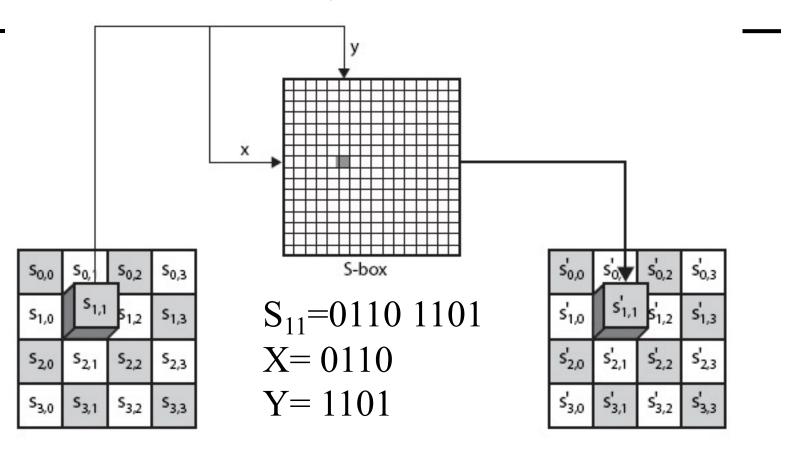
Four Steps in Each Round

- Substitute Bytes single byte based
- Shift Rows row-wise permutation
- Mix Columns column-wise mixing
- Add Round Keys

AES: SubBytes() (S-Box)



A simple substitution of each byte Uses one table of 16x16 bytes containing a permutation of all 256 8-bit values

AES S-Box

5 50%															
00	01	02	03	04	05	06	07	08	09	0a	0b	0с	0d	0e	Of
63	7c	77	7b	f2	6b	6f	с5	30	01	67	2b	fe	d7	ab	76
ca	82	с9	7d	fa	59	47	fO	ad	d4	a2	af	9с	a4	72	c0
b7	fd	93	26	36	3f	f7	СС	34	a5	e5	f1	71	d8	31	15
04	с7	23	сЗ	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	е3	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	3с	9f	a8
51	аЗ	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	0c	13	ес	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	За	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	Зе	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9е
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e 9	се	55	28	df
8c	a1	89	0d	bf	e6	42	68	41	99	2d	Of	b0	54	bb	16
	63 ca b7 04 09 53 d0 51 cd 60 e0 e7 ba 70 e1	63 7c ca 82 b7 fd 04 c7 09 83 53 d1 d0 ef 51 a3 cd 0c 60 81 e0 32 e7 c8 ba 78 70 3e e1 f8	63 7c 77 ca 82 c9 b7 fd 93 04 c7 23 09 83 2c 53 d1 00 d0 ef aa 51 a3 40 cd 0c 13 60 81 4f e0 32 3a e7 c8 37 ba 78 25 70 3e b5 e1 f8 98	63 7c 77 7b ca 82 c9 7d b7 fd 93 26 04 c7 23 c3 09 83 2c 1a 53 d1 00 ed d0 ef aa fb 51 a3 40 8f cd 0c 13 ec 60 81 4f dc e0 32 3a 0a e7 c8 37 6d ba 78 25 2e 70 3e b5 66 e1 f8 98 11	63 7c 77 7b f2 ca 82 c9 7d fa b7 fd 93 26 36 04 c7 23 c3 18 09 83 2c 1a 1b 53 d1 00 ed 20 d0 ef aa fb 43 51 a3 40 8f 92 cd 0c 13 ec 5f 60 81 4f dc 22 e0 32 3a 0a 49 e7 c8 37 6d 8d ba 78 25 2e 1c 70 3e b5 66 48 e1 f8 98 11 69	63 7c 77 7b f2 6b ca 82 c9 7d fa 59 b7 fd 93 26 36 3f 04 c7 23 c3 18 96 09 83 2c 1a 1b 6e 53 d1 00 ed 20 fc d0 ef aa fb 43 4d 51 a3 40 8f 92 9d cd 0c 13 ec 5f 97 60 81 4f dc 22 2a e0 32 3a 0a 49 06 e7 c8 37 6d 8d d5 ba 78 25 2e 1c a6 70 3e b5 66 48 03 e1 f8 98 11 <	63 7c 77 7b f2 6b 6f ca 82 c9 7d fa 59 47 b7 fd 93 26 36 3f f7 04 c7 23 c3 18 96 05 09 83 2c 1a 1b 6e 5a 53 d1 00 ed 20 fc b1 d0 ef aa fb 43 4d 33 51 a3 40 8f 92 9d 38 cd 0c 13 ec 5f 97 44 60 81 4f dc 22 2a 90 e0 32 3a 0a 49 06 24 e7 c8 37 6d 8d d5 4e ba 78 25 2e 1c a6	63 7c 77 7b f2 6b 6f c5 ca 82 c9 7d fa 59 47 f0 b7 fd 93 26 36 3f f7 cc 04 c7 23 c3 18 96 05 9a 09 83 2c 1a 1b 6e 5a a0 53 d1 00 ed 20 fc b1 5b d0 ef aa fb 43 4d 33 85 51 a3 40 8f 92 9d 38 f5 cd 0c 13 ec 5f 97 44 17 60 81 4f dc 22 2a 90 88 e0 32 3a 0a 49 06 24 5c e7 c8 37	63 7c 77 7b f2 6b 6f c5 30 ca 82 c9 7d fa 59 47 f0 ad b7 fd 93 26 36 3f f7 cc 34 04 c7 23 c3 18 96 05 9a 07 09 83 2c 1a 1b 6e 5a a0 52 53 d1 00 ed 20 fc b1 5b 6a d0 ef aa fb 43 4d 33 85 45 51 a3 40 8f 92 9d 38 f5 bc cd 0c 13 ec 5f 97 44 17 c4 60 81 4f dc 22 2a 90 88 46 e0 32	63 7c 77 7b f2 6b 6f c5 30 01 ca 82 c9 7d fa 59 47 f0 ad d4 b7 fd 93 26 36 3f f7 cc 34 a5 04 c7 23 c3 18 96 05 9a 07 12 09 83 2c 1a 1b 6e 5a a0 52 3b 53 d1 00 ed 20 fc b1 5b 6a cb d0 ef aa fb 43 4d 33 85 45 f9 51 a3 40 8f 92 9d 38 f5 bc b6 cd 0c 13 ec 5f 97 44 17 c4 a7 e0 81 4f	63 7c 77 7b f2 6b 6f c5 30 01 67 ca 82 c9 7d fa 59 47 f0 ad d4 a2 b7 fd 93 26 36 3f f7 cc 34 a5 e5 04 c7 23 c3 18 96 05 9a 07 12 80 09 83 2c 1a 1b 6e 5a a0 52 3b d6 53 d1 00 ed 20 fc b1 5b 6a cb be d0 ef aa fb 43 4d 33 85 45 f9 02 51 a3 40 8f 92 9d 38 f5 bc b6 da cd 0c 13 ec 5f 97	63 7c 77 7b f2 6b 6f c5 30 01 67 2b ca 82 c9 7d fa 59 47 f0 ad d4 a2 af b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 04 c7 23 c3 18 96 05 9a 07 12 80 e2 09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 53 d1 00 ed 20 fc b1 5b 6a cb be 39 d0 ef aa fb 43 4d 33 85 45 f9 02 7f 51 a3 40 8f 92 9d 38 f5 bc b6 da	63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 51 a3 40 8f	63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c d0 ef aa fb 43 4d 33 85 45 fp 02 7f	63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 d0 ef aa fb 43 4d

The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value $9a_{16}$ is converted into $b8_{16}$.

Example:

itution

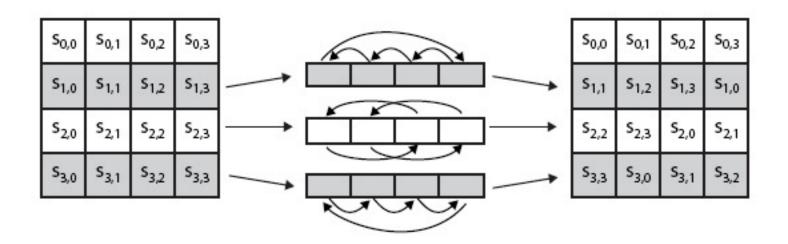
EA	04	65	85
83	45	5D	96
5C	33	98	В0
F0	2D	AD	C5

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

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

AES: ShiftRows()

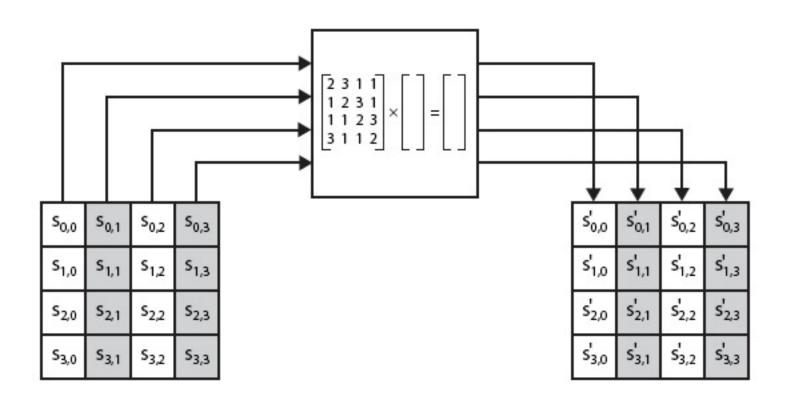


87	F2	4D	97	87	F2	4D	97
EC	6E	4C	90	6E	4C	90	EC
4A	C3	46	E7	46	E7	4A	C3
8C	D8	95	A6	A6	8C	D8	95

Mix Columns

- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column

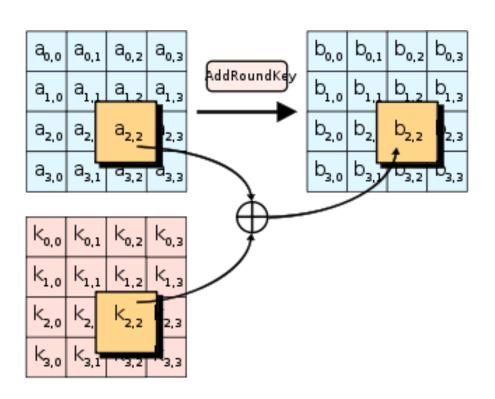
AES: MixColumns()



Add Round Key

- XOR state matrix with 128-bits of the round key
- Inverse for decryption is identical since XOR is own inverse, just with correct round key
- Designed to be as simple as possible

AES: AddRoundKey()



Implementation Aspects of AES

- Can be efficiently implemented even 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 more complicated operations (matrix multiply) on byte values, but can be simplified to use a table lookup and XORs.
 - Observe that each step is invertible, so decryption given key bits is straightforward
 - All operations can be combined into XOR and table lookups -hence very fast & efficient

Private-Key Cryptography

- Traditional **private/secret/single key** cryptography uses **one** key
- Shared by both sender and receiver
- If this key is disclosed communications are compromised
- Also is **symmetric**, parties are equal
- Hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

- Probably most significant advance in the 3000 year history of cryptography
- Uses **two** keys a public & a private key
- Asymmetric since parties are not equal
- Uses clever application of number theoretic concepts to function
- Complements rather than replaces private key crypto

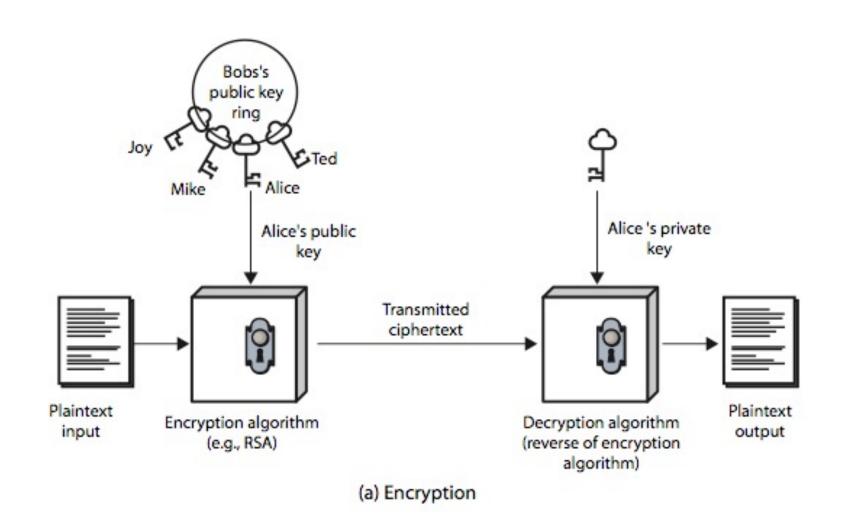
Why Public-Key Cryptography?

- Developed to address two key issues:
 - key distribution how to have secure communications in general without having to trust a key distribution center with your key
 - Digital signatures how to verify a message comes intact from the claimed sender
- Public invention due to Diffie & Hellman at Stanford University in 1976
 - known earlier in classified community

Public-Key Cryptography

- Public-key/two-key/asymmetric cryptography involves the use of two keys:
 - a public-key, which may be known by anybody, and can be used to encrypt messages, and verify signatures
 - a private-key, known only to one party, used to decrypt messages, and sign (create) signatures
- Asymmetric because
 - those who encrypt messages or verify signatures
 cannot decrypt messages or create signatures

Public-Key Cryptography



Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
 - when the relevant (en/decrypt) key is known it is computationally easy to en/decrypt messages
 - it is computationally infeasible to find decryption key, knowing only algorithm & encryption key

Diffie & Hellman Key Exchange

- Alice and Bob want to share a secret (e.g., a key) in an open channel.
- Assume they agree on two numbers n and g
- g is primitive root mod (n)
 - For each p < n s.t. p is coprime to n, there is an a such that

$$g^a = p \mod (n)$$

• These *g* and *n* do not have to be kept secret

Alice

- Chooses a large random number *x*
- Calculates

$$X = g^x \bmod (n)$$

• Sends *X*, *g*, and *n* to Bob.

Bob

- Chooses a large random number y
- Calculates

$$Y = g^y \bmod (n)$$

• Sends Y to Alice.

• Alice calculates

$$k = Y^x \mod(n)$$

Bob calculates

$$k' = X^y \mod(n)$$

The Key

• k' = k is the shared key

$$k = Y^x \bmod (n) = (g^y)^x \bmod (n) = g^{yx} \bmod (n)$$

$$k' = X^y \mod (n) = (g^x)^y \mod (n) = g^{xy} \mod (n)$$

No efficient classical algorithm for computing general discrete logarithms.

Nobody can calculate k given
 n, g, X, and Y

Example

Alice and Bob get public numbers

$$-n = 23, g = 9$$

Alice and Bob compute public values

$$-X = 9^4 \mod 23 = 6561 \mod 23 = 6$$

$$-Y = 9^3 \mod 23 = 729 \mod 23 = 16$$

Alice and Bob exchange public numbers

Example

- Alice and Bob compute symmetric keys
 - $-k_a = 16^4 \mod 23 = 9$
 - $-k_b = 6^3 \mod 23 = 9$
- Alice and Bob now can talk securely!

Diffie-Hellman Limitations

- Only Alice and Bob know *k*
- Good for only one session
- Used if you only want a symmetric key
- Can't be sure connected to the same person
- No authentication