AES :Advanced Encryption Standard

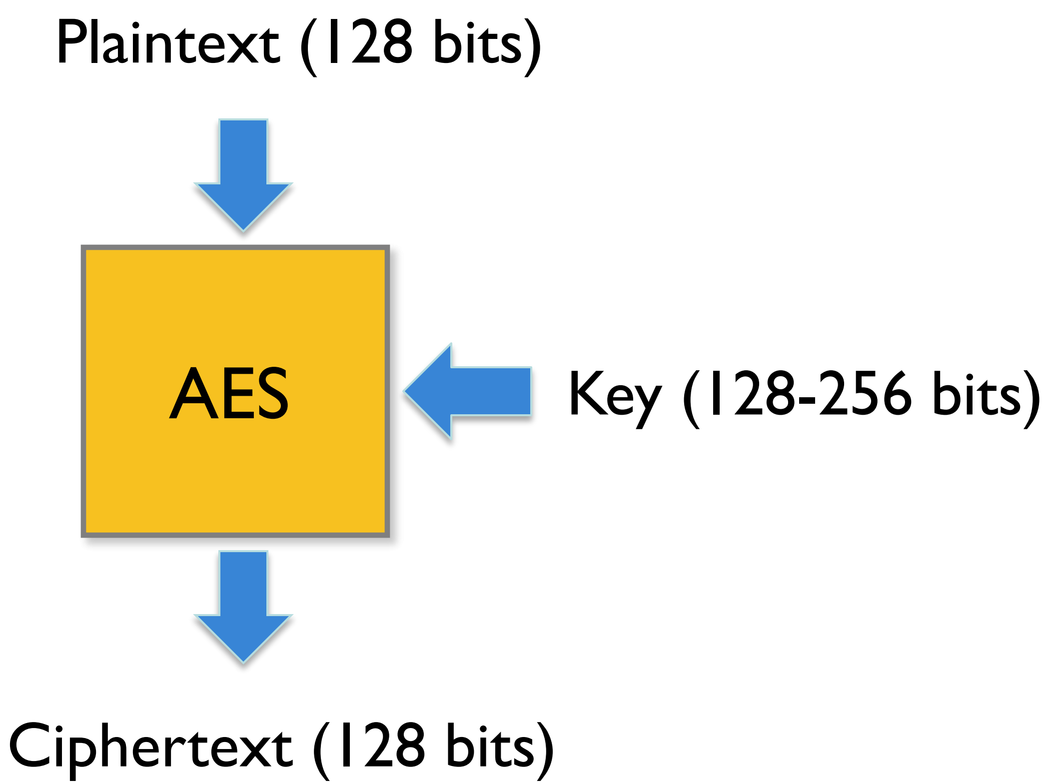
Origins

- clear a replacement for DES was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
- can use Triple-DES – but slow with small blocks
- US NIST issued call for ciphers in 1997
- 5 were short-listed 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
- Rijndael was selected as the AES in Oct-2000
- 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 Triple-DES
- active life of 20-30 years (+ archival use)
- provide full specification & design details
- both C & Java implementations
- NIST have released all submissions & unclassified analyses
- Evaluation criteria of submitted ones
  - General security – effort to practically cryptanalyse
  - algorithm & implementation characteristics
  - cost – computational, software & hardware implementation ease, minimize implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

AES Conceptual Scheme

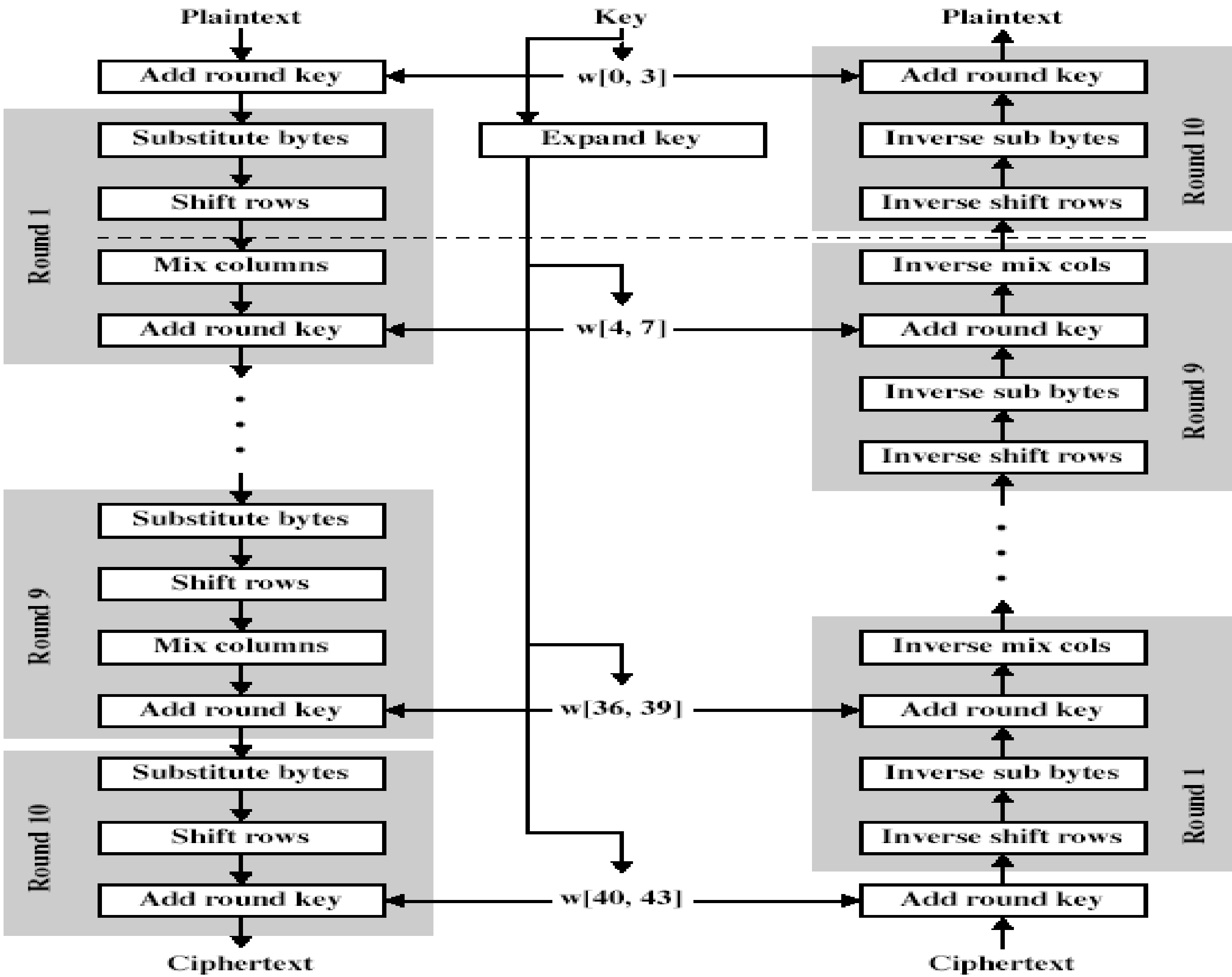


Rijndael

- processes data as 4 groups of 4 bytes (state)
- has 9/11/13 rounds in which state undergoes:
  1. byte substitution (1 S-box; byte to byte substitution)
  2. shift rows (permutation of bytes)
  3. mix columns (subs using $gf2^8$)
  4. Add Round Key (XOR state with a portion of expended K)
- initial XOR key material & incomplete last round
- all operations can be combined into XOR and table lookups - hence very fast & efficient

The AES Cipher

- designed by Rijmen-Daemen in Belgium
- has 128/192/256 bit keys, 128 bit data
- an **iterative** rather than **feistel** cipher
  - treats data in 4 groups of 4 bytes
  - operates an entire block in every round
- designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

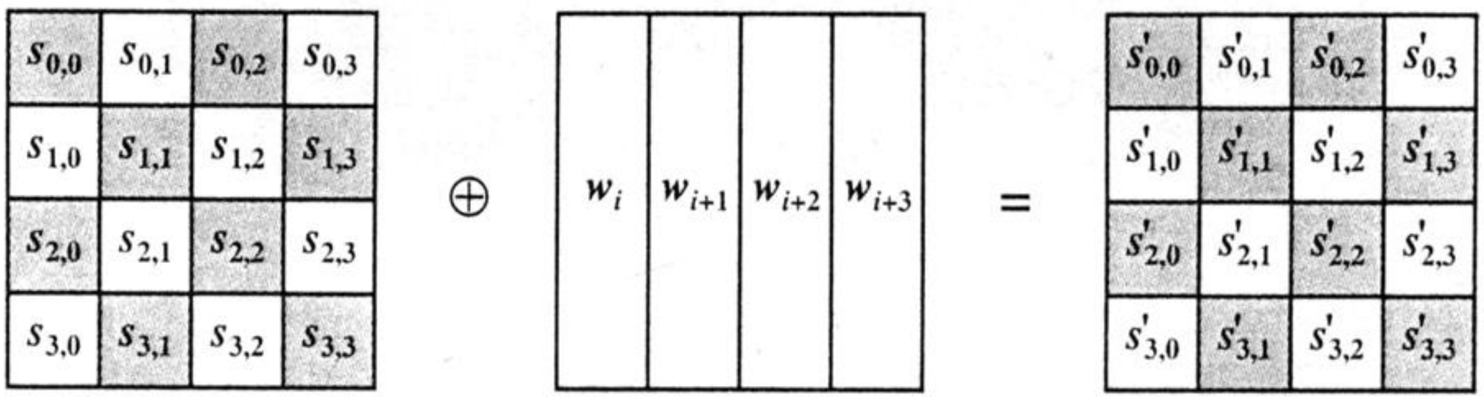


(a) Encryption

(b) Decryption

AddRoundKey

- Each round uses four different words from the expanded key array.
- Each column in the state matrix is XORed with a different word.
- The heart of the encryption. All other functions' properties are permanent and known to all.



(b) Add Round Key Transformation

InvAddRoundKey

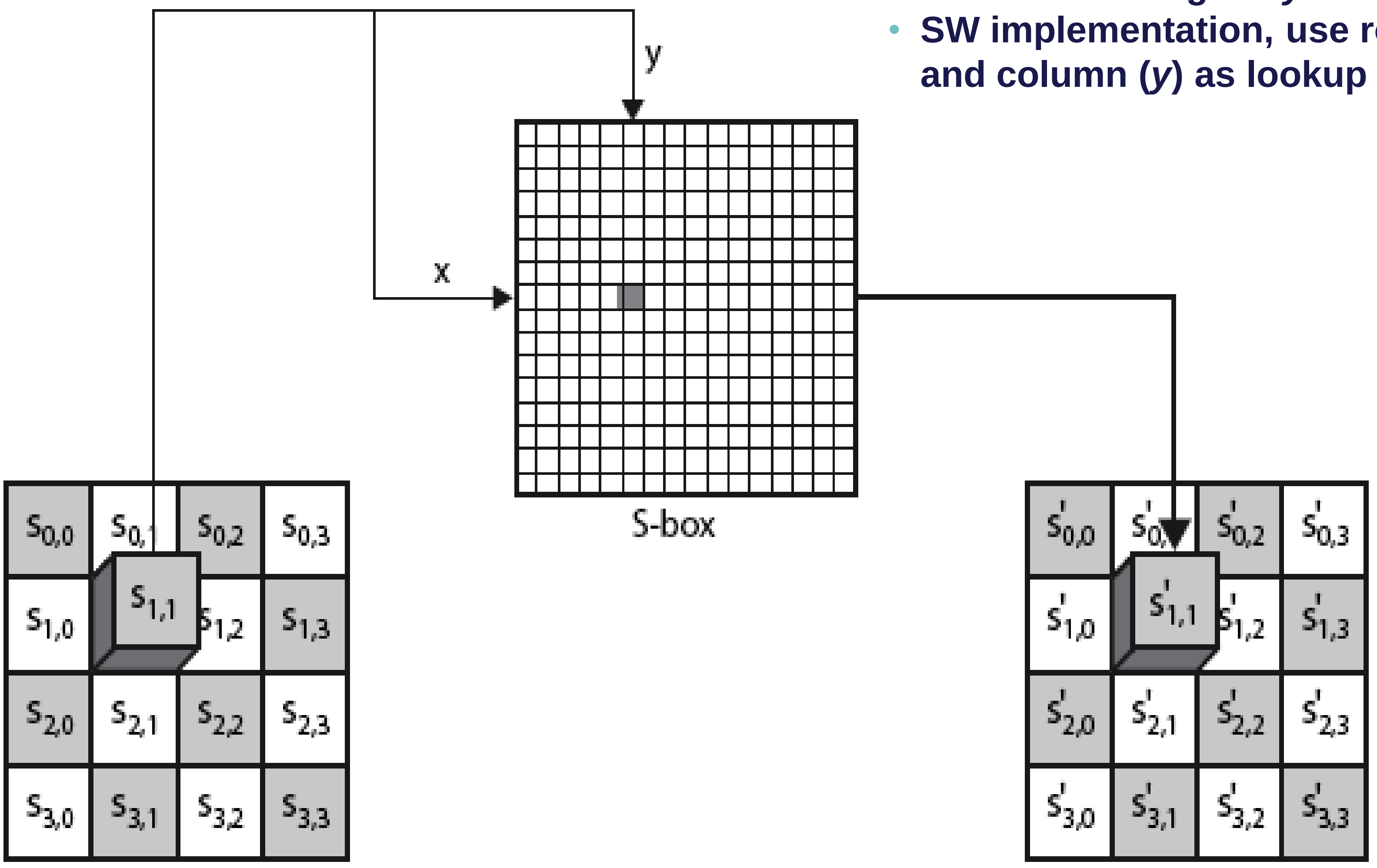
- $(A \oplus B) \oplus B = A$
- Key is used in reverse order

Substitution Byte (Subbyte)

- It is a bytewise lookup process that returns a 4-byte word in which each byte is the result of applying the Rijndael S-box. Designed to be resistant to all known attacks
- Simple substitution of each byte using one table of 16x16 bytes containing a permutation of all 256 8-bit values
- each byte of state is replaced by byte in row (left 4-bits) & column (right 4-bits)
  - eg. byte {95} is replaced by row 9 col 5 byte
  - which is the value {2A}
- S-box is constructed using a transformation of the values in $GF(2^8)$

Substitution Byte

- **Interpret the byte as two hexadecimal digits *xy***
- **SW implementation, use row (*x*) and column (*y*) as lookup pointer**



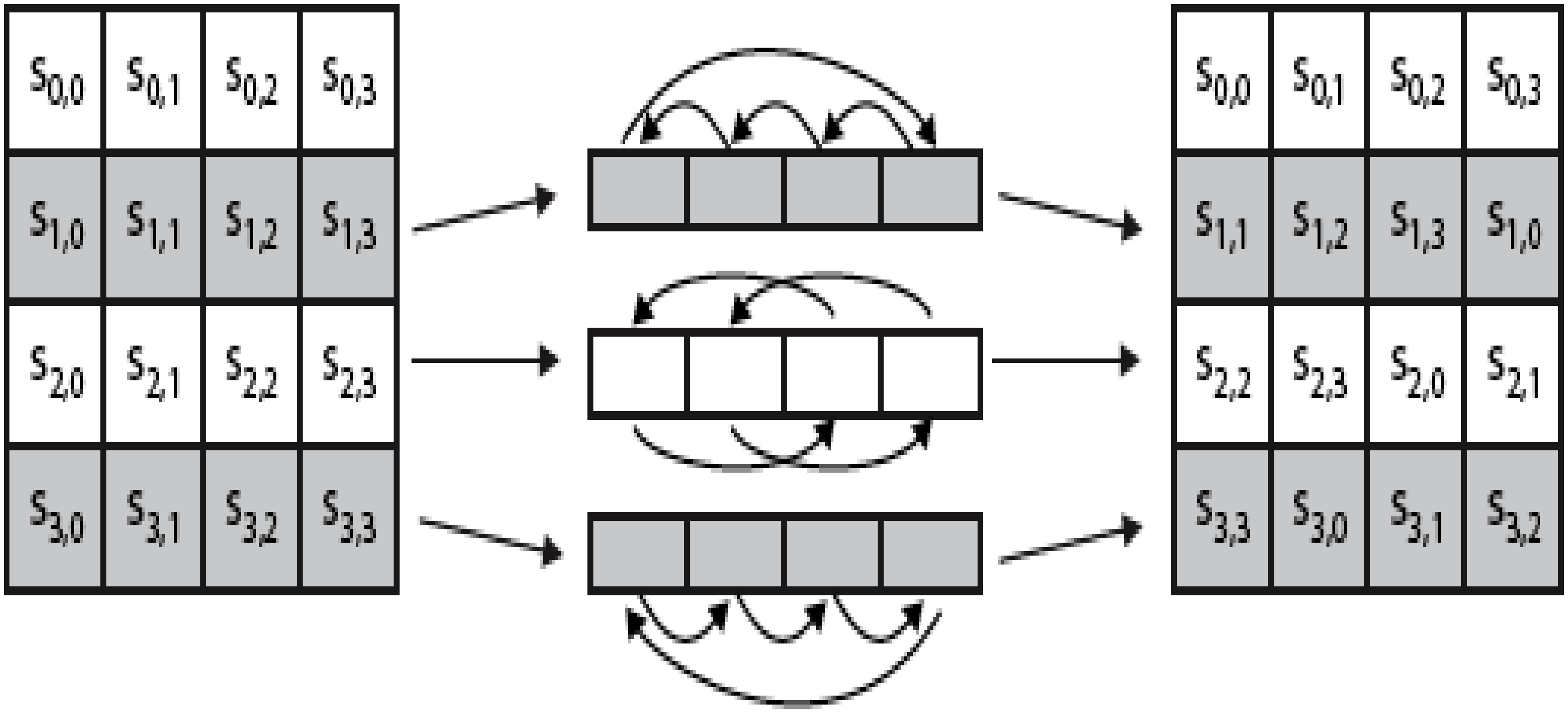
S-Box eg. byte {95} is replaced by row 9 col 5 byte which is the value {2A}

		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
x	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

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 does 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

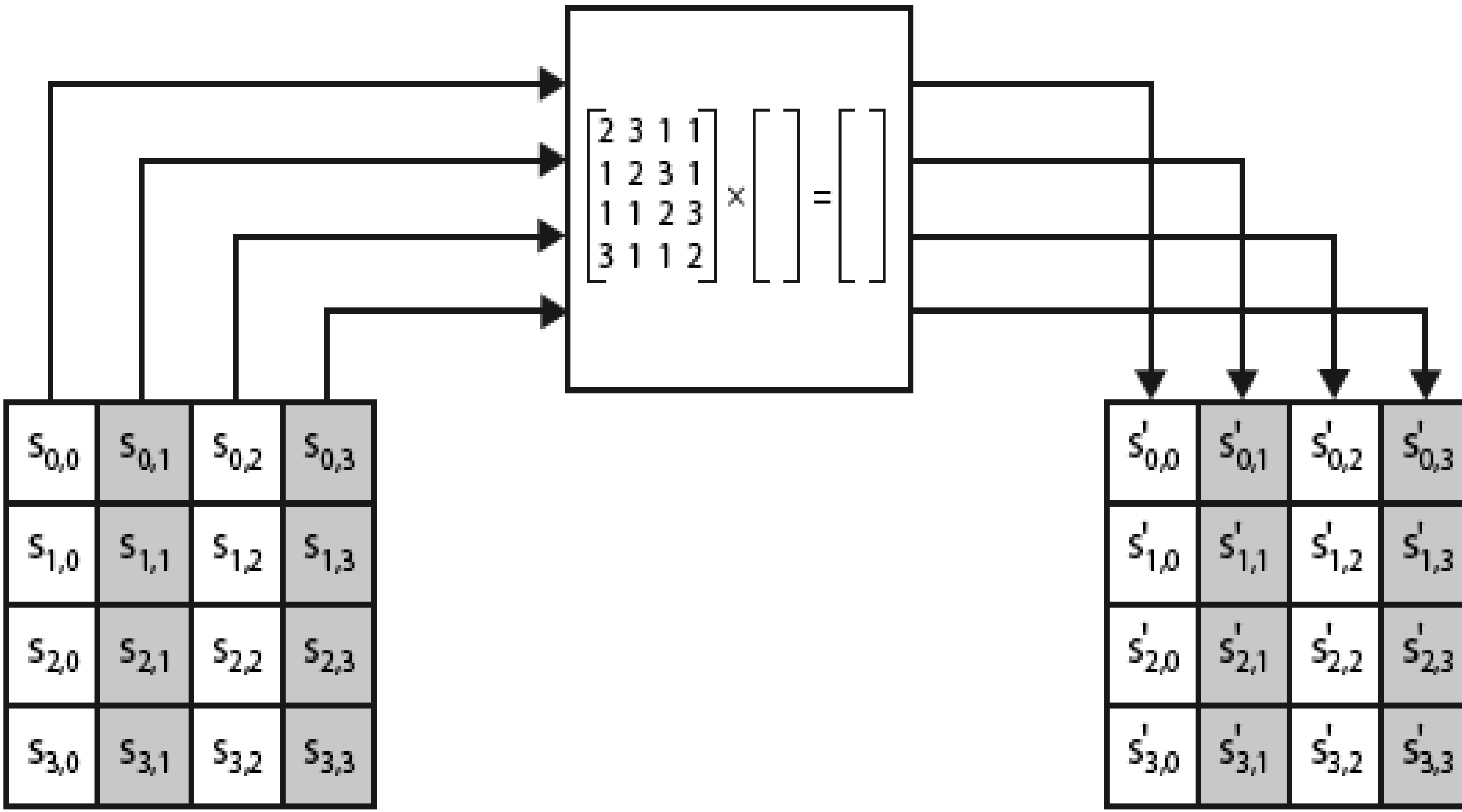
- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in $GF(2^8)$ using prime poly $m(x) = x^8+x^4+x^3+x+1$

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

Add Round Key

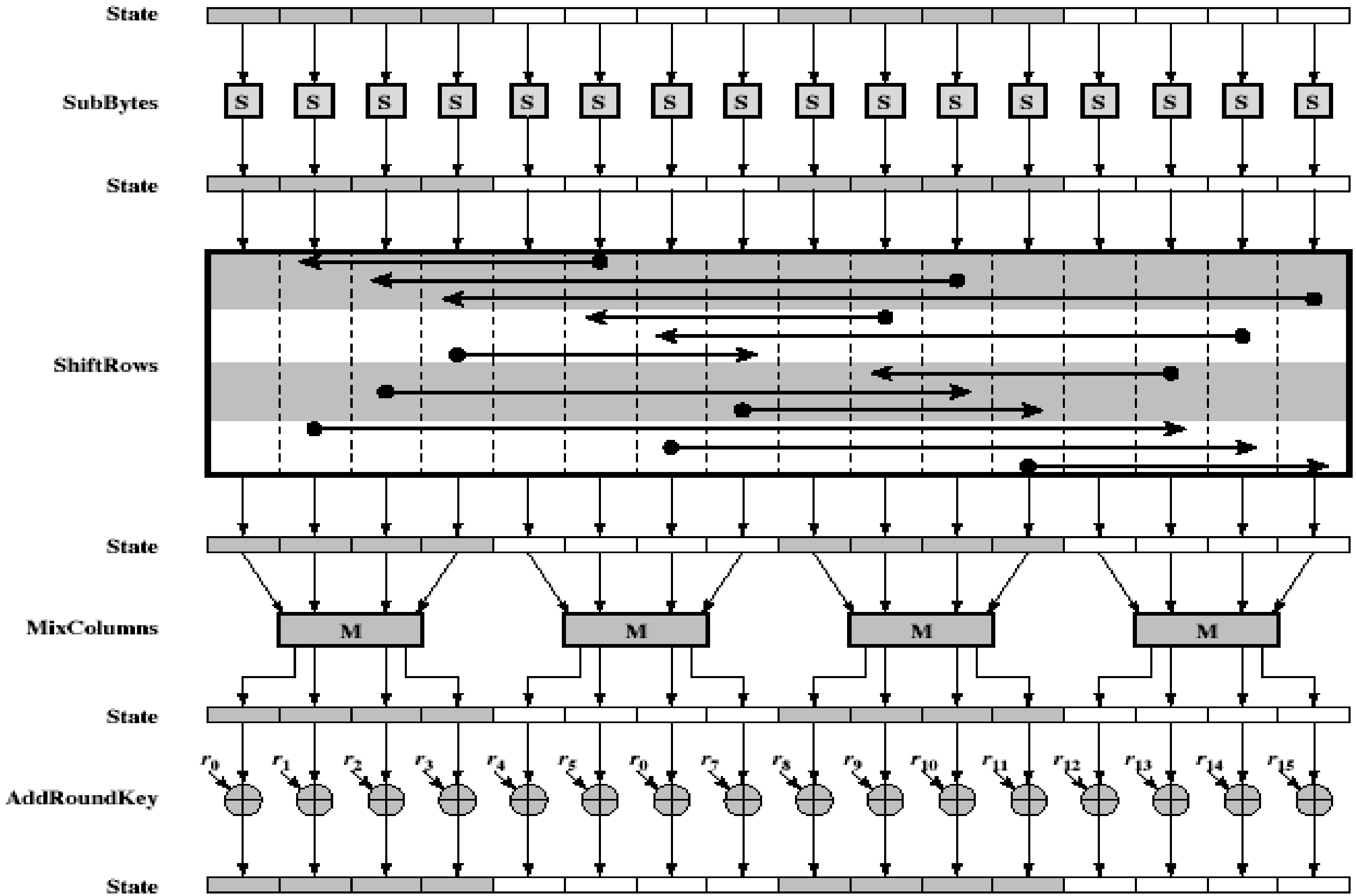
- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption is identical since XOR is own inverse, just with correct round key
- designed to be simple

Mix Columns



2 3 1 1
1 2 3 1
1 1 2 3
3 1 1 2
×
=
s0,0 s0,1 s0,2 s0,3
s1,0 s1,1 s1,2 s1,3
s2,0 s2,1 s2,2 s2,3
s3,0 s3,1 s3,2 s3,3
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

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

AES Key Expansion

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

AES Key Expansion

KeyExpansion ([key_0 to key_{15}], [$\mathbf{w}_0$ **to** $\mathbf{w}_{43}$])

{

  for (i = 0 to 3)

    $\mathbf{w}_i \leftarrow \text{key}_{4i} + \text{key}_{4i+1} + \text{key}_{4i+2} + \text{key}_{4i+3}$

  for (i = 4 to 43)

  {

    if ($i \bmod 4 \neq 0$) $\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbf{w}_{i-4}$

    else

    {

      $\mathbf{t} \leftarrow \text{SubWord}(\text{RotWord}(\mathbf{w}_{i-1})) \oplus \text{RCon}_{i/4}$ *// **t** is a temporary word*

      $\mathbf{w}_i \leftarrow \mathbf{t} + \mathbf{w}_{i-4}$

    }

  }

}

AES Decryption

- AES decryption is not identical to encryption since steps done in reverse
- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

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 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
- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can pre-compute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 16Kb to store tables
- designers believe this very efficient implementation was a key factor in its selection as the AES cipher

Summary

- have considered:
  - the AES selection process
  - the details of Rijndael – the AES cipher
  - looked at the steps in each round
  - the key expansion
  - implementation aspects

  SEE ANIMATION OF AES @

  http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf