RSA & AES Algorithm

Dr. Risala Tasin Khan

Professor

IIT, JU

RSA Cryptosystem

- RSA is the most commonly used public-key cryptography algorithm, which uses prime factorization as the trapdoor one-way function. That is, it is based on the presumed difficulty of factoring large integers.
- It is named so after the surnames of its inventors Ron Rivest, Adi Shamir, and Leonard Adleman of the Massachusetts Institute of Technology (MIT).
- It was first published in 1978.
- This algorithm relies on one way function. A one way function is easy to compute but hard to invert. For example it is easy to take the product of two prime numbers but given the product, it is difficult to split it into the original prime factors.
- This algorithm lets you choose the size of your public key.
  - The 512-bit keys are considered insecure or weak, but the 768-bit keys are secure from everything but the National Security Administration (NSA).
  - The 1024-bit keys are secure from everything virtually.
- RSA is embedded in major products such as Windows, Netscape Navigator etc.

How the RSA Cryptosystem Works?

- Briefly, the RSA algorithm involves multiplying two large prime numbers P and Q and through additional operations deriving a set of two numbers e and d where e is the public key and d is the private key.
- Once the keys have been developed, the original prime numbers are no longer important and can be discarded.
- Both the public and the private keys are needed for encryption /decryption but only the owner of a private key ever needs to know it.
- Using the RSA system, the private key never needs to be sent across the Internet.
- Anyone can use the public key to encrypt a message. But the message can be decrypted only by the owner of the private key.
  - Thus, if Alice wants to send a message to Bob, she can find out Bob's public key (but not his private key) from a central administrator.
  - After getting the public key of Bob, Alice then encrypt the message using Bob's public key and sends the encrypted message to Bob.
  - When Bob receives it, he decrypts it with his private key.
- In addition to encrypting messages (which ensures privacy), Bob can authenticate himself to Alice (so Alice knows that it is really Bob who sent the message) by using Bob's private key to encrypt a digital certificate. When Alice receives it, she can use Bob's public key to decrypt it.

Steps in RSA Algorithm

- The RSA algorithm involves three steps:
  1. Key generation (Generating public and private key)
  2. Encryption (Encrypting the message)
  3. Decryption (Decrypting the message)

RSA Algorithm: Key Generation

The keys for the RSA algorithm are generated by the following ways:

1. Choose two large and distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length.
   - In RSA, p and q must be at least 512 bits; n must be at least 1024 bits.
   - Prime integers can be efficiently found using a primality test.
2. Compute n = p * q
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute the number of integers less than n that are coprime with n (otherwise known as the totient or Euler's Phi function):

$$\varphi(n) = \varphi(p*q) = \varphi(p)* \varphi(q)=(p - 1) * (q - 1)$$

4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$; i.e. e and $\varphi(n)$ are coprime.
   - e is released as the public key exponent (encryption exponent).
   - e having a short bit-length results in more efficient encryption– most commonly $2^{16} + 1 = 65,537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.

RSA Algorithm: Key Generation

5. Determine the multiplicative inverse d of e; i.e., compute a value for d such that it satisfies the relation: (d * e) mod φ(n) = 1
   - d is kept as the private key exponent (decryption exponent).
   - d is often computed using the Extended Euclidean Algorithm.
   - d must be kept secret.
   - p, q, and φ(n) must also be kept secret because they can be used to calculate d.
7. The public key consists of the modulus n and the public key exponent e; i.e., the public key is (e, n).
8. The private key consists of the modulus n and the private key exponent d; i.e., the private key is (d, n).
9. To encrypt message m using the public key, use the relation:

$$c = m^e \bmod n$$

9. To decrypt c using the private key, use the relation:

$$m = c^d \bmod n$$

RSA Algorithm: Encryption

- Bob transmits his public key (e, n) to Alice and keeps the private key (d, n) secret.
- Alice then wishes to send message M to Bob.
- The message is encrypted by the following ways:

1. Alice first turns message M into an integer m, such that $0 \leq m < n$.
   - That is, the message is represented as an integer between 0 and (n-1).
   - Large messages can be broken up into a number of blocks. Each block would then be represented by an integer in the same range.
2. After turning the message into integer, Alice then computes the ciphertext c using the following relation:

$$c = m^e \bmod n$$

3. After computing ciphertext, Alice then transmits c to Bob.

RSA Algorithm: Decryption

- Bob can recover m from c by using his private key exponent d using the following relation:

$$m = c^d \bmod n$$

- After having m, Bob can recover the original message M by reversing the padding scheme.
- The encryption, decryption and key generation in RSA is shown in the figure below.

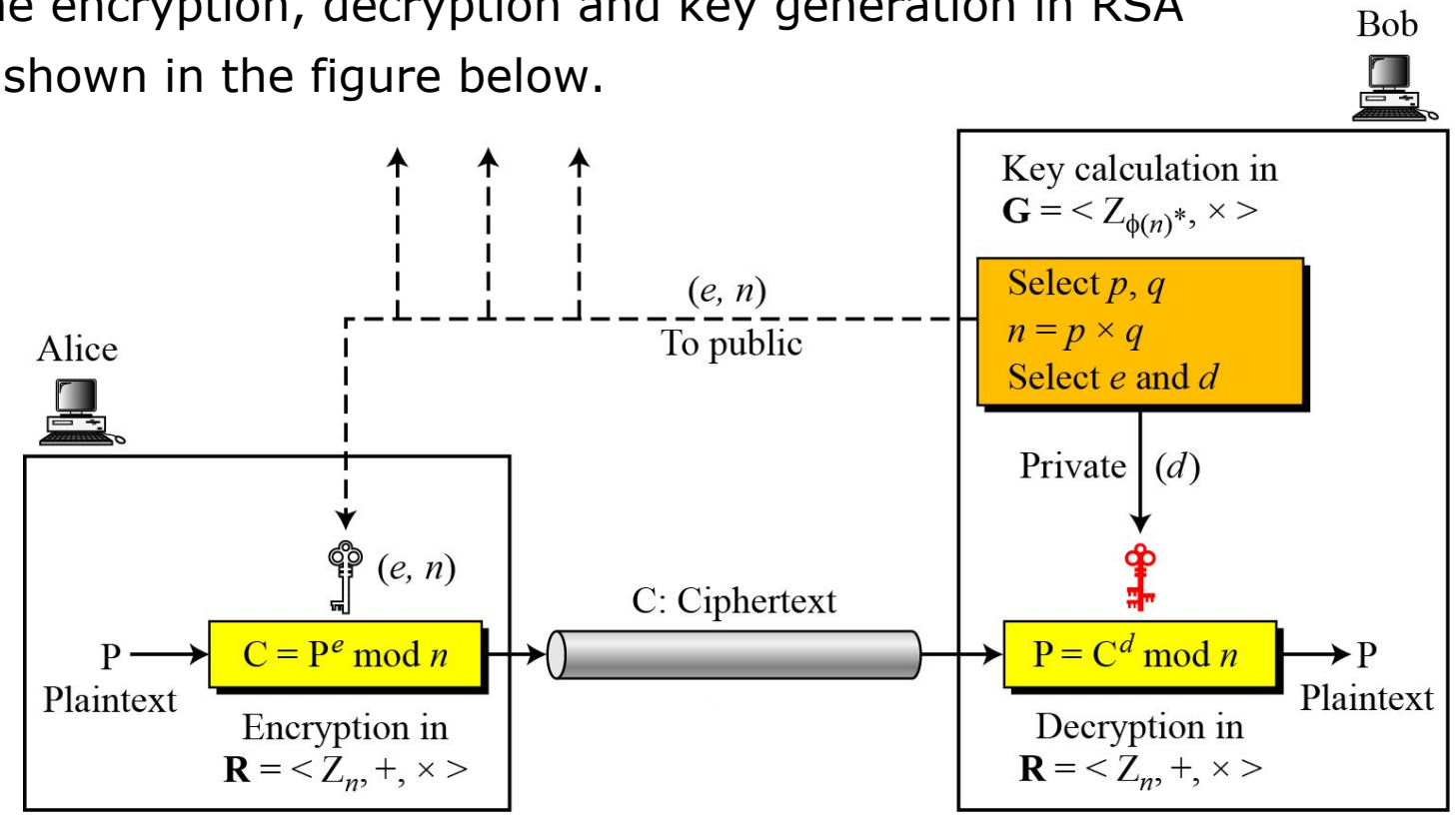


Figure: Encryption, decryption, and key generation in RSA

RSA Cryptosystem: Trivial Examples

Example-1:

1. Choose p = 3 and q = 11
2. Compute n = p * q = 3 * 11 = 33
3. Compute φ(n) = φ(p*q) = φ(p)* φ(q)=(p - 1) * (q - 1) = 2 * 10 = 20
4. Choose e such that 1 < e < φ(n) and e and φ(n) are coprime. We have several choices for e: 7, 11, 13, 17, 19. (We cannot use 5 as e, because 20 is divisible by 5). Let e = 7
5. Compute a value for d such that (d * e) mod φ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1] [d is the multiplicative inverse of e]
6. Public key is (e, n) => (7, 33)
7. Private key is (d, n) => (3, 33)
8. The encryption of m = 2 is c = m^e mod n = 2^7 mod 33 = 29
9. The decryption of c = 29 is m = c^d mod n = 29^3 mod 33 = 2

RSA Cryptosystem: Trivial Examples

Example-2:

- Bob chooses 7 and 11 as p and q.
- He calculates n = 77. The value of φ(n) = (7 − 1)(11 − 1) or 60.
- Now he chooses two exponents, e and d, from Z_{60}*. If he chooses e to be 13, then d is 37. Note that e × d mod 60 = 1 (they are inverses of each).
- Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

Plaintext: 5	$C = 5^{13} = 26 \bmod 77$	Ciphertext: 26

- Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

Ciphertext: 26	$P = 26^{37} = 5 \bmod 77$	Plaintext: 5

RSA Cryptosystem: Trivial Examples

Example-3:

- Now assume that another person, John, wants to send a message to Bob.
- John can use the same public key announced by Bob (probably on his website), 13.
- John's plaintext is 63. John calculates the following:

Plaintext: 63	$C = 63^{13} = 28 \bmod 77$	Ciphertext: 28

- Bob receives the ciphertext 28 and uses his private key 37 to decipher the ciphertext:

Ciphertext: 28	$P = 28^{37} = 63 \bmod 77$	Plaintext: 63

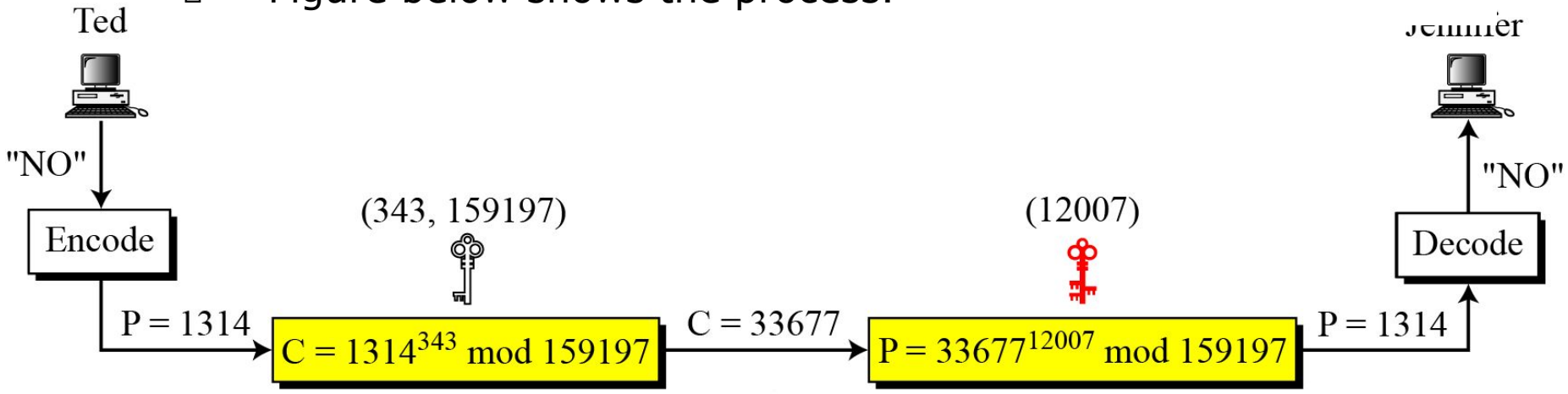
RSA Cryptosystem: Trivial Examples

Example-4:

- Jennifer creates a pair of keys for herself. She chooses p = 397 and q = 401.
- She calculates n = 159197. She then calculates φ(n) = 158400. She then chooses e = 343 and d = 12007.
- Show how Ted can send a message to Jennifer if he knows e and n.

Solution:

- Suppose Ted wants to send the message "NO" to Jennifer.
- He changes each character to a number (from 00 to 25), with each character coded as two digits.
- He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314.
- He then uses e and n to encrypt the message.
- Figure below shows the process.



Attacks on RSA Cryptosystem

- No devastating attacks on RSA have been yet discovered.
- Several attacks have been predicted based on the weak plaintext, weak parameter selection, or inappropriate implementation.
- Figure below shows the category of potential attacks on RSA.

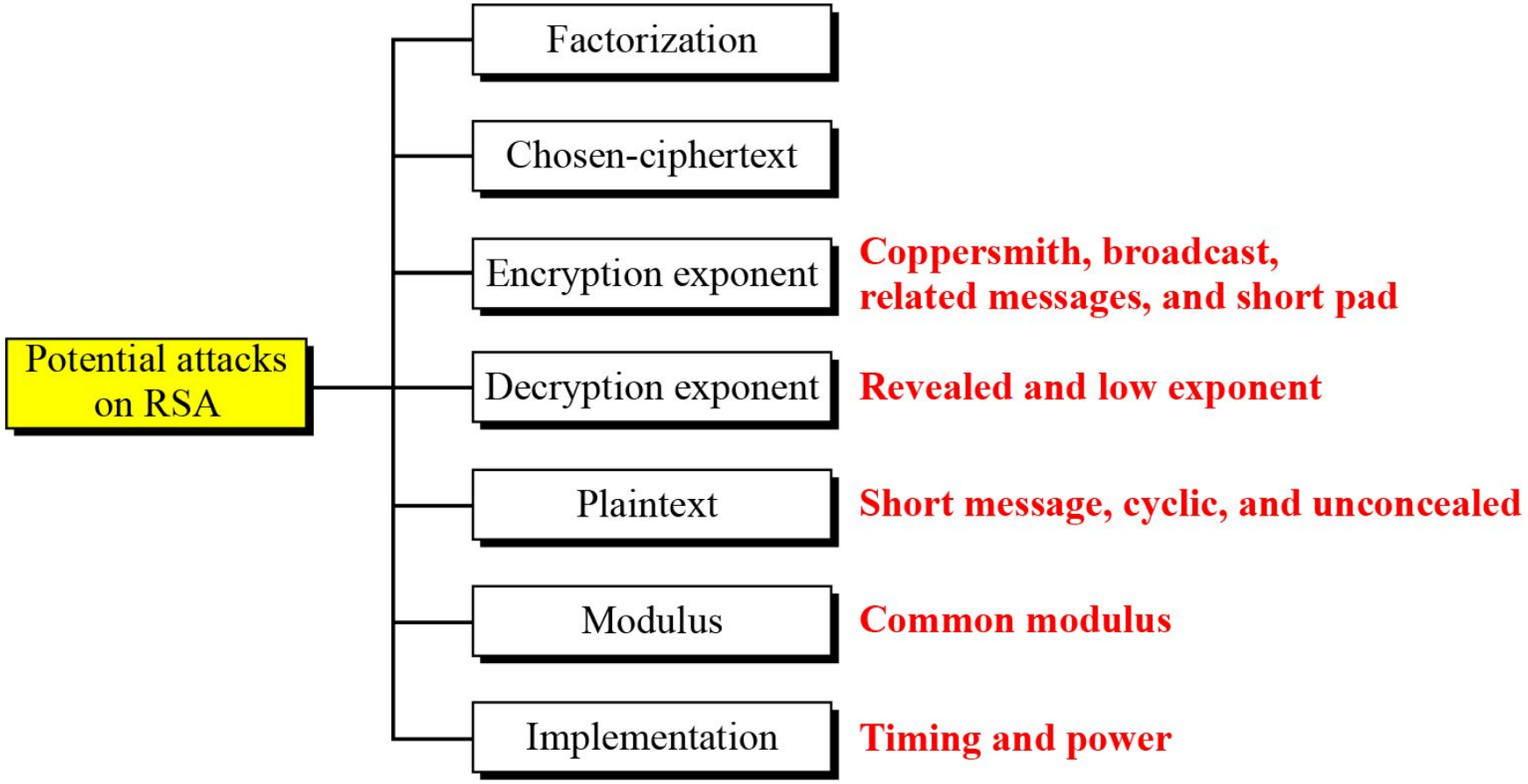


Figure: Taxonomy of potential attacks on RSA

RSA Cryptosystem: Cracking the Code

- The essential requirement of the Public Key Cryptography, like RSA, is that-the public and secret keys are mathematically related, but this relationship must be made very hard to determine by an outsider.
- As we saw in the preceding text, everything starts with p and q, from which we calculated n.
- The public key consists of two numbers: e and n, where e is calculated from φ(n), and φ(n) is calculated from p and q.
- The secret key d, was calculated from e and φ(n) and, as we just stated, e and φ(n) are calculated from p and q.
- It follows then, that d is also calculated from p and q, which proves that the public and private keys are mathematically related.
- So, if an adversary (like Eve) wanted to find the secret key d, by only knowing n, he must break down n into the two prime numbers that were used to produce it (remember that n = p * q).
- Now, here is the real crux of the bisquit: Decomposing a very large n into p and q is really difficult to do. It is easy with the small numbers that we have used in our demonstration, but, for example, if 100 digit numbers are used for p and q, the resulting n will be approximately 200 digits. Then decomposing n into p and q will be very hard. The fastest known factoring algorithm would take far too long for an attacker to ever break the code.
- Any cryptographic technique which can resist a concerted attack is regarded as secure. At this point in time, the RSA algorithm is considered secure.

RSA Cryptosystem

- In RSA cryptosystem, encryption using public keys is normally computationally intensive. So, in practice-
  - The sender encrypts the message with a secret key that is randomly generated.
  - The secret key is encrypted using the public key of the recipient and sent with the encrypted message.
  - The recipient decrypts the secret key using his private key and using that secret key, he decrypts the rest of the message.
- The following lists all the steps in the process:
  1. The client(Alice) and server(Bob) go through a handshaking procedure.
  2. The handshake begins when the client connects to a SSL enabled server requesting a secure connection and presents a list of encryption algorithms and hash functions that it supports.
  3. From this list the server chooses the most secure encryption algorithm and hash function that it also supports and lets the client know about its choice.
  4. In the above transaction, the server also sends it identification in the form of a digital certificate. The digital certificate contains the server's name, the trusted Certificate Authority, and the server's public encryption key.
  5. The client may contact the trusted Certificate Authority for verification.
  6. The client generates a random number and encrypts it with the server's public key and sends it to the server. Only the server can decrypt this with its private key.
  7. The random number generated by the client is then used in the encryption and decryption process on both the client and server sides.

RSA Cryptosystem: Applications

- Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long.
- Therefore, RSA is useful for short messages.
- RSA is used in digital signature and other cryptosystems that often need to encrypt a small message without having access to a symmetric key.
- RSA is also used for authentication.

What is AES (Advanced Encryption Standard):

The AES algorithm, also known as the Rijndael algorithm is a symmetric block cipher that takes a block size of 128 bits and converts them into ciphertext using keys of 128, 192, and 256 bits.

- Rijndael is an iterated block cipher.
- This type of algorithm can encrypts and decrypts a block of data by the iteration or round of a specific transformation.
- Rijndael is named after its creators Vincent Rijment and Joan Daemen - the two Belgian cryptologists.
- The weakness found in DES algorithm was solved by Rijndael algorithm.

History of AES (Advanced Encryption Standard):

- Until 2000, DES (Data Encryption Standard) had been used as a standard method of encryption, but with increase in speed in computers and having shorter key length, it is no more considered secure as a cryptanalyst can break the code by exhaustively searching for all the keys using a fast computer.
- From 2001, DES has been replaced by a new standard known as the Advanced Encryption Standard (AES) which is published by the National Institute of Standards and Technology (NIST).
  - In 1997, NIST started looking for a replacement for DES, which would be called the Advanced Encryption Standard or AES.
  - The NIST specifications required a block size of 128 bits and three different key sizes of 128, 192, and 256 bits.
  - The specifications also required that AES be an open algorithm, available to the public worldwide. The announcement was made internationally to solicit responses from all over the world.
  - After the *First AES Candidate Conference*, NIST announced that 15 out of 21 received algorithms had met the requirements and been selected as the first candidates (August 1998).
    - Algorithms were submitted from a number of countries; the variety of these proposals demonstrated the openness of the process and worldwide participation.

History of AES (Advanced Encryption Standard):

- After the ***Second AES Candidate Conference***, which was held in Rome, NIST announced that **5** out of **15** candidates (algorithms) **—MARS, RC6 (Rivest Ciphers), Rijndael, Serpent, and Twofish**— were selected as the finalists (**August 1999**).

- After the *Third AES Candidate Conference*, NIST announced that **Rijndael designed by** Belgian researchers **Joan Daemen** and **Vincent Rijment**, was selected as Advanced Encryption Standard (**October 2000**).

  - **Considering several factors like better performance, less cost, more security, high efficiency, openness, and ease of design and implementation, Rijndael algorithm was finally selected as AES by NIST**.

- **In February 2001**, NIST announced that a draft of the Federal Information Processing Standard (FIPS) was available for public review and comment.

- **Finally**, **AES was published** as **FIPS 197** in the Federal Register **in December 2001**.

Outstanding Features of AES:

- The features that make AES a unique algorithm are:

1. As a non-Feistel cipher, AES uses only invertible components.
2. Its design and implementation is very easy.
3. It provides better performance with less cost and more security.
4. It is an open algorithm that is available to the public worldwide
5. It uses Substitution and Permutations, also called SP Networks.
6. A single key is expanded to be used in multiple rounds.
7. AES performs on byte data, instead of bit data.
8. No. of rounds is dependent on key length.
9. Three different key length:
   - 128-bit Key Length uses 10 rounds
   - 192-bit Key Length uses 12 rounds
   - 256-bit Key Length uses 14 rounds

Criteria Defined by NIST for AES:

- The criteria defined by NIST for selecting AES fall into three areas:

1. Security:
   - The main emphasis was on security. Because NIST explicitly demanded a 128-bit key, this criterion focused on resistance to cryptanalysis attacks other than brute-force attack.
2. Cost:
   - The second criterion was cost, which covers the computational efficiency and storage requirement for different implementations such as hardware, software, or smart cards.
3. Implementation:
   - The third criterion was implementation. This criterion included the requirement that the algorithm must have flexibility (be implementable on any platform) and simplicity. It also required that AES be an open algorithm, available to the public worldwide.

- At the end, Rijndael was judged the best at meeting the combination of these criteria.

Parameters for Three Versions of AES:

- AES is a non-Feistel cipher (i.e., it uses only invertible components).
- In AES, there is no need to divide the plaintext into two halves as we saw in the Feistel ciphers like DES.
- AES is a block cipher that encrypts and decrypts data as a block of 128 bits.
- It uses 10, 12, or 14 rounds.
- Three different key sizes of 128, 192, and 256 bits can be used which depends on the number of rounds.
- AES has defined three versions with 10, 12, and 14 rounds. The versions are referred as AES-128, AES-192, and AES-256.
- Each version uses a different cipher key size (128, 192, or 256 bits), but the round keys (which are created by the **key-expansion algorithm**) are always 128 bits which is the same size as the plaintext or ciphertext block.

Common Parameters in AES-128

- Block Size: 128bit plain text (4 words/16 byte)
- No of Round: 10 rounds
- Key Size: 128bit (4 words/16 byte)
- No of Subkey: 44 subkeys
- Each subkey size: 32 bit/1 word/4 byte
- Each round: 4 subkeys
- Pre Round Calculation: 4 subkeys (128 bits/ 4 words/ 16 bytes)

Manner of Storing Input Data: Block-to-State Conversion

Lets see how data is being stored during the process of AES encryption.

- The plaintext block to be encrypted is just a sequence of 128 bits.
- AES works with byte quantities. So at first, we convert the 128 bits into 16 bytes.
- These 16 bytes of plaintext data is arranged in a 4 x 4 matrix format which is known as state array.
- Each round takes the state array as input and gives corresponding output of 4 x 4 matrix.
- At the start of the encryption, the 16 bytes of data, numbered are loaded into the array as shown in Table where each cell corresponds to one byte.
- 4 bytes (i.e., 32 bits) make one word, so each state array has 4 words.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Steps in AES Encryption Process

The AES encryption process uses a set of specially derived keys called round keys. Along with other operations, these round keys are applied on an array of data that holds exactly one block of data that is to be encrypted.

The steps in the encryption of AES 128-bit block are listed below:

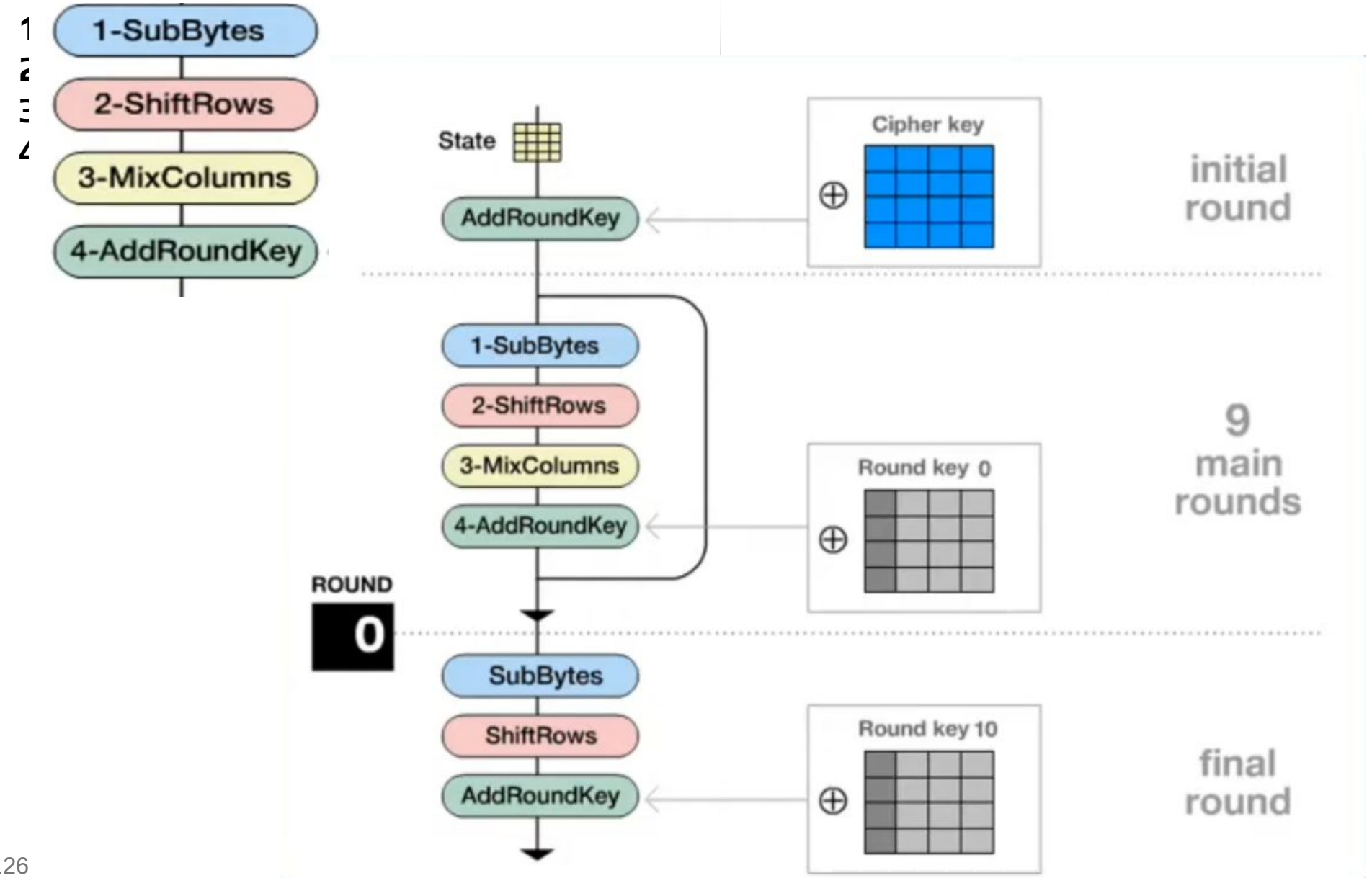
1) Derive the set of round keys from the cipher key.
2) Initialize the state array with the block data (plaintext).
3) Add the initial round key to the starting state array.
4) Perform nine rounds of state manipulation.
5) Perform the tenth and final round of state manipulation.

Note:

The reason that the rounds have been listed as "nine followed by a final tenth round" is because the tenth round involves a slightly different manipulation from the others.

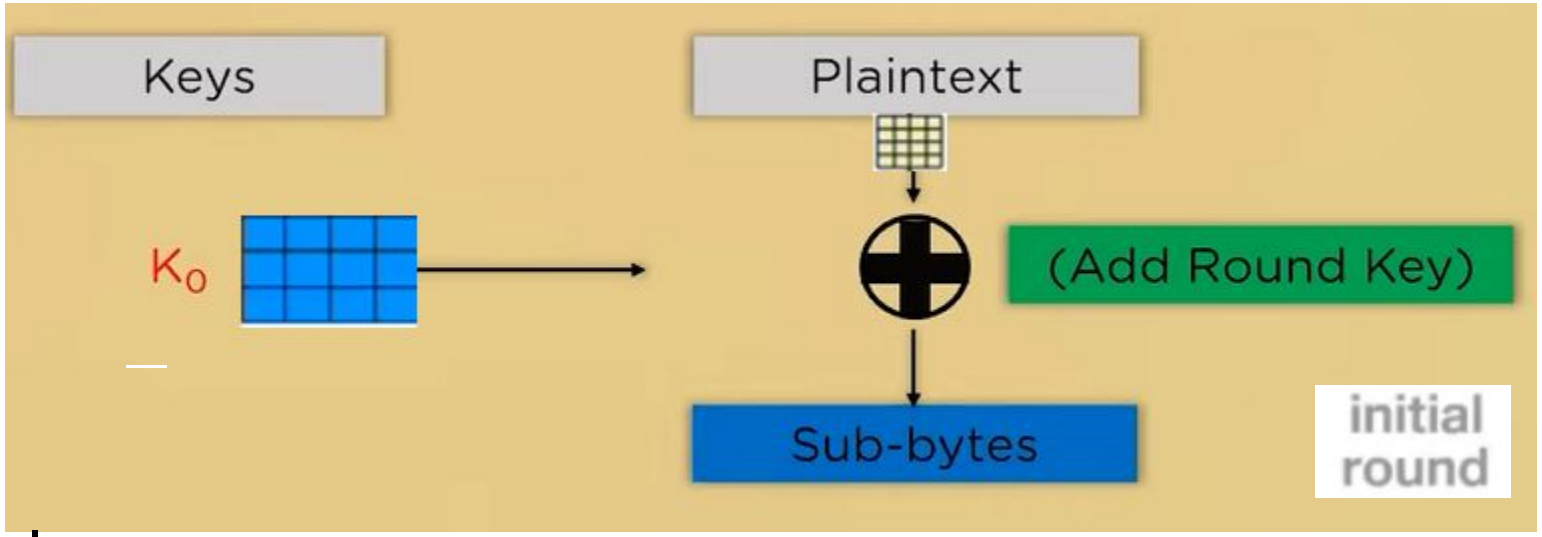
Operations Used In Computing Each Round

Four operations are required in each round:



Pre-round Operations in AES

- In the very beginning, the plaintext state array is Exor-ed with the initial round key as a supplement.
- The output is passed to a byte substitution process.



Example:

54		54		00
77	XOR	68	=	1F
6F		61		0E
20		74		54

54 = 01010100
54 = 01010100
00 = 00000000

77 = 01110111
68 = 01101000
1F = 00011111

54	4F	4E	20
77	6E	69	54
6F	65	6E	77
20	20	65	6F

Plaintext

XOR

54	73	20	67
68	20	4B	20
61	6D	75	46
74	79	6E	75

Round 0 Key

00	3C	63	47
1F	4E	22	74
0E	08	1B	31
54	59	0B	1A

New State Array

Round Operations in AES

Following four operations are required to perform in round-1 to round-9:

1. **SubBytes**
2. **ShiftRows**
3. **MixColumns**
4. **XorRoundKey**

In the final round, following three operations are required to perform

1. **SubBytes**
2. **ShiftRows**
3. **XorRoundKey**

SubByte Operation

- **This step implements the substitution.**
- In this step, each byte is substituted by another byte.
- **It is performed using a lookup table also called the S-box.**
- This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte.
- The result of this step is a 16-byte (4 x 4) matrix like before.

1. SubBytes Operations

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

State Matrix before SubBytes Operation

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

State Matrix after SubBytes Operation

19

9

1

hex	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
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

x

S-BOX byte substitution table

Shift Row Operation

- This step is just as it sounds.
- Each row is shifted a particular number of times.
- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

Operations Used In Computing Each Round

2. ShiftRows Operation:

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

← Row 0: Rotate each byte from right to left over 0 byte

← Row 1: Rotate each byte from right to left over 1 byte

← Row 2: Rotate each byte from right to left over 2 byte

← Row 3: Rotate each byte from right to left over 3 byte

State Matrix before ShiftRows Operation

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

State Matrix after ShiftRows Operation

Mix Column Operation

- This step is a matrix multiplication.
- Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

Operations Used In Computing Each Round

3. MixColumn Operation:

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

State Matrix before MixColumn Operation

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ e5 \end{bmatrix}$$

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

State Matrix after MixColumn Operation

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

The four numbers of one column are modulo multiplied in Rijndael's Galois Field by a given matrix.

Add Round Key Operation

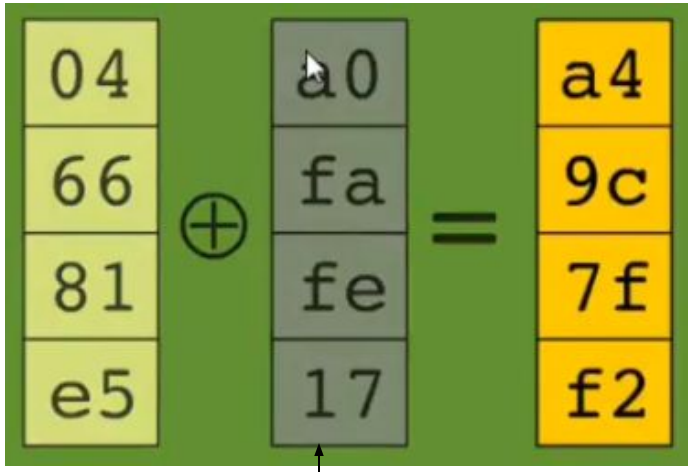
- Now the resultant output of the previous stage is XOR-ed with the corresponding round key.
- Here, the 16 bytes are not considered as a grid but just as 128 bits of data.
-

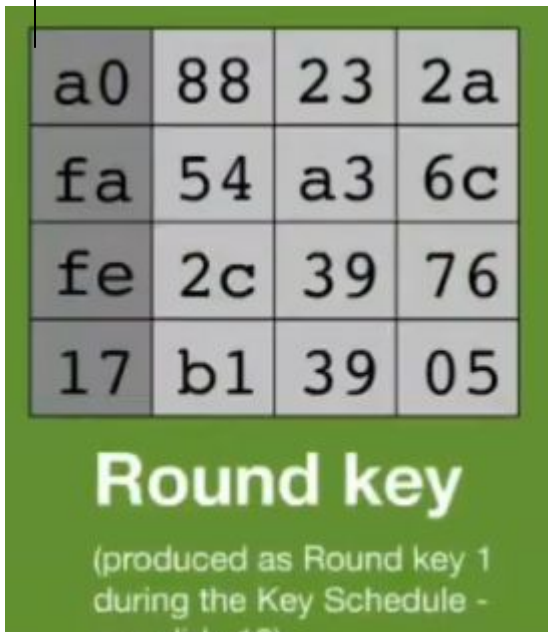
Operations Used In Computing Each Round

4. AddRoundKey Operation:

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

State Matrix before AddRoundKey Operation





a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

State Matrix after AddRoundKey Operation

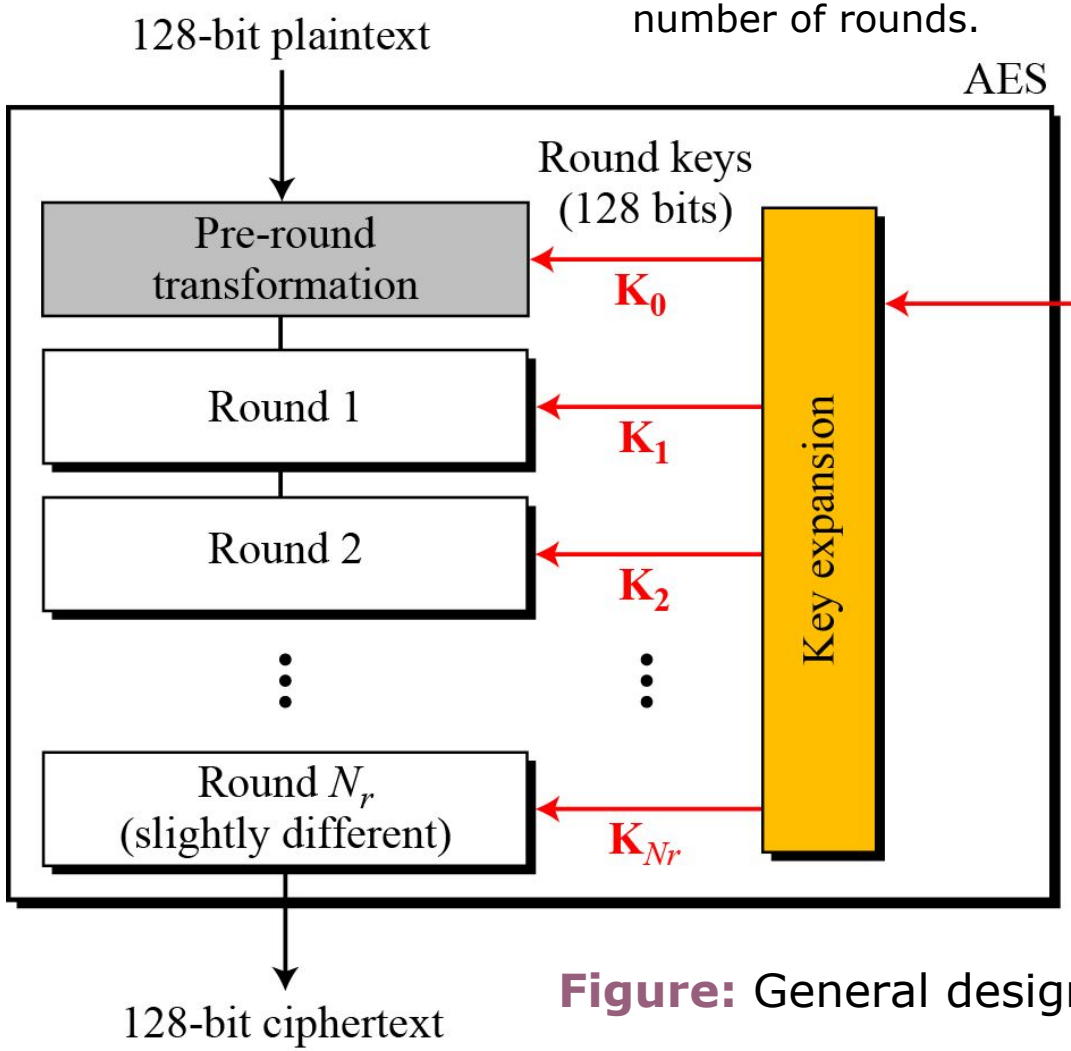
AES Key Expansion

The cipher key is expanded to n + 1 rounds, with n being the number of rounds.

- 4 words in each key.
- Each key is used for a single round. The first key is used as initial round key before any round begins.

General Design of AES Encryption Cipher:

- Figure below shows the general design for the encryption algorithm (called cipher). N_r defines the number of rounds. Figure also shows the relationship between the number of rounds and the key size.
- The decryption algorithm (called inverse cipher) is similar, but the round keys are applied in the reverse order.
- The number of round keys generated by the key-expansion algorithm is always one more than the number of rounds.
- In other words, we have: Number of round keys = N_r+1
- We refer to the round keys as $K_0, K_1, K_2, \ldots, K_{Nr}$.



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

Figure: General design of AES encryption cipher

Data Units in AES:

- AES uses five units of measurement to refer to data:

1. **Bit:**
   - In AES, a bit is a binary digit with a value of 0 or 1 . We use a lowercase letter b to refer to a bit.
2. **Byte:**
   - A byte is a group of eight bits that can be treated as a single entity: a row matrix (1 x 8) of eight bits, or a column matrix (8 x 1) of eight bits.
   - When treated as a row matrix, the bits are inserted to the matrix from left to right; when treated as a column matrix, the bits are inserted into the matrix from top to bottom. We use a lowercase bold b letter to refer to a byte.
3. **Word:**
   - A word is a group of 32 bits (4 bytes) that can be treated as a single entity, a row matrix of four bytes, or a column matrix of four bytes.
   - When it is treated as a row matrix, the bytes are inserted into the matrix from left to right; when it is considered as a column matrix, the bytes are inserted into the matrix from top to bottom. We use the lowercase bold letter w to show a word.
4. **Blocks:**
   - AES encrypts and decrypts data as blocks. A block in AES is a group of 128 bits (4 words or 16 bytes). However, a block can be represented as a row matrix of 16 bytes.

Data Units in AES:

5. State:

- AES uses several rounds in which each round is made of several stages.
- Data block is transformed from one stage to another.
- At the beginning and end of the cipher, AES uses the term data block; before and after each stage, the data block is referred to as a state.
- We use an uppercase bold letter **S** to refer to a state.
- Although the states in different stages are normally called S, we occasionally use the letter T to refer to a temporary state.
- States, like blocks, are made of 16 bytes, but normally are treated as matrices of 4 x 4 bytes. In this case, each element of a state is referred to as $S_{r,c}$, where r (0 to 3) defines the row and the c (0 to 3) defines the column.

Block-to-state and state-to-block transformation:

Example:

- Let us see how **a 16-character block** can be shown as a 4 x 4 matrix.
- Assume that the text block is "AES uses a matrix".
- We **add two bogus characters at the end** to get "AESUSESAMATRIXZZ".
- Now we **replace each character with a decimal integer between 00 and 25**.
- We then show each byte as an integer with two hexadecimal digits. For example, the character "S" is first changed to 18 and then written as 12_{16} in hexadecimal. The state matrix is then filled up, **column by column**, as shown in the figure below.

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

Figure: Changing plaintext to state