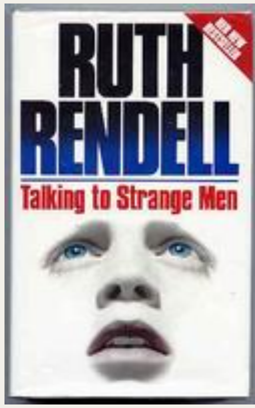
COM 5335 NETWORK SECURITY LECTURE 4 ADVANCED ENCRYPTION STANDARD

Scott CH Huang

Advanced Encryption Standard



- “It seems very simple.”
- “It is very simple. But if you don't know what the key is it's virtually indecipherable.” ~ Talking to Strange Men, Ruth Rendell

Origins

- A replacement for DES was needed
  - *Exhaustive key search attacks can break DES*
- An alternative: 3DES
  - *Very Slow*
- Some History
  - *National Institute of Standards and Technology (NIST) issued a call for ciphers in 1997*
  - *15 candidates accepted in Jun 1998*
  - *5 were shortlisted in Aug-1999*
  - *Rijndael was selected as the AES in Oct-2000*
  - *NIST issued as FIPS PUB 197 standard in Nov-2001*

AES Requirements

- Private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger & faster than 3DES
- Active life of 20-30 years (+ archival use)
- Provide full specification & design details
- Both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Shortlist

- After testing and evaluation, 5 were shortlisted 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) – clean, slow, v. high security margin*
  - *Twofish (USA) - complex, v. fast, high security margin*
- Then subject to further analysis & comment
- Contrast between
  - *few complex rounds and many simple rounds*
  - *refined existing ciphers and new proposals*

Mathematical Conventions

- Each byte corresponds to a polynomial and is represented by either a binary or hex number.
- For example, 'x^6+x^3+x+1' is represented by 01001011 in binary or {4B} in hex.
- ADD & MULT are done in GF(28) with generating polynomial $m(x)=x^8+x^4+x^3+x+1$.
  - *ADD & MULT are done mod m(x).*

Multiplication by x

- If the highest bit is 0 (i.e. x^7 doesn't appear), we simply do a left shift.
- If the highest bit is 1 (i.e. x^7 appears), after a left shift we have to perform mod m(x), which will be equivalent to performing XOR with 0001 1011 (or {1B}).

General Multiplication

- Multiplication by xi can be done by calling ‘MULT by x’ i times.
- General multiplications can be done by performing ‘MULT by xi’ one by one and add the results up.
- For example, multiplying ‘x^6+x^3+x’ can be done by multiplying ‘x^6’ , ‘x^3’ , and ‘x’ separately and add them up.

Winner : Belgium Team Rijndael

- Designed by Vincent Rijmen and Joan Daemen
- Using 128/192/256 bit keys, 128 bit data
- An iterative rather than Feistel cipher
  - *Treating data in 4 groups of 4 bytes*
  - *Operating an entire block in every round*
- Designed to be:
  - *Resistant against known attacks*
  - *Efficient on many CPUs*
  - *Simple*

Rijndael

- Processing data as 4 groups of 4 bytes (state)
- Consisting of 9/11/13 rounds in which the state undergoes 4 stages:
  - *Byte substitution (1 S-box used on every byte)*
  - *Shift rows (permute bytes between groups/columns)*
  - *Mix columns (subs using matrix multiplication of groups)*
  - *Add round key (XOR state with key material)*
- Initial XOR key material & incomplete last round
- All operations combined into XOR (substitution with keys) and table lookups (permutation) - efficiency

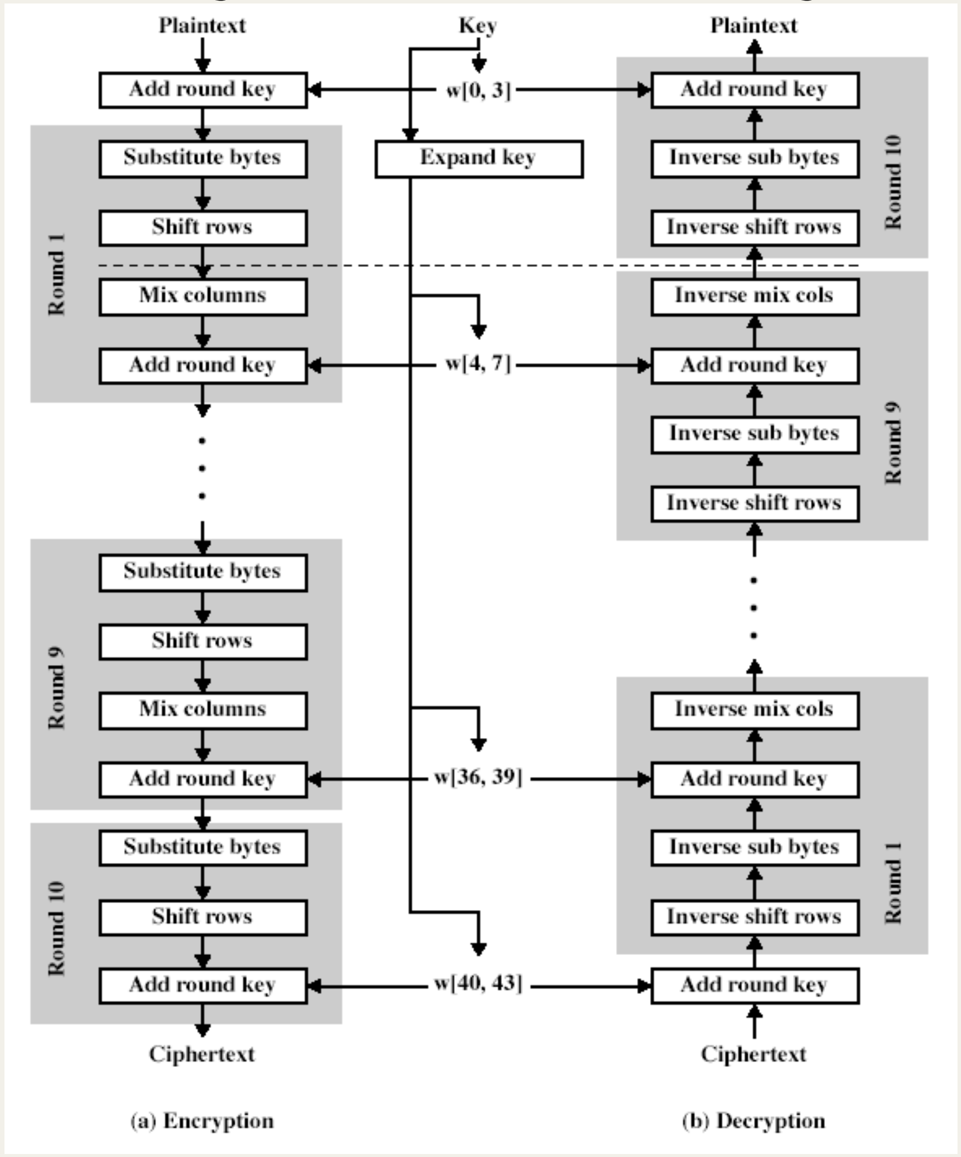
Rijndael Parameters

Key size (bytes)	16	24	32
Plaintext block size (bytes)	16	16	16
Number of rounds	10	12	14
Round key size (bytes)	16	16	16
Expanded key size (bytes)	176	208	240

$\uparrow$

AES

AES Encryption/Decryption



The Cipher

- The AES cipher consists of
- An initial Round Key addition
- Nr-1 rounds
- A final round.

State, Cipher Key, and Number of Rounds

- Different transformations operate on intermediate results, called states.
- Definition: The intermediate cipher result is called the State.
- The State can be pictured as a rectangular array of bytes (8 bits).
  - *Each component is a byte (8 bits)*
  - *There are 4 rows*
  - *The number of columns is denoted by Nb*
  - *Nb = the block length divided by 32 [=4 (rows) * 8 bits (for a byte)]*

An example for *Nb*=4 (a block of 128 bits)

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

An example for *Nb*=6 (a block of 192 bits)

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

An example for Nk=4 (key of length 128)

The Cipher Key is similarly pictured as a rectangular array with 4 rows. The number of columns of the Cipher Key is denoted by Nk and it equal to the key length divided 32.

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

- These blocks can be considered as one dimensional arrays of 4-byte vectors.
- They are sometimes referred to as *words*.

- The number of rounds is denoted by *Nr* , which depends on *Nb* and *Nk*.

Nr	*Nb*=4	*Nb*=6	*Nb*=8
Nk=4	10	12	14
Nk=6	12	12	12
Nk=8	14	14	14

Each Round Transformation

- The round transformation is composed of four different transformations.

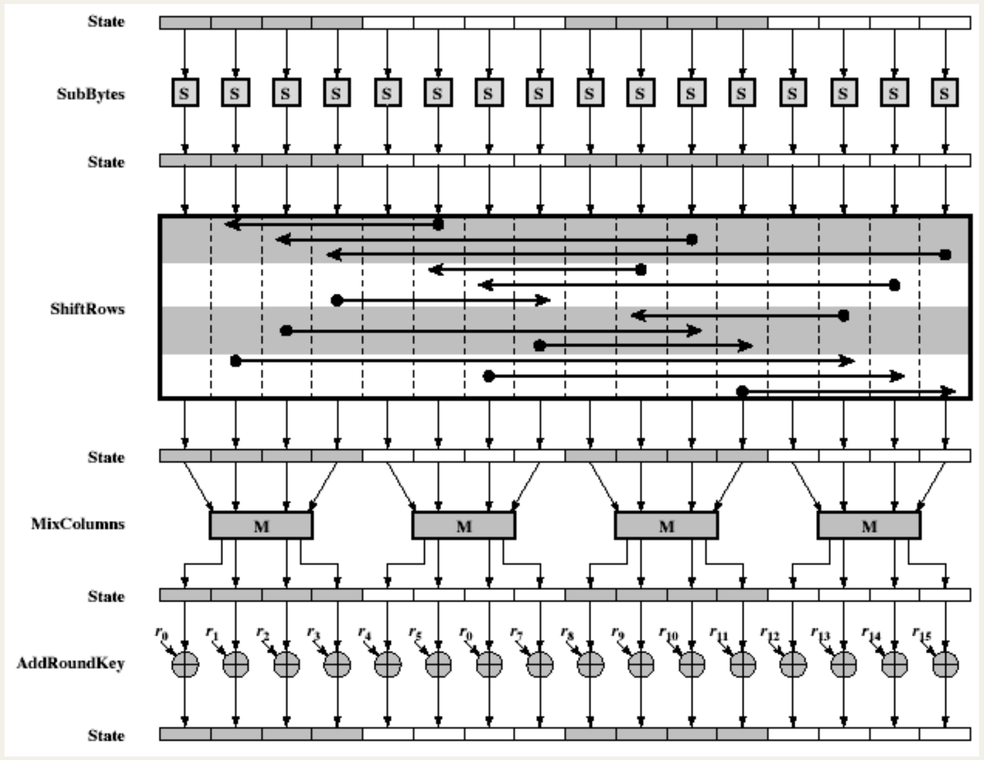
```
Round (State, Roundkey)
{
        ByteSub (State);
        ShiftRow (State);
        Mixcolumn (State);
        AddRoundKey(State, RoundKey);
}
```

- The final round of AES is slightly different. It is defined by:

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

- In this notation, the “function” (Round, ByteSub, ShiftRow, …) operate on two arrays of State, RoundKey.

AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r_0 r_1 r_2 r_3 r_4 r_5 r_0 r_7 r_8 r_9 r_{10} r_{11} r_{12} r_{13} r_{14} r_{15}
State

Byte Substitution

- A simple substitution for each byte
- It uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- Each byte of state is replaced by a byte in row (left 4-bits) & column (right 4-bits) independently
  - *e.g. byte {95} is replaced by {2A}*
- S-box is constructed using a defined transformation of the values in GF(2^8)
- It's designed to be resistant to all known attacks

AES S-Box

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

ByteSub transformation

- Equivalently, we can take the multiplicative inverse in GF(28) of each component of State.
  - *As a special case, '00' is mapped onto itself.*
  - *Note: the inverse is derived from multiplication of corresponding polynomials mod $m(x)=x^8+x^4+x^3+x+1$*
- After that, an affine transformation over GF(2) will be applied.

Affine Transformation over GF(2) in ByteSub

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

ByteSub Example

ByteSub for {95}

1. Find $\{95\}^{-1}$ = {8A} = 1000 1010 $\xrightarrow{}$ $\begin{matrix}0\\1\\0\\1\\0\\0\\0\\1\end{matrix}$

   $\overset{x_0}{\downarrow}$

2. Apply the affine transformation over GF(2) below

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

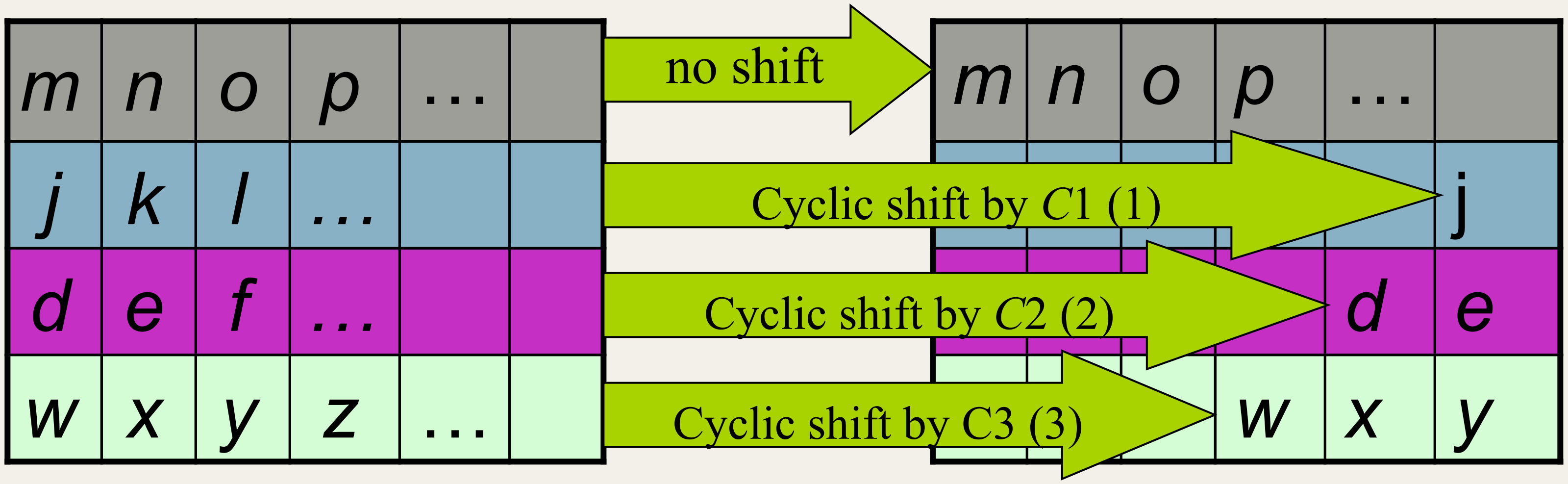
y = 01001001 + 01100011 = 00101010 = {2A}, ByteSub({95}) = {2A}

Shift Rows

- In ShiftRow, the rows of the State are cyclically shifted (left) over different offsets.
  - *Row 0 is not shifted,*
  - *Row 1 is shifted over C1 bytes,*
  - *Row 2 is shifted over C2 bytes*
  - *Row 3 over C3 bytes.*
- The shift offsets depend on the block length Nb and are specified on the next page.
- Decryption does the opposite: shifts to right.
- Since state is processed by columns, this step permutes bytes between columns.

Shift Offsets for Different *Nb*'s

Nb	*C*1	*C*2	*C*3
4	1	2	3
6	1	2	3
8	1	3	4



ShiftRow operates on the rows of the State.

Mix Columns

- Each column is processed separately.
- Each byte is replaced by a value dependent on all 4 bytes in the column.
- Matrix multiplication in GF(2^8) w.r.t. $m(x) = x^8+x^4+x^3+x+1$.

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

Add Round Key

- XOR state with 128-bits of the round key.
- Again processed by column (though effectively a series of byte operations).
- Inverse for decryption is identical since XOR is own inverse, just with correct round key.
- Designed to be as simple as possible.

In the Round Key addition the Round Key is bitwise XOR-ed to the State.

$$\begin{array}{|c|c|c|c|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\ \hline \end{array}$$

- AddRoundKey is its own inverse.
- The Round Key is derived from the Cipher Key by means of the key schedule to be addressed next.

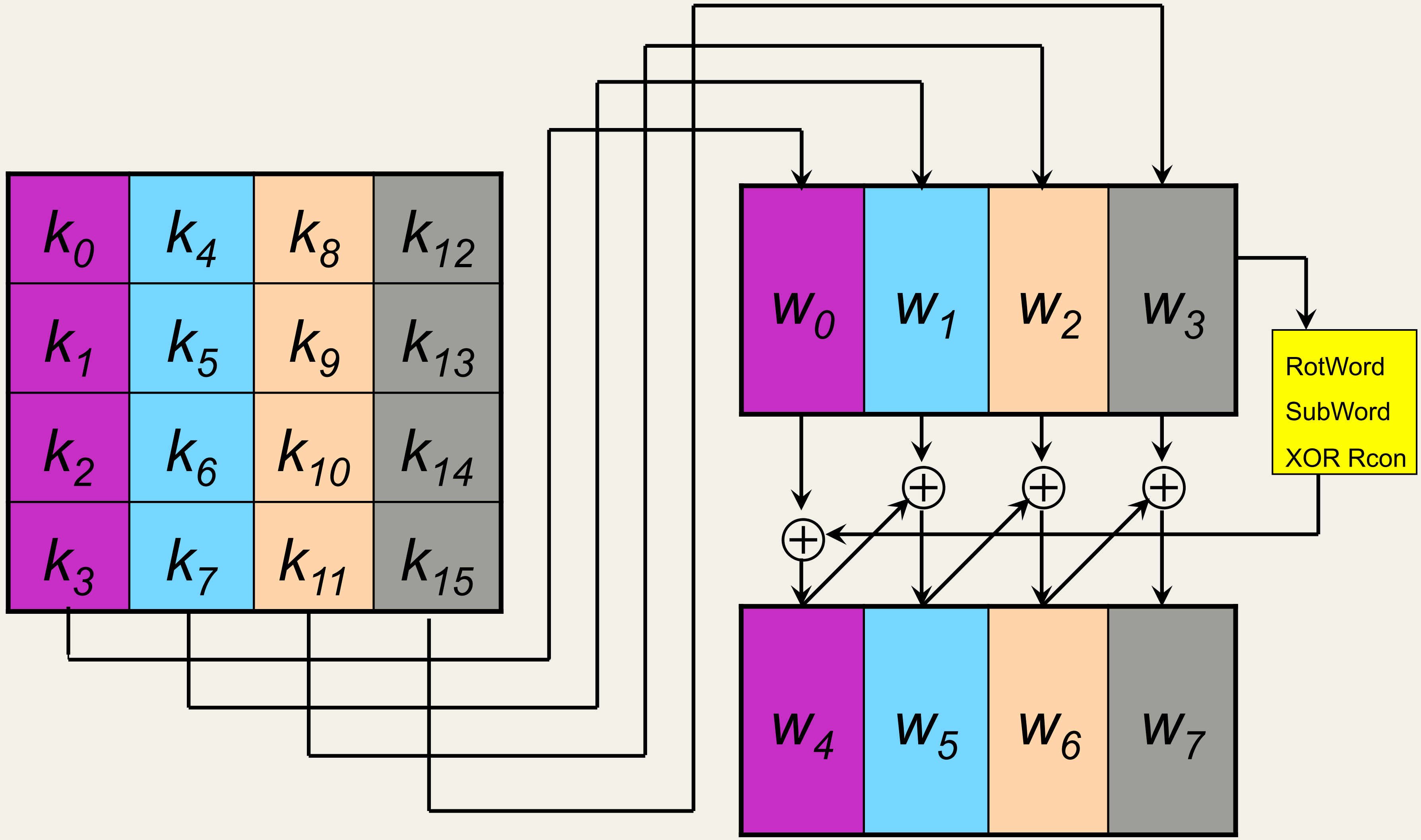
AES Key Expansion

- AES takes an 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words.
- We start by copying key into first 4 words.
- We then loop creating words that depend on values in previous & 4 places back.
  - *in first 3 of 4 cases just XOR these together*
  - *every 4th has S-box + rotate + XOR Rcon*
- It’s designed to resist known attacks

Key Schedule

- The Round Key are derived from the Cipher Key by means of the key schedule. This consists of two components: the Key Expansion and the Round Key Selection. The basic principle is the following:
- The total number of Round Key bits is: Nb*(Nr+1).
- The Cipher Key is expanded into an Expanded Key.
- Round Keys are taken from this Expanded Key in the following way: the first Round Key consists of the first Nb words, the second one of the following Nb words, and so on.

Key Expansion



RotWord, SubWord, Rcon

- RotWord performs a one-byte circular left shift.
  - *[b0,b1,b2,b3] will become [b1,b2,b3,b0]*
- SubWord performs a byte substitution using the S-box.
- The above result is XOR-ed with a round constant Rcon[j]
  - *Rcon[j]=(RC[j],0,0,0) where RC[1]=1, RC[j]=2*RC[j] (mult. over GF(2^8))*

RC[j] in HEX

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

An Example of Key Expansion

- Round key is shown on the right.
- Then the 1st 4 bytes (1st column) of the 9th round key are calculated as follows

EA	B5	31	7F
D2	8D	2B	8D
73	BA	F5	29
21	D2	60	2F

w_{31}	After RotWord	After SubWord	Rcon[9]	After XOR w/ Rcon	w_{28}	w_{32}
7F-8D-29-2F	8D-29-2F-7F	5D-A5-15-D2	1B-00-00-00	46-A5-15-D2	EA-D2-73-21	AC-77-66-F3

AES Decryption

- Different from Feistel ciphers, AES decryption is not identical to encryption.
- All steps done in reverse.

Inverse Mixcolumn Matrix

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

Inverse SubByte Affine Transformation

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

Implementation Aspects

- Efficient implementation on 8-bit CPU
  - *Byte substitution works on bytes using a table of 256 entries*
  - *Shift rows is simple byte shifting*
  - *Add round key works on byte XORs*
  - *Mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use a table lookup*

Implementation Aspects

- Efficient implementation on 32-bit CPU
  - *We can redefine steps to use 32-bit words*
  - *We can pre-compute 4 tables of 256-words*
  - *Each column in each round can be computed using 4 table lookups + 4 XORs*
  - *Tradeoff: 16Kb to store tables*
- Designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Summary

- We have considered:
  - *The AES selection process*
  - *The details of Rijndael – the AES cipher*
  - *All steps in each round*
  - *The key expansion*
  - *Implementation aspects*