5.1 Advanced Encryption Standard (AES)

AES Main points

- Key length: 128^{256}
- Average bit key sizes: 128, 192, 256
- Block size: 128 bits (16 bytes)

An iterative cipher (not like Feistel)

- Processes data as a block of 4 columns of 4 bytes
- Operates on **entire data block** (not half and half) in every round

All operations are performed on 8-bit bytes

- The arithmetic operations are performed over the finite field GF(2^8), with irreducible polynomial m(x) = x^8 + x^4 + x^3 + x + 1 (first of the 30 degree-8 reducible polynomials)

Designed to be:

- Resistant against known attacks
- Speed and code compactness on many CPUs
- Design simplicity

Additional notes of AES features:

- Four different stages are used, one permutation and three of substitution:
  1. Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
  2. Shift rows: Simple permutation
  3. Mix columns: A substitution that makes use of arithmetic over GF(2^8)
  4. Add round key: A simple bitwise XOR of the current block with a portion of the expanded key.
- Both encryption and decryption begin with an *AddRoundKey* followed by nine rounds. Each round includes the four different stages as mentioned above. The tenth round has 3 stages (No mix column).
- Only the *AddRoundKey* stage makes use of the key (This is why the cipher begins and ends with this stage)
- The *AddRoundKey* is like a Vernam Cipher, pretty shit. However with the other three rounds they provide confusion, diffusion, and nonlinearity.
- The last round for both encryption and decryption ends with 3 stages.
- Decryption is done by going through all the rounds with the following order:
  - Inverse shift row
  - Inverse sub bytes
  - Add round key
  - Inverse mix columns
- To achieve decrypting the *AddRoundKey* all you have to do is XOR it again.
  - (A $\oplus$ B) is the *AddRoundKey* for encryption
  - (A $\oplus$ B $\oplus$ B = A) this is the *AddRoundKey* for decryption (gives you the original value)

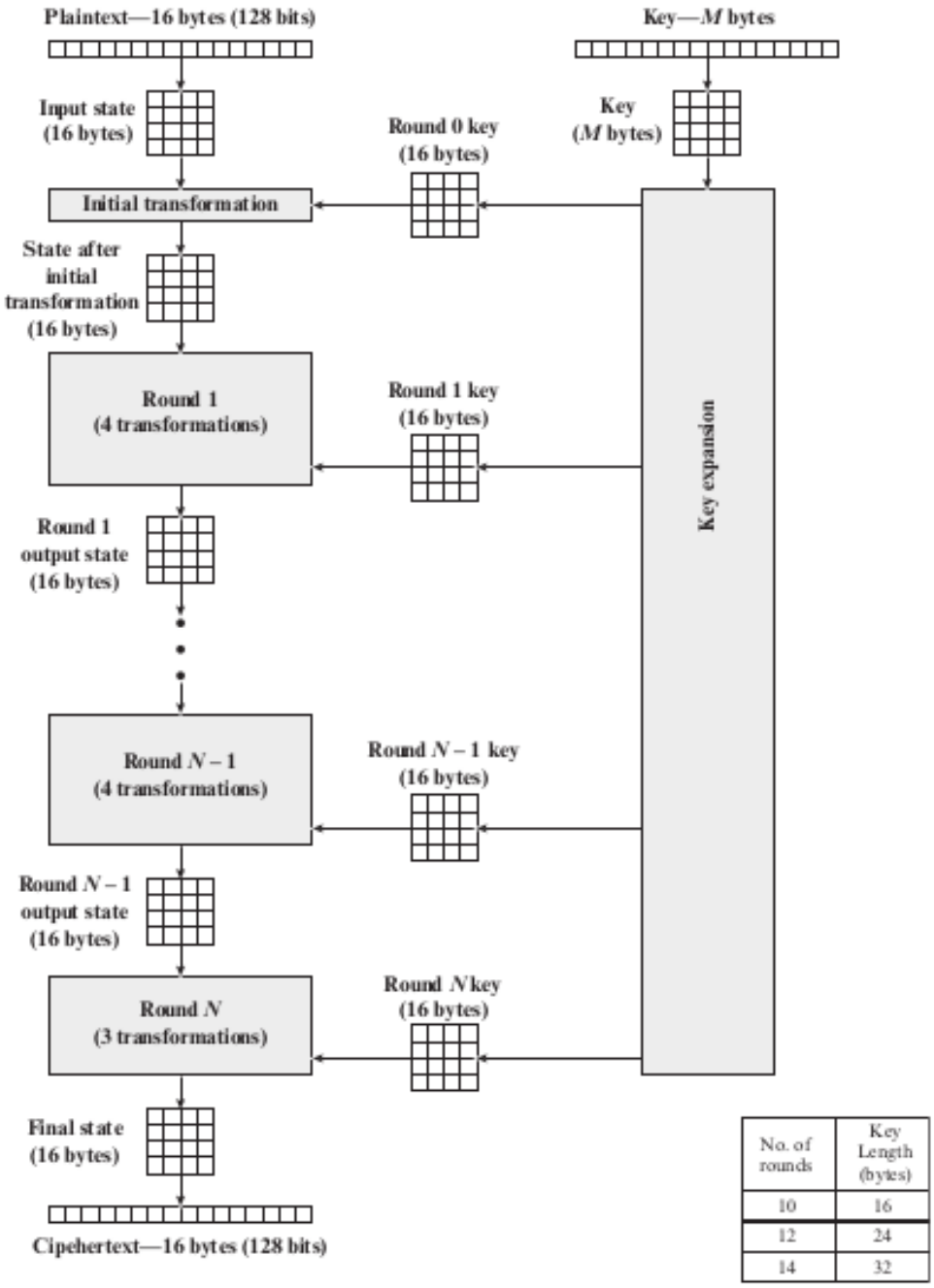

Figure 1: Diagram of full AES encryption process

- 128 Bit key length: expanded into 44 32-bit words
- Number of rounds relates to key length
- The byte ordering is by column for both the data and key

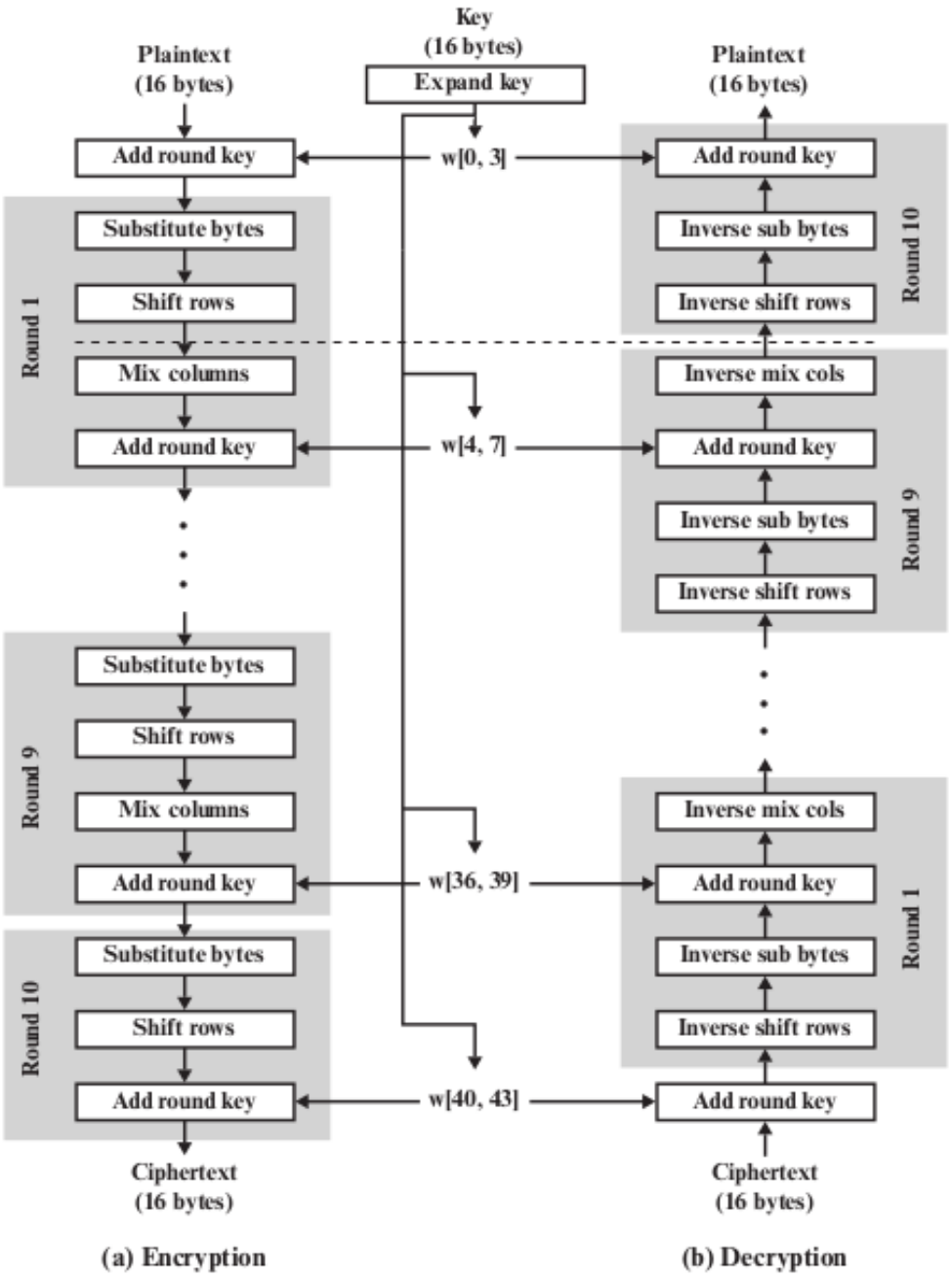

Figure 2: Close up diagram of the rounds for AES encryption/decryption

- One S-box is used on every byte-by-byte substitution
- Encryption and decryption process are not identical
- Key is expanded into array of 32-bit words; four words form a round key in each round

Substitute Bytes

Substitute bytes: use a S-box table and substitute each byte using the rows and columns of the S-Box table.

- S-box is 16x16
- Each byte of a state is replaced by byte indexed by row (left 4 bits) and column (right 4 bits)

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	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

Figure 3: S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	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

Figure 4: Inverse S-box

Example:

Given a byte is 95, the resulting substitution using S-box would result in 2A (row 9 column 5)

Constructing S-Boxes

Consider the byte input $\in$ GF(2^8) and computes its inverse.

- A = byte input 11000010 as polynomial
- B' = inverse of A
- A * B' = 1 mod (irreducible polynomial for AES)
- With B' multiply it with the affine mapping to produce the S-box

[**LINK**]: Video on constructing sbox

Shift Rows

Circular byte shifts for each row:

- 1st row: unchanged
- 2nd row: 1 byte circular shift to left
- 3rd row: 2 byte circular shift to left
- 4th row: 3 byte circular shift to left

Decryption uses right shifts

Since state is processed by columns, this step permutes bytes between columns

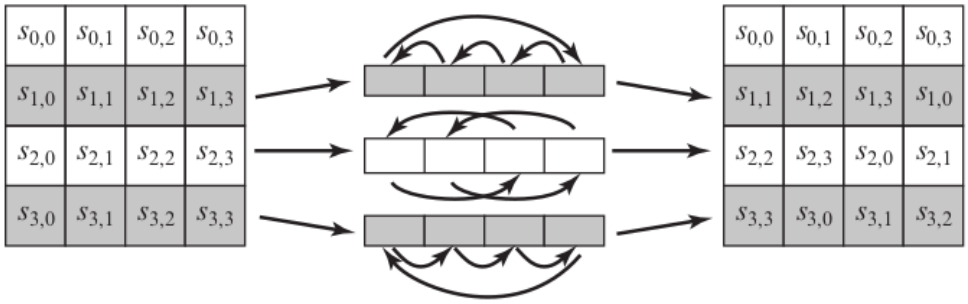

Figure 5: Example of Shift Rows

Mix Columns

Mix columns:

- Do the 4 byte columns one at a time (columns processed independently)
- Multiply it by the **special constant matrix**
- How to multiply???
  - Each byte is 8 bits
  - The output is a vector of size 4
  - Example of output_0 = 02 * Col_{11} + 03 * Col_{12} + 01 * Col_{13} + 01 * Col_{14}
  - For each of the (constant matrix) * (column) these are both 8 bits
  - Need to do polynomial multiplication and addition in the Galois field for all these bytes

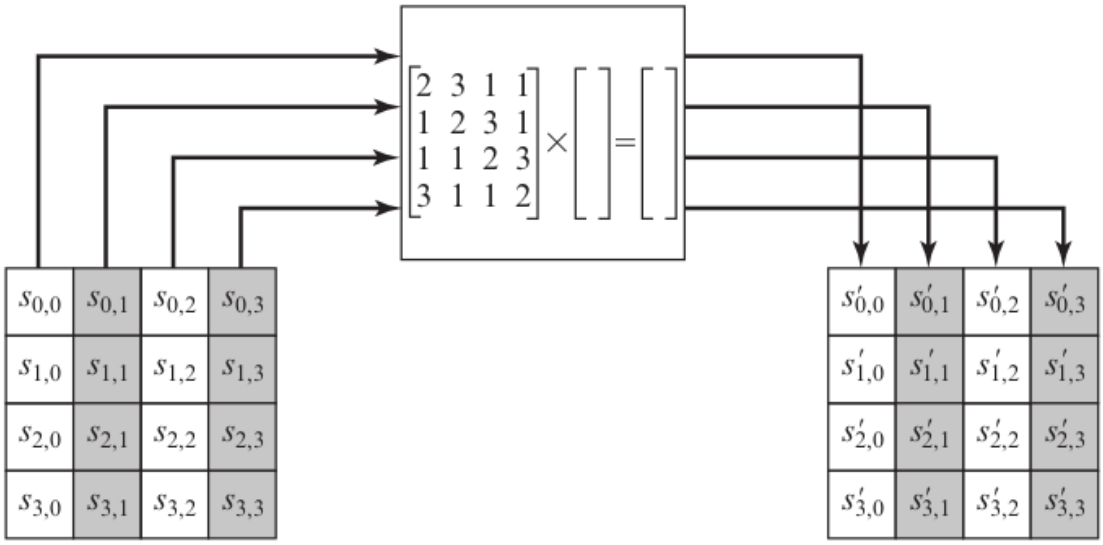

Figure 6: Example of Mix Column, matrix in the middle rectangle is the special constant matrix

- Coefficients based on linear code with maximal distance between code words
- Shift row plus mix columns (all output bits depend on all input bits after a few rounds

Add Round Key

Operation is just XOR.

Design criteria

- fast on wide range of CPUs
- Diffusion of key bits into round keys
- Enough nonlinearity to prohibit determination of round key differences from cipher key differences

AES Key Expansion

1. Start with a cipher key (16 byte). This is given to us.
2. Generate the first round key that will be used by one of the rounds in the AES
   - First Column First Round Key
     a. Take 4th column from cipher key (Rot Word)
     b. Move the 1st byte in that column and move it to the bottom of the column
     c. Now perform the S-box substitution on this column
     d. XOR with Column 1 of cipher key and RCon of your current round
   - Second Column First Round
     – XOR column 2 in cipher key with First column First Round key
   - Third Column First Round
     – XOR column 3 in cipher key with Second column First Round key
   - Fourth Column First Round
     – XOR column 4 in cipher key with Third column First Round key
3. Generate the rest of the round key
   - Basically the same algorithm except use the previous round key as your new “cipher key”

Key Expansion Rationale

Designed to resist known attacks. Design criteria includes:

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

Figure 7: Table of RCON values corresponding to the respect round j

- Knowing part of key is insufficient to find more key bits
- An invertible transformation with Nk consecutive words of expanded keys
- Fast on wide range of CPUs
- Use round constants to eliminate symmetries
- Diffusion of key bits
- Enough non-linearity to prohibit determination of round key differences from cipher key differences
- Simplicity of description

[LINK]: great animation of how key expansion works[1]

Decryption and Implementation Aspects

- AES decryption is not identical to encryption as their sequence of transformations are different
  - Need separate implementation modules
  - Some cipher modes of operation and MAC only uses encryption
- Can be efficiently implemented on 8-bit CPU
- Can be efficiently implemented on 32-bit CPU

Links

1. https://www.youtube.com/watch?v=gP4PqVGudtg