Cryptography and Data Security

Lecture No. 13

2-10-2019

FALL 2019

FAST National University of Computer and Emerging Sciences, Islamabad

Origins

- 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.
- AES cipher (& other candidates) form the latest generation of block ciphers, and now we see a significant increase in block size - from old standard of 64-bits up to 128-bits; and keys from 128 to 256-bits.
- While triple-DES is secure and well understood, it is slow, especially in s/w.
- 15 candidates accepted in Jun 98. 5 were shortlisted in Aug-99
- NIST published a final standard (FIPS PUB 197) in Nov. 2001.
- NIST selected Rijndael as the proposed AES algorithm.
- The two researchers from Belgium were: Dr. Joan Daemen and Dr.Vincent Rijmen.

AES Requirements

- 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)
- Provides full specification & design details
- Both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- Initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics
- Final criteria
  - 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
- Then subject to further analysis & comment
- Saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

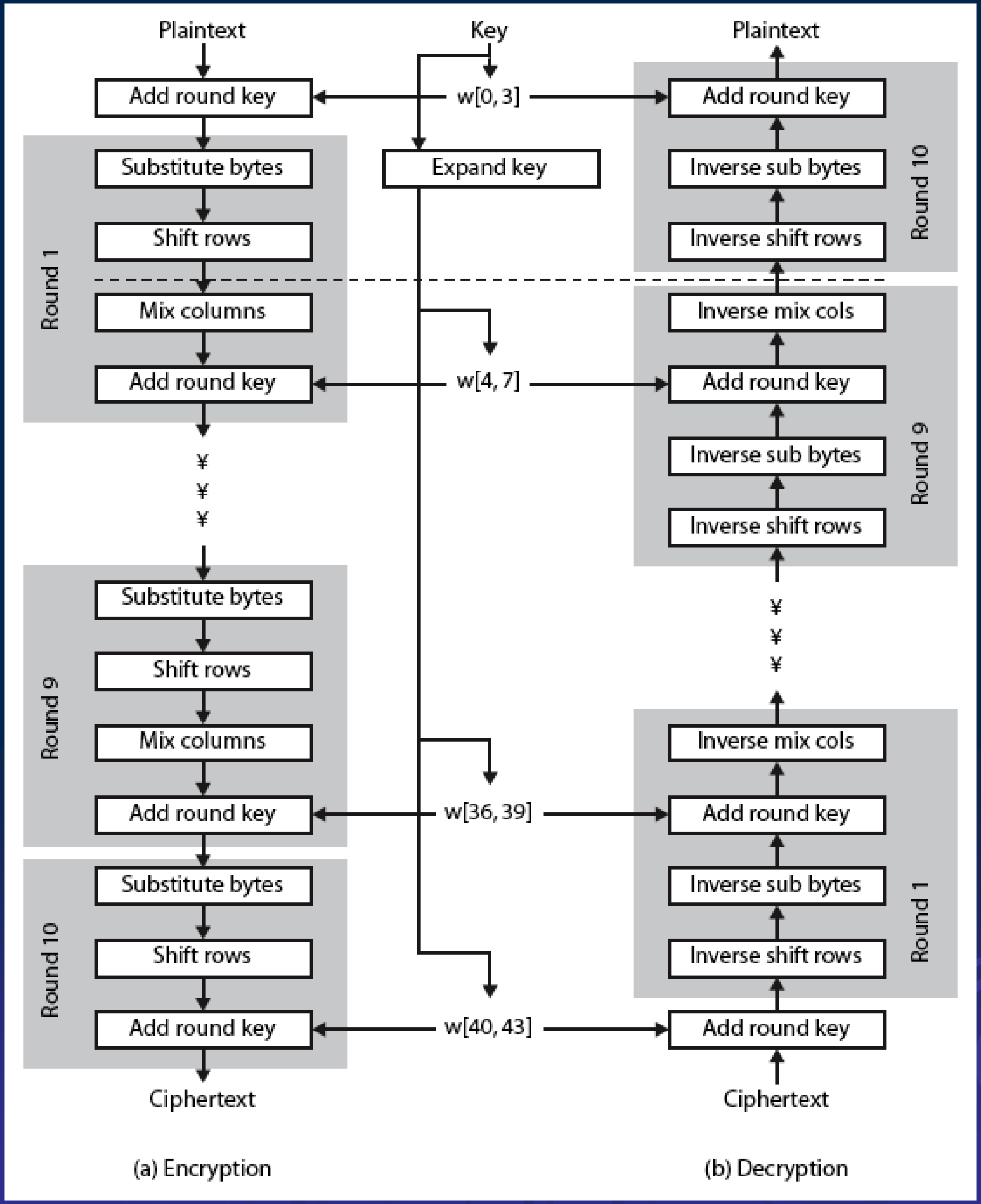
The AES Cipher - Rijndael

- Has 128/192/256 bit keys, 128 bit data
- An **iterative** rather than **feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round rather than feistel (operates on halves at a time),
- Designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Rijndael

- Data block of 4 columns of 4 bytes
- This block is copied into State array, which is modified at each stage of encryption or decryption.
- After the final stage, State is copied to an output.
- Key is expanded to array of words
- Has 10/12/14 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (substitution using matrix multiply of groups)
  - add round key (XOR state with key material)
  - view as alternating XOR key & scramble data bytes
- Initial XOR key material & incomplete last round
- With fast XOR & table lookup implementation

Rijndael



Plaintext
Key
Plaintext
Add round key
w[0, 3]
Add round key
Substitute bytes
Expand key
Inverse sub bytes
Shift rows
Inverse shift rows
Round 10
Round 1
Mix columns
Inverse mix cols
Add round key
w[4, 7]
Add round key
Inverse sub bytes
Round 9
Inverse shift rows
¥
¥
¥
Substitute bytes
Shift rows
Round 9
Mix columns
Inverse mix cols
Add round key
w[36, 39]
Add round key
Substitute bytes
Inverse sub bytes
Round 10
Shift rows
Inverse shift rows
Round 1
Add round key
w[40, 43]
Add round key
Ciphertext
Ciphertext
(a) Encryption
(b) Decryption

Byte Substitution

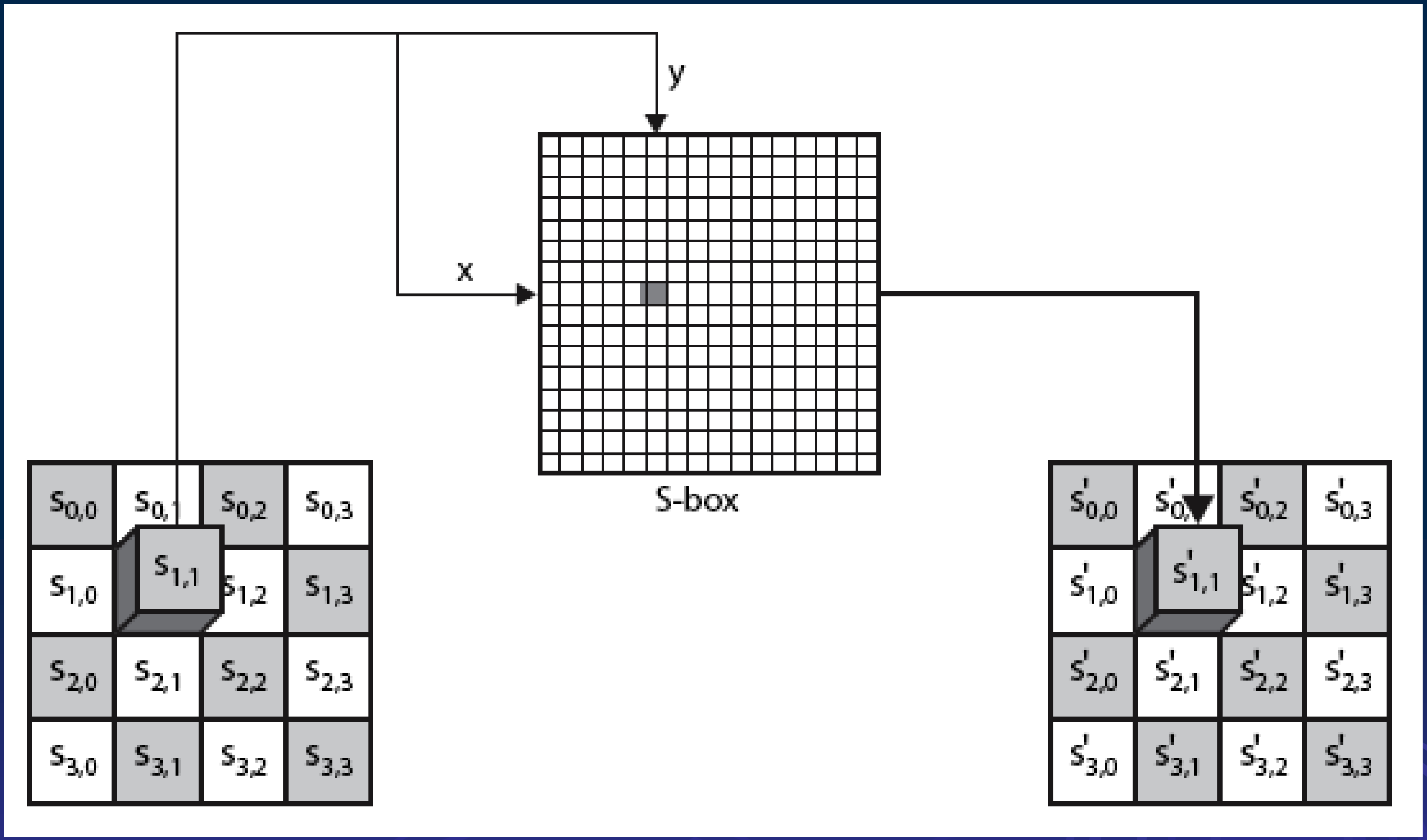
- A simple substitution of each byte
- Uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- Each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by byte in row 9 column 5
  - which has value {2A}
- S-box constructed using defined transformation of values in GF(2^8)
- Designed to be resistant to all known attacks

Byte Substitution

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

S-Box

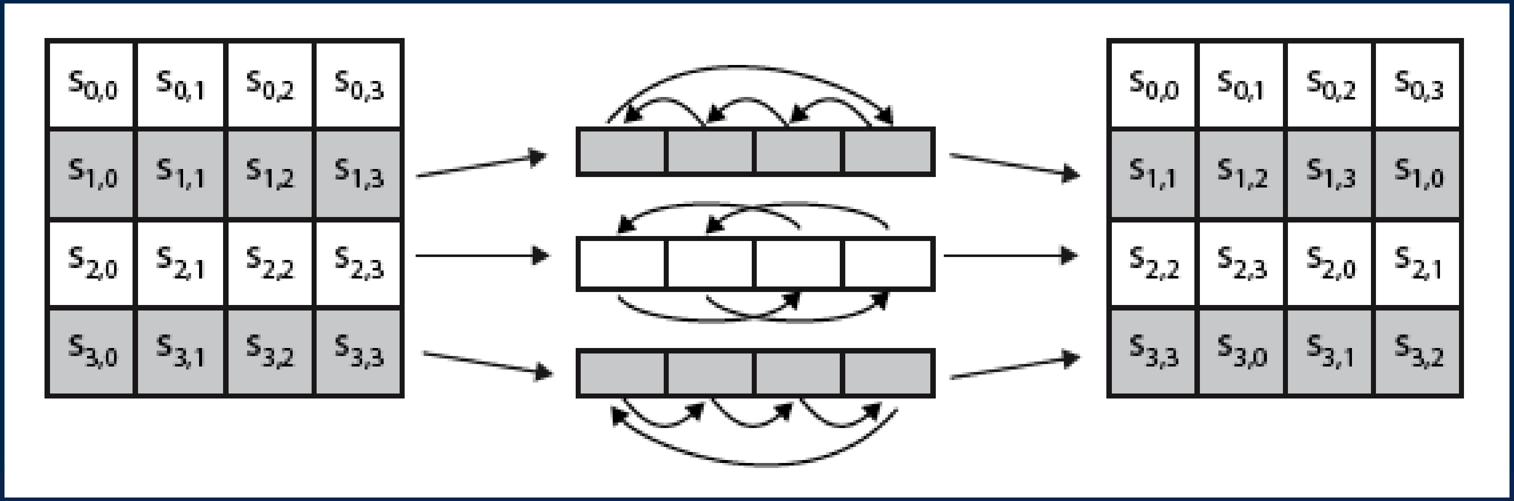
Byte Substitution



Shift Rows

- A circular byte shift in each row.
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- Decrypt inverts using shifts to right
- Since state is processed by columns, this step permutes bytes between the columns

Shift Rows



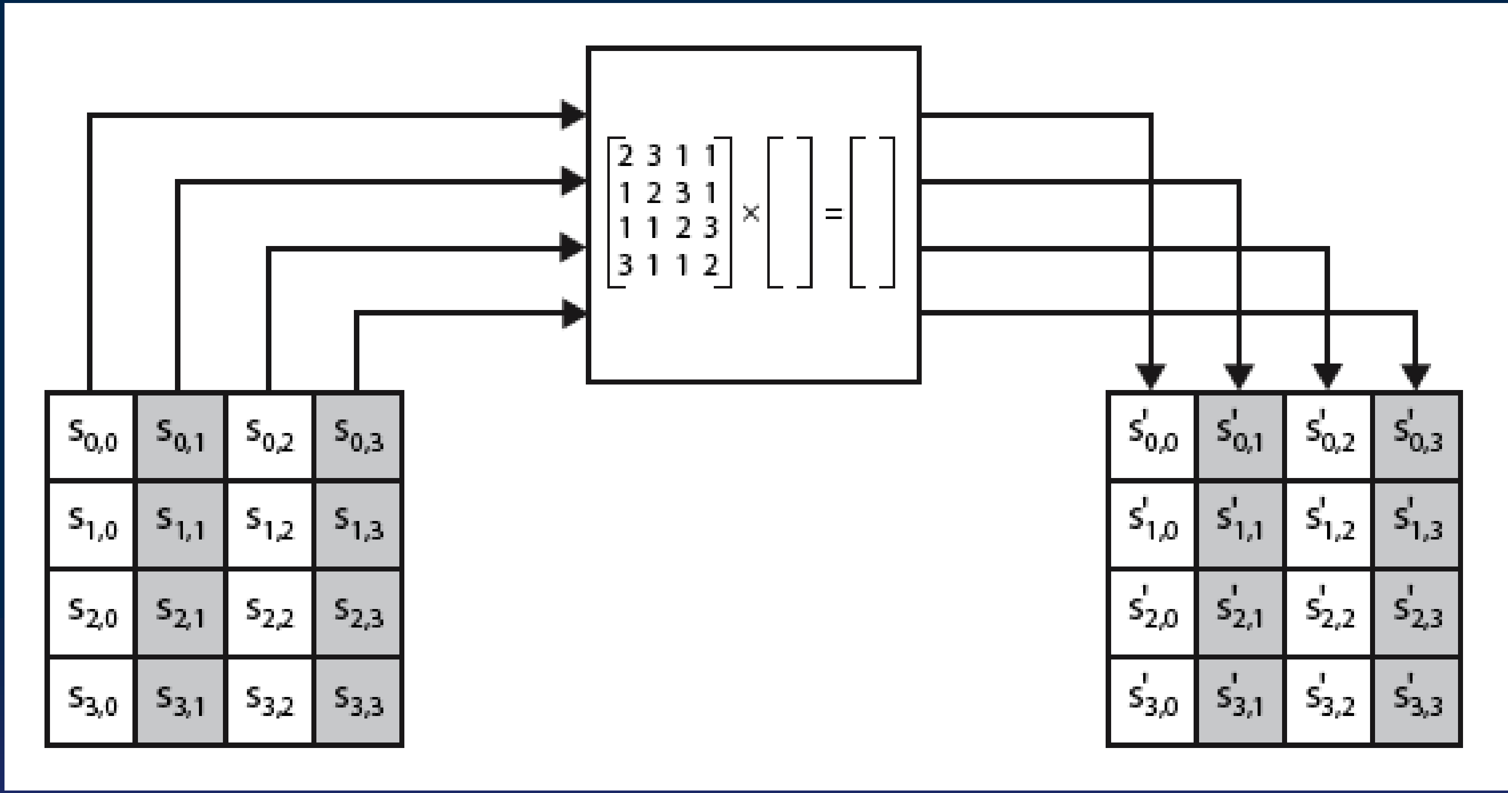
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$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

Mix Columns

- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Effectively a matrix multiplication in $GF(2^8)$ using prime poly $m(x) = x^8+x^4+x^3+x+1$

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

Mix Columns



2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

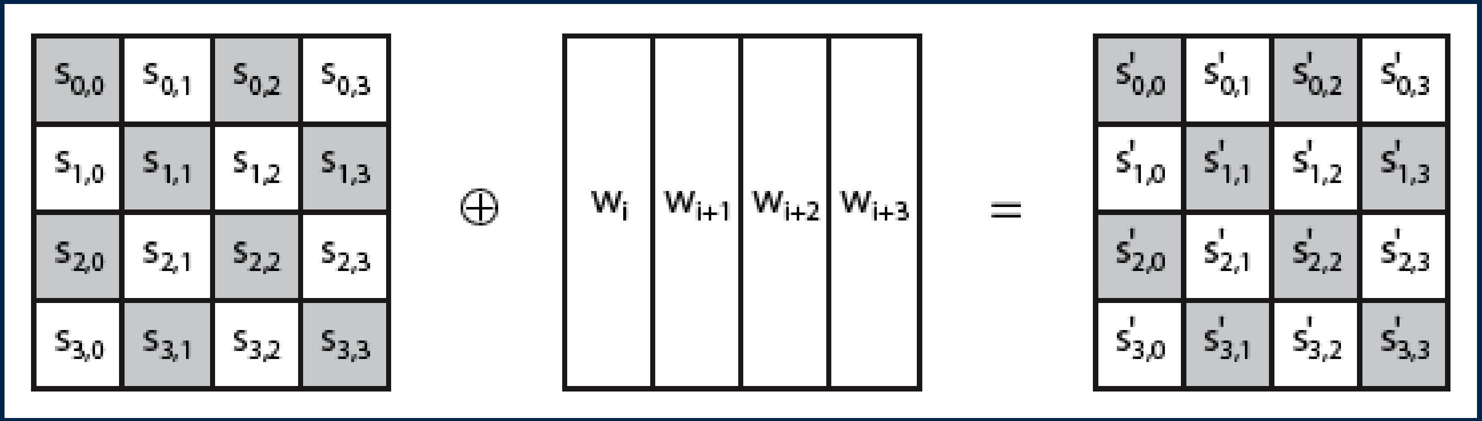
Mix Columns

- Can express each col as 4 equations
  - to derive each new byte in col
- Decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder
- Have an alternate characterisation
  - each column a 4-term polynomial
  - with coefficients in $GF(2^8)$
  - and polynomials multiplied modulo (x^4+1)

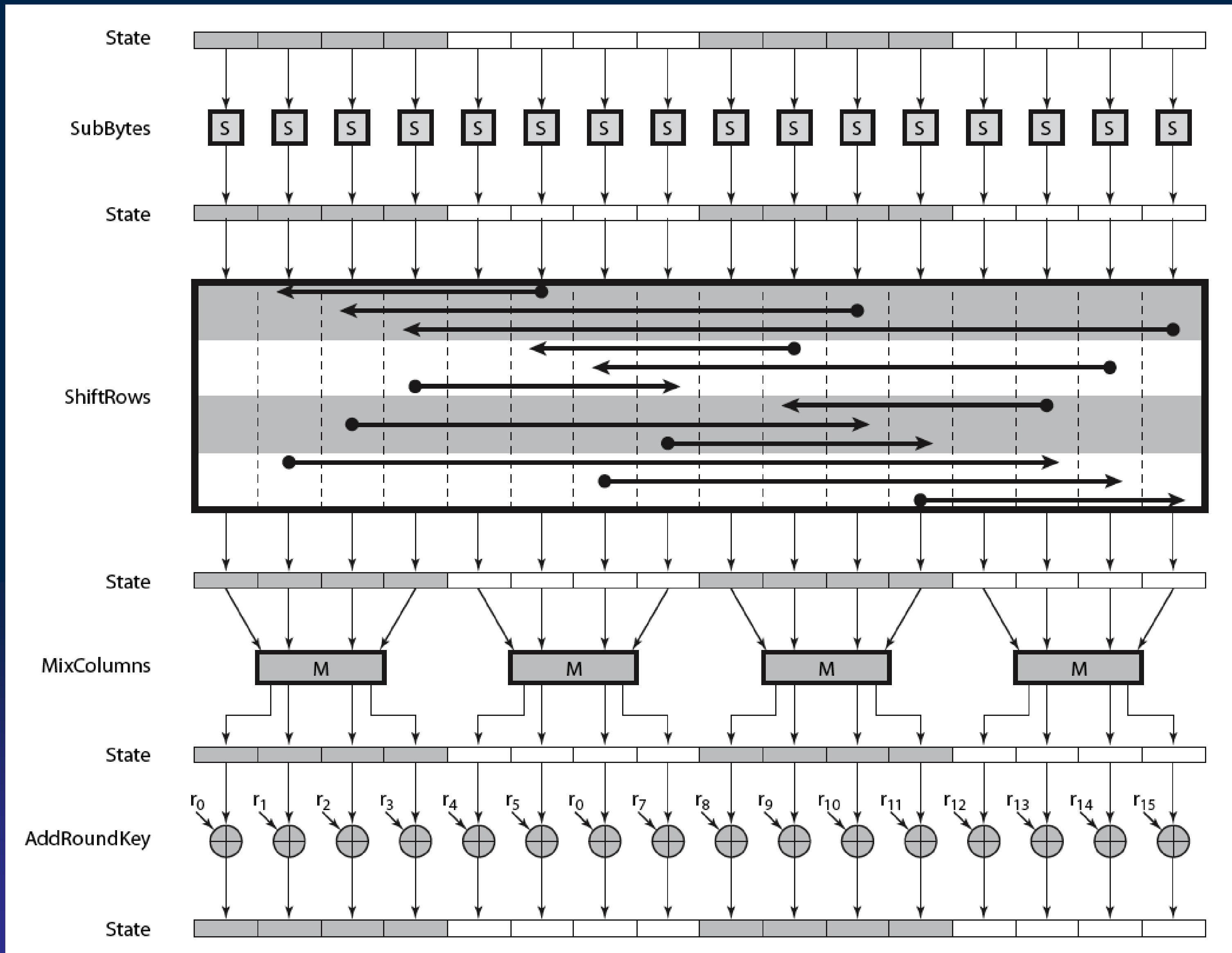
Add Round Key

- XOR state with 128-bits of the round key
- Again processed by column (though effectively a series of byte operations)
- Inverse for decryption identical
  - since XOR own inverse, with reversed keys
- Designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security

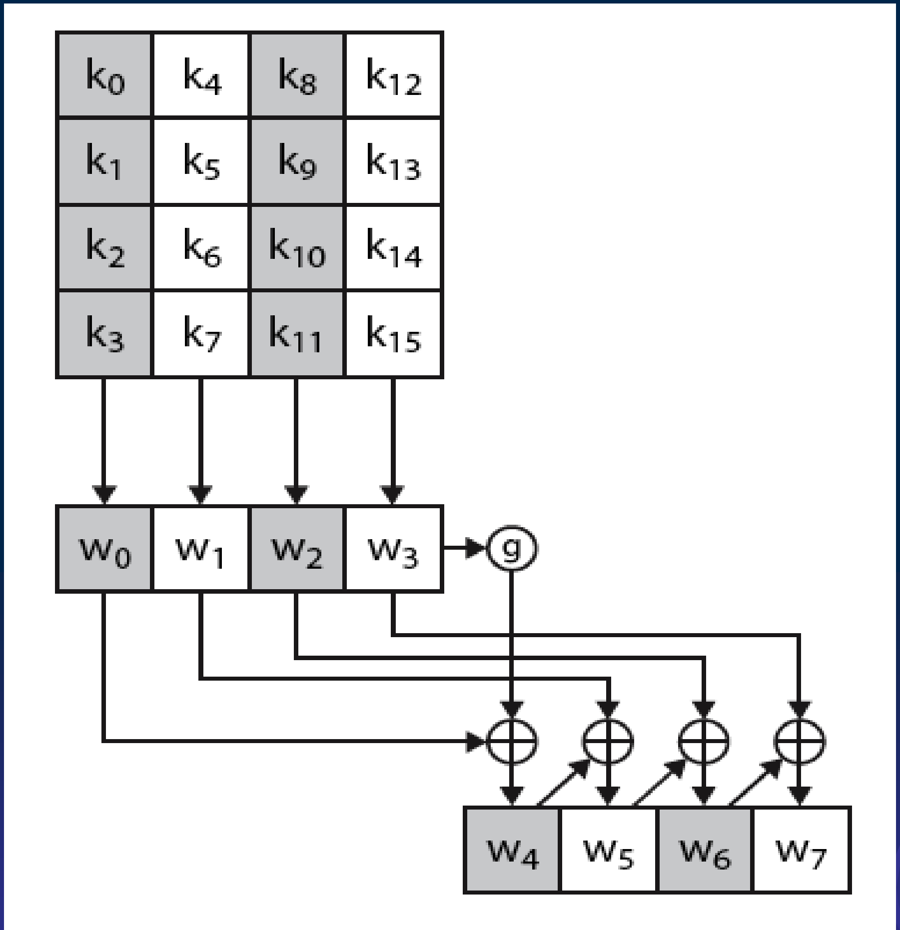
Add Round Key



AES Round


State
SubBytes
S
State
ShiftRows
State
MixColumns
M
State
AddRoundKey
r0 r1 r2 r3 r4 r5 r0 r7 r8 r9 r10 r11 r12 r13 r14 r15
State

AES Key Expansion



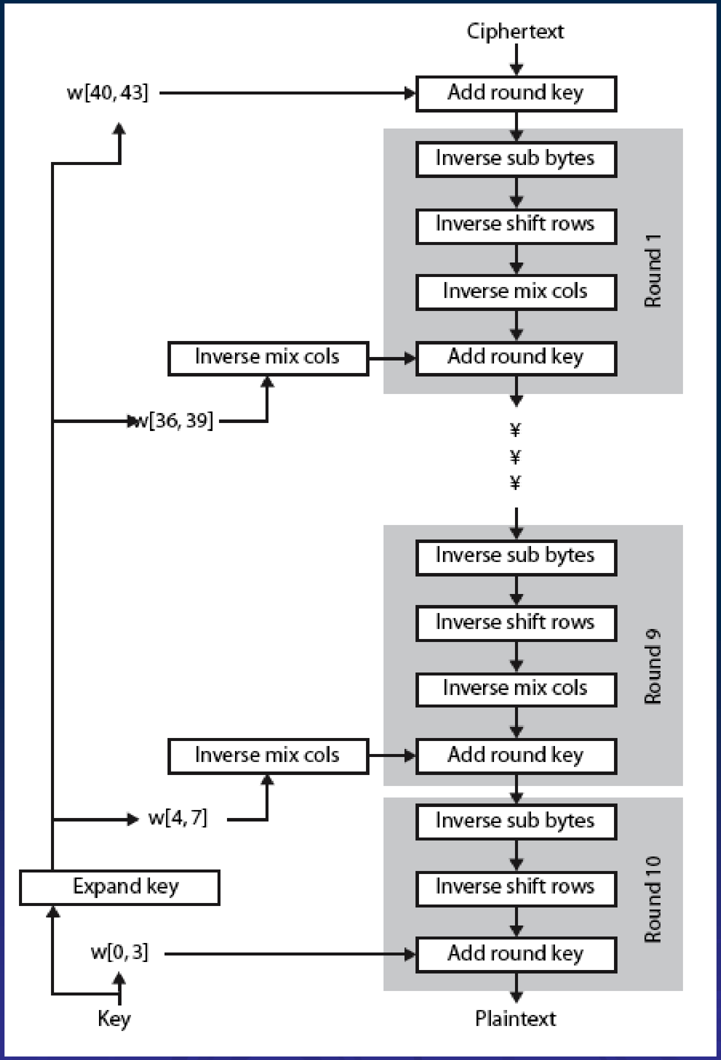
AES Decryption

- AES decryption cipher is not identical to the encryption cipher.
- The sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is same.
- Disadvantage - two separate software or firmware modules are needed for applications that require both encryption and decryption.
- An equivalent version of decryption algorithm exists that has same structure as encryption, with same sequence of transformations as encryption algorithm (with transformations replaced by their inverses).
- To achieve this equivalence, a change in key schedule is needed.

AES Decryption

- By constructing an equivalent inverse cipher with steps in same order as for encryption, we can derive a more efficient implementation.
- Swapping byte substitutions and shift rows has no effect, since both work just on bytes.
- Swapping mix columns and add round key steps requires inverse mix columns step be applied to the round keys first – this makes decryption key schedule a little more complex with this construction, but allows use of same h/w or s/w for the data en/decrypt computation.

AES Decryption



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

Implementation Aspects

- 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

Applications of AES

- Internet banking
- FTPS, HTTPS, SFTP, AS2, WebDAV's etc

QUESTIONS