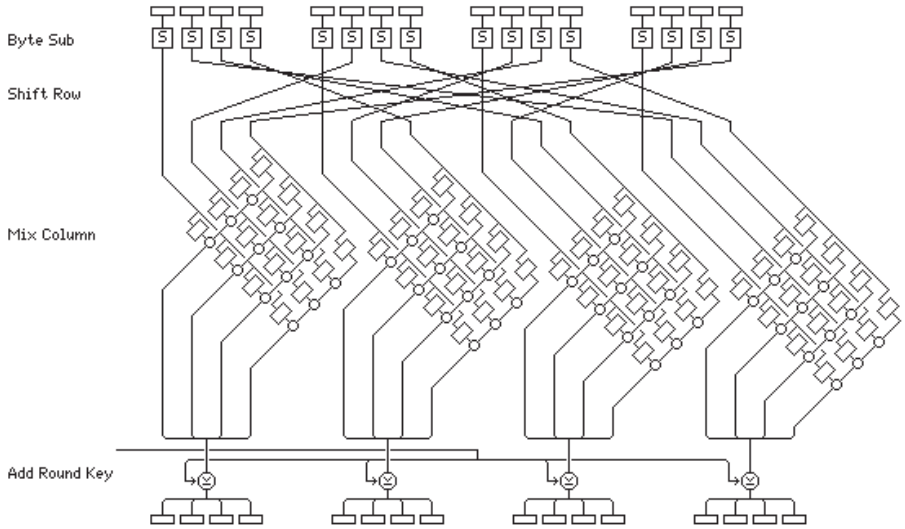


- **The Round Key Addition**



- Regardless the user key length, round key (generated) equals to N_b





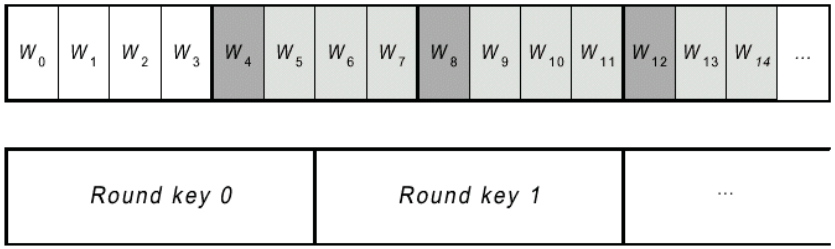
Key Schedule

- 3 simple rules;

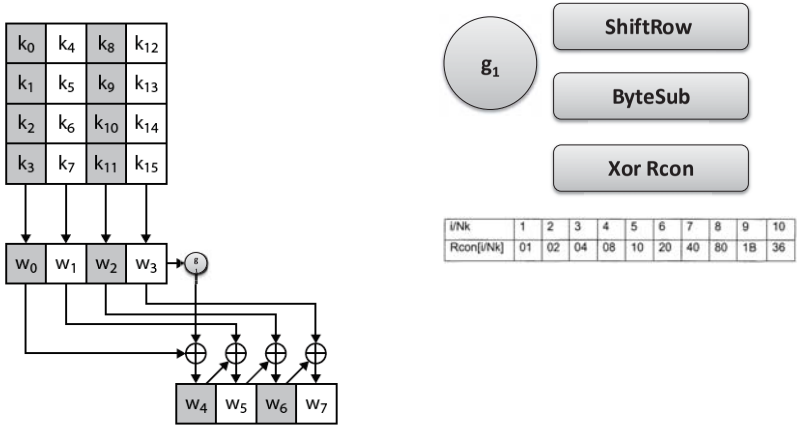
1. expanded key length (bits) = (block length * number of rounds+1)
2. User key (cipher) expanded to expanded key
3. Expanded key divided into N_r round keys in ascending order

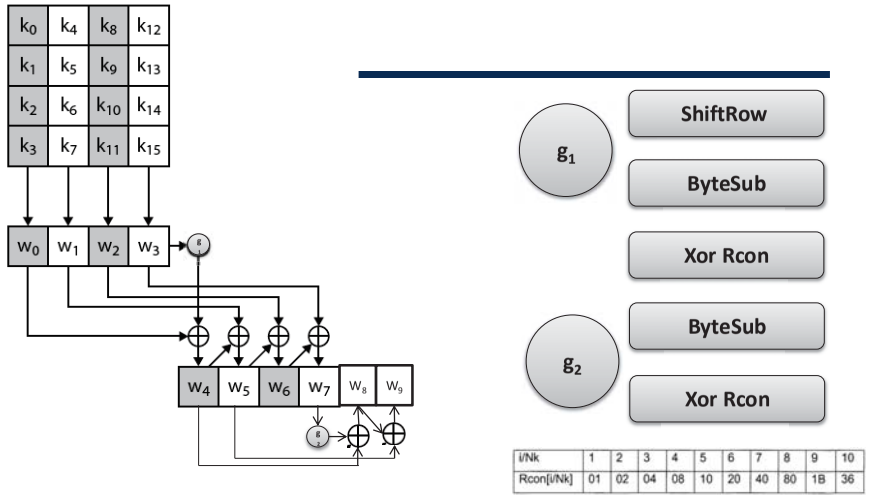
- Generated as; $W[N_b \times (N_r+1)]$.
- W is 4 bytes array.

- **Round Key Selection**



- Key expansion and round key selection for $N_b = 6$ and $N_k = 4$





$N_k < 6$

```
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]);
    For (I = Nk ; I < (Nb × (Nr + 1)); I++)
            {
            temp = W[I – 1];
            if ( I % Nk == 0)
                    temp = SubByte (RotByte (temp)) ^ Rcon [I / Nk];
            W[I] = W[I – Nk] ^ temp;
            }
    }
```


$N_k >= 6$

```
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]);
    For (I = Nk; I < (Nb × (Nr + 1)); I++)
            {
            temp = W[I – 1];
            if ( I % Nk == 0)
                    temp = SubByte (RotByte (temp)) ^ Rcon [I / Nk];
            else if ((I % Nk == 4)
                    temp = SubByte(temp);
            W[I] = W[I – Nk] ^ temp;
            }
    }
```


AES Decryption

The Cipher

RIJNDAEL (*State, CipherKey*)

{

KeyExpansion (*CipherKey, ExpandedKey*);

AddRoundKey (*State, RoundKey*);

For (*I* = 0; *I* < N_r ; *I*++)

Round (*State, ExpandedKey* + $N_b \times I$);

FinalRound (*State, ExpandedKey* + $N_b \times N_r$);

}

The Inverse Cipher

InverseRIJNDAEL (*State, CipherKey*)

{

KeyExpansion (*CipherKey, ExpandedKey*);

InverseFinalRound (*State, ExpandedKey* + $N_b \times N_r$);

For (*I* = 0; *I* < N_r ; *I*++)

InverseRound (*State, ExpandedKey* + $N_b \times I$);

AddRoundKey (*State, RoundKey*);

}

Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR's
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR's

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

RIJNDAEL cryptanalysis

- **Brute Force Attack**

Key length	Number of Trials
128 bits	$2^{128} \approx 3.4 \times 10^{38}$
192 bits	$2^{192} \approx 6.4 \times 10^{57}$
256 bits	$2^{256} \approx 1.16 \times 10^{77}$

Linear Cryptanalysis

- By 2006,…

Key length	Number of rounds	Broken number of rounds
128	10	7
192	12	8
256	14	9

- **RIJNDAEL Algebraic Description Concerns**
- Rijndael has a neat algebraic description.
- This has not yet led to any attacks some cryptanalyst believe that complicated algebraic description is required.

XSL Attack

- announced by “Nicolas Courtois” and “Josef Pieprzyk”, in 2002. Some famous cryptographers faced problems in the underlying mathematics of the proposed attack, so they think authors may mistaken in some estimates. So it remains an open question.

Side Channel Attacks

- Side channel attacks do not attack the underlying cipher,
- but attack implementations of the cipher on systems which inadvertently leak data.

- In April 2005, “D.J. Bernstein” announced a cache timing attack that he used to break a custom server that used OpenSSL's AES encryption.
- The custom server was designed to give out as much timing information as possible, and the attack required over 200 million chosen plaintexts.
- Some say the attack is not practical over the internet with a distance of one or more hops;
- Bruce Schneier called the research a "nice timing attack".
- In October 2005, "Dag Arne Osvik", "Adi Shamir" and "Eran Tromer" presented a paper demonstrating several cache timing attacks against AES. One attack was able to obtain an entire AES key after only 800 writes.

Simplified AES example

Dr. Ahmed M. ElShafee

Example 01

- Encrypt the following message “**tomorrow never die**” using simplified AES operates on 26 English plaintext characters, using the following key “**simpler than des**”
- Sbox: pdqjkfvobwselcmtirhgnyxazu

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

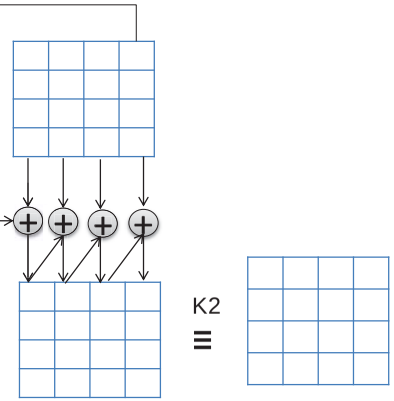
simpler than des

Key generation

k1

s	i	m	p
l	e	r	t
h	a	n	d
e	s	x	x

≡



K2

≡

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

simpler than des

Key generation

k1

s	i	m	p
l	e	r	t
h	a	n	d
e	s	x	x

≡

18	8	12	15
11	4	17	19
7	0	13	3
4	18	23	23

7	15	1	16
4	8	25	18
10	10	23	0
1	19	16	13

K2

≡

h	p	b	q
e	i	z	s
k	k	x	a
b	t	q	n

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

tomorrow never die

PlainState

Enc 1st Round

byteSub

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

ShiftRow

MixColumn

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

=

18	8	12	15
11	4	17	19
7	0	13	3
4	18	23	23

K1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

PlainState

t	o	m	o
r	r	o	w
n	e	v	e
r	d	i	e

Enc 1st Round

byteSub

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

ShiftRow

MixColumn

6	12	11	12
23	17	17	12
24	10	2	10
9	1	10	17

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

=

11	21	10	12
23	6	7	4
19	0	3	11
6	22	20	19

18	8	12	15
11	4	17	19
7	0	13	3
4	18	23	23

K1

19	13	24	23
12	2	16	11
12	0	16	8
2	4	23	22

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

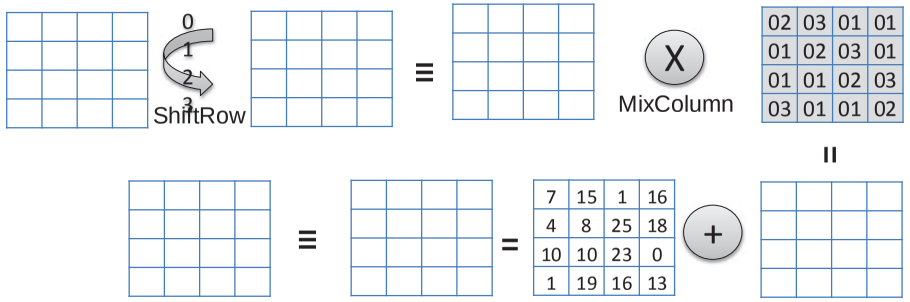
PlainState

l	v	k	m
x	g	h	e
t	a	d	l
g	w	u	t

byteSub

Enc 2nd Round

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

PlainState

l	v	k	m
x	g	h	e
t	a	d	l
g	w	u	t

byteSub

Enc 2nd Round

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

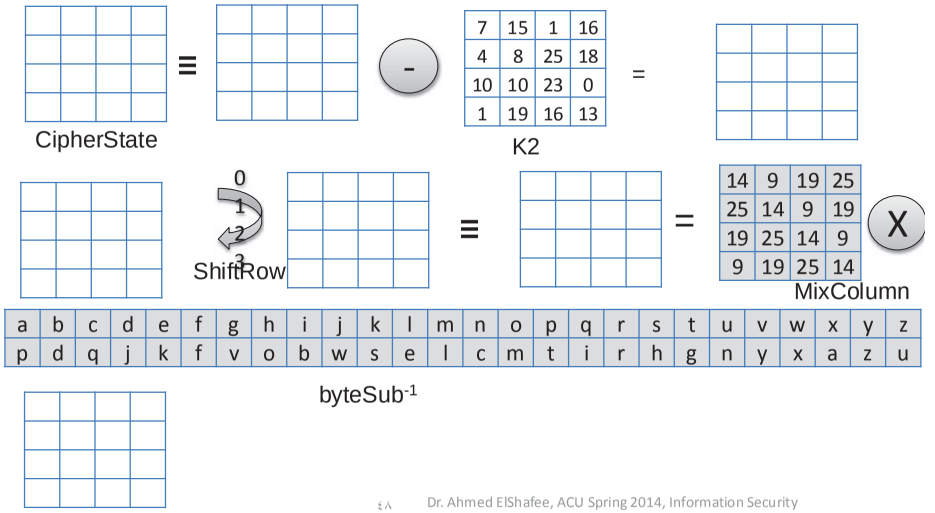
Example 02

- Decryot the following message "**uhqhhiogepdtoaja**" using simplified AES operates on 26 English plaintext characters, using the following key "**simpler than des**"
- Sbox: pdqjkfvobwselcmtirhgnyxazu

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

uhqhhiogepdtoaja

Dec 1st Round



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

uhqhhiogepdtoaja

Dec 1st Round

u	h	q	h
h	i	o	g
e	p	d	t
o	a	j	a

CipherState

≡

20	7	16	7
7	8	14	6
4	15	3	19
14	0	9	0

-

7	15	1	16
4	8	25	18
10	10	23	0
1	19	16	13

K2

=

13	18	15	17
3	0	15	14
20	5	6	19
13	7	19	13

e	b	e	a
s	t	q	l
w	f	o	h
f	d	b	r

ShiftRow (0, 1, 2, 3)

e	b	e	a
l	s	t	q
o	h	w	f
d	b	r	f

≡

4	1	4	0
11	18	19	16
14	7	22	5
3	1	17	5

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

l	i	l	x
k	p	c	m
j	f	h	s
f	b	i	r

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Dec 2nd Round

l	i	l	x
k	p	c	m
j	f	h	s
f	b	i	r

CipherState

≡

-

18	8	12	15
11	4	17	19
7	0	13	3
4	18	23	23

K1

=

ShiftRow (0, 1, 2, 3)

≡

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Dec 2nd Round

l	i	l	x
k	p	c	m
j	f	h	s
f	b	i	r

CipherState

≡

11	8	11	23
10	15	2	12
9	5	7	18
5	1	8	17

-

18	8	12	15
11	4	17	19
7	0	13	3
4	18	23	23

K1

=

19	0	25	8
25	11	11	19
2	5	20	15
1	9	11	20

h	m	b	g
b	h	d	r
k	p	s	p
d	e	k	a

ShiftRow (0, 1, 2, 3)

h	m	b	g
r	b	h	d
s	p	k	p
e	k	a	d

≡

7	12	1	6
17	1	7	3
18	15	10	15
4	10	0	3

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

s	o	i	t
i	s	b	r
e	a	k	a
b	l	e	x

So it is breakable x

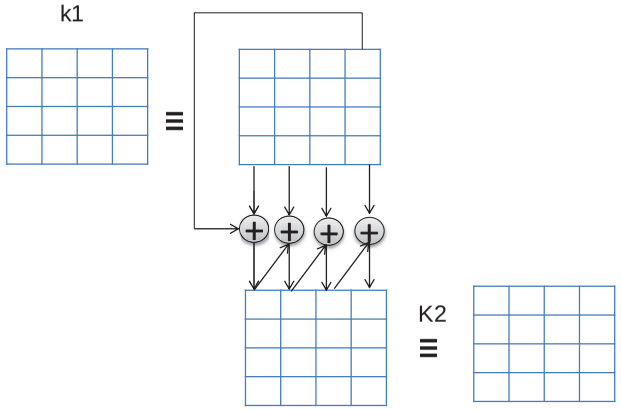
Example 03

- Encrypt the following message "**play hide and seek**" using simplified AES operates on 26 English plaintext characters, using the following key "**my hidden secret key**"
- Sbox: pdqjkfvobwselcmtirhgnyxazu

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Key generation

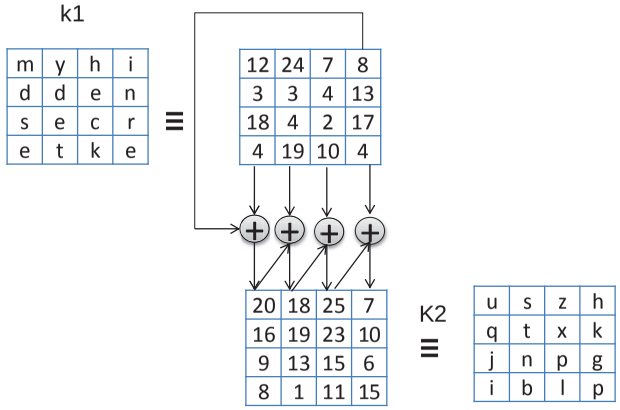
my hidden secret key



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Key generation

my hidden secret key



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

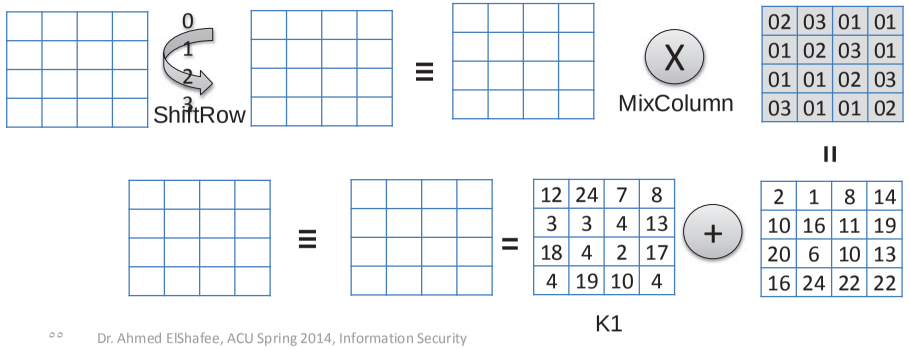
PlainState

play hide and seek

Enc 1st Round

byteSub

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

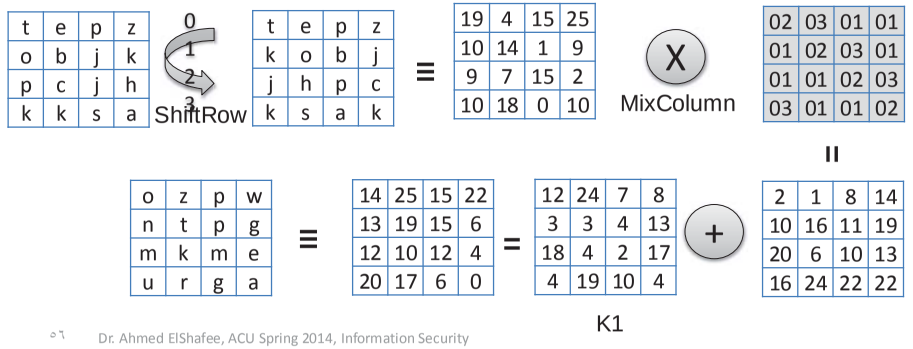
PlainState

play hide and seek

Enc 1st Round

byteSub

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

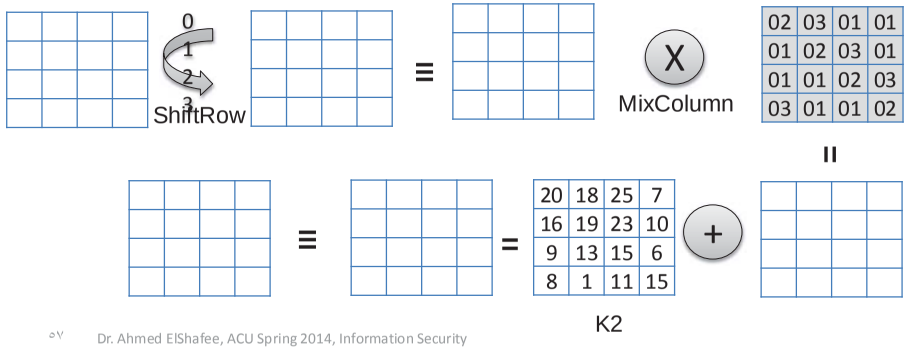
PlainState

o	z	p	w
n	t	p	g
m	k	m	e
u	r	g	a

byteSub

Enc 2nd Round

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

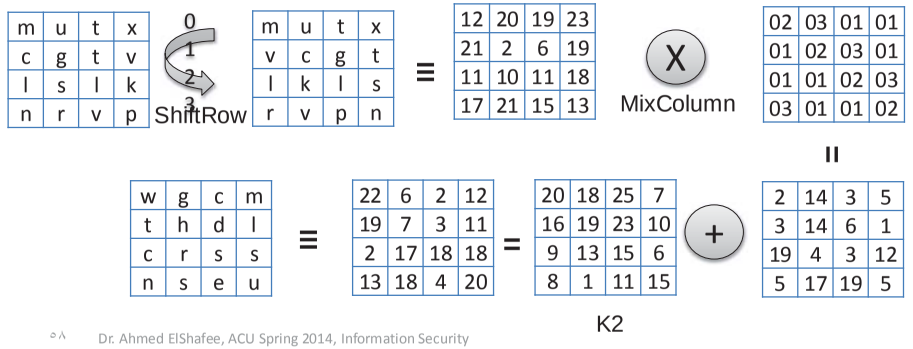
PlainState

o	z	p	w
n	t	p	g
m	k	m	e
u	r	g	a

byteSub

Enc 2nd Round

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u



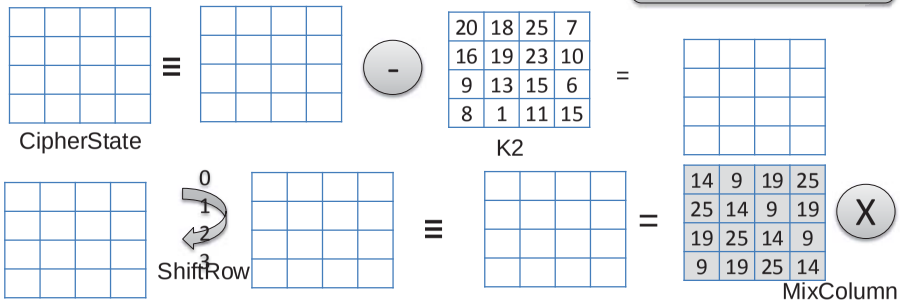
Example 04

- Decrypt the following message "**ltnyobbbswbgrmxj**" using simplified AES operates on 26 English plaintext characters, using the following key "**my hidden secret key**"
- Sbox: pdqjkfvobwselcmtirhgnyxazu

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

ltnyobbbswbgrmxj

Dec 1st Round



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

ltnyobbbswbgrmxj

Dec 1st Round

l	t	n	y
o	b	b	b
s	w	b	g
r	m	x	j

CipherState

≡

11	19	13	24
14	1	1	1
18	22	1	6
17	12	23	9

-

20	18	25	7
16	19	23	10
9	13	15	6
8	1	11	15

K2

=

17	1	14	17
24	8	4	17
9	9	12	0
9	11	12	20

g	i	r	c
x	v	m	l
e	k	h	n
q	d	f	c

0 1 2 3 ShiftRow

g	i	r	c
l	x	v	m
h	n	e	k
d	f	c	q

≡

6	8	17	2
11	23	21	12
7	13	4	10
3	5	2	16

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

t	q	r	n
w	g	o	m
l	e	s	u
c	b	f	n

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Dec 2nd Round

t	q	r	n
w	g	o	m
l	e	s	u
c	b	f	n

CipherState

≡

-

K1

=

0 1 2 3 ShiftRow

≡

3	10	14	1
2	2	9	10
25	4	10	11
2	10	7	1

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Dec 2nd Round

t	q	r	n
w	g	o	m
l	e	s	u
c	b	f	n

CipherState

≡

19	16	17	13
22	6	14	12
11	4	18	20
2	1	5	13

-

12	24	7	8
3	3	4	13
18	4	2	17
4	19	10	4

K1

=

7	18	10	5
19	3	10	25
19	0	16	3
24	8	21	9

d	k	o	b
c	j	k	c
k	l	z	e
b	c	k	h

0 1 2 3 ShiftRow

d	k	o	b
c	c	j	k
z	e	k	l
c	k	h	b

≡

3	10	14	1
2	2	9	10
25	4	10	11
2	10	7	1

=

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

X

MixColumn

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}

b	e	h	i
n	d	e	n
e	m	y	l
i	n	e	s

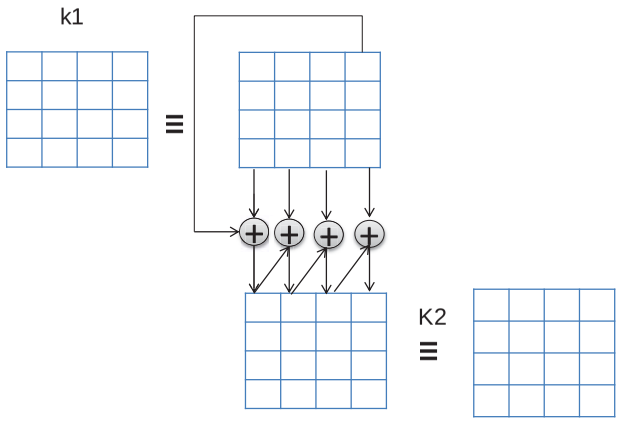
Behind enemy lines

Thanks,..

See you next week (ISA),...

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Key generation



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

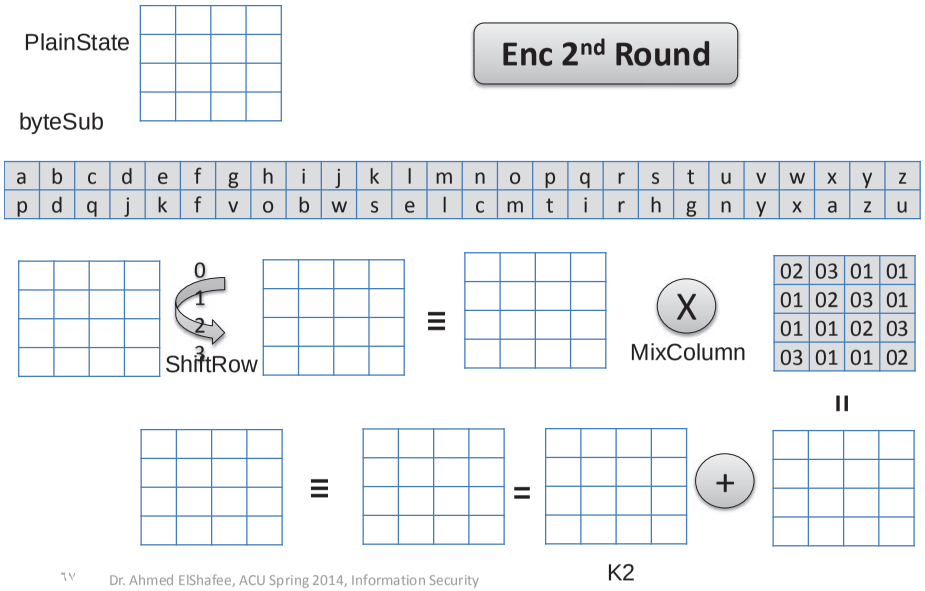
Enc 1st Round

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Enc 2nd Round



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

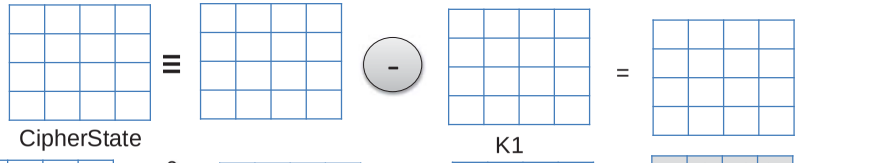
Dec 1st Round

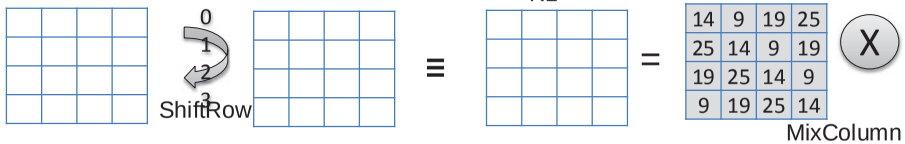
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

14	9	19	25
25	14	9	19
19	25	14	9
9	19	25	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Dec 2nd Round





a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
p	d	q	j	k	f	v	o	b	w	s	e	l	c	m	t	i	r	h	g	n	y	x	a	z	u

byteSub^{-1}