Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001. AES is a symmetric block cipher, proposed by Rijndael

Size of Data Block	*Number of Rounds*	*Key Size*
128 bits	10	128 bits
	12	192 bits
	14	256 bits

Round Key size	**Expanded Key size**
128 bits	**44 words**

AES have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes (data block of 4 columns of 4 bytes is state). This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix.

Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key. The key that is provided as input is expanded into an array of forty-four 32-bit words, **w**[*i*]. Four distinct words (128 bits) serve as a round key for each round;

4 Transformations are used, one of permutation and 3 of substitution.

1. Byte Substitution (1 S-box used on every byte)
2. Shift Rows (permute bytes between groups/columns)
3. Mix Columns (subs using matrix multiply of groups, Except 10^{th} round)
4. Add Round Key (XOR state with key material)

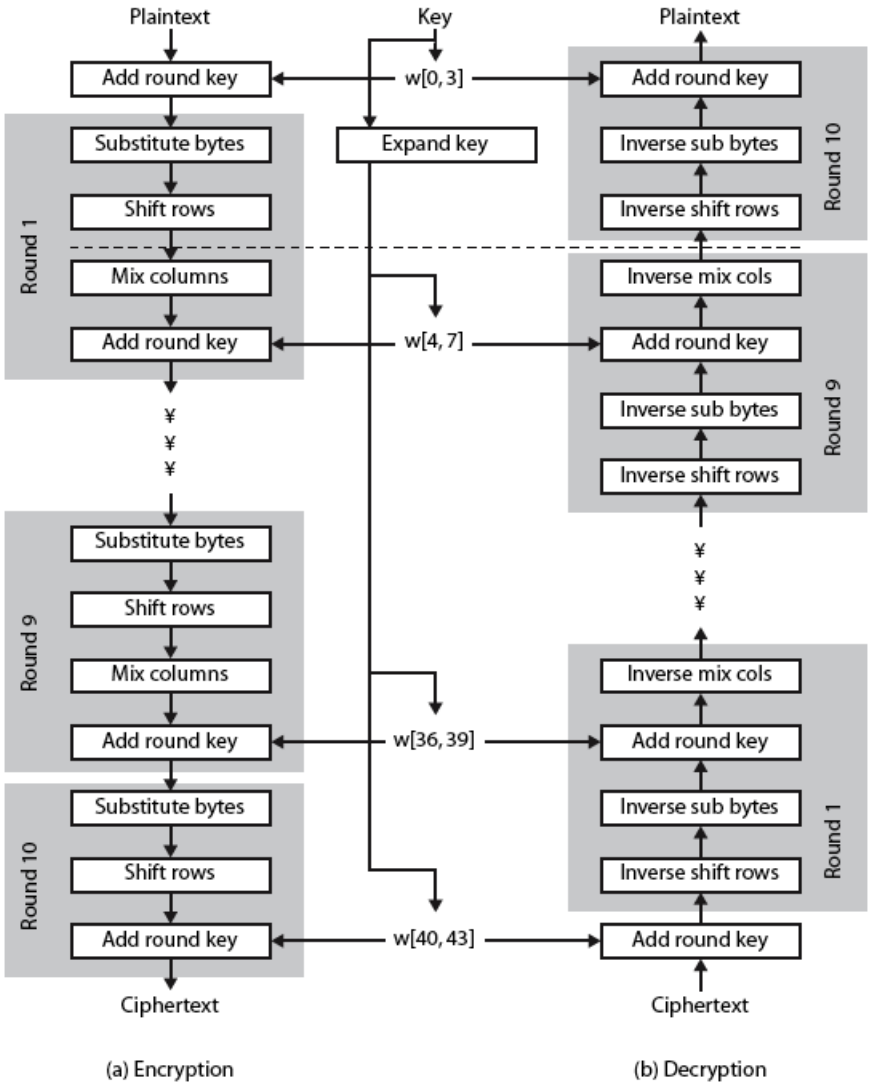


Fig: AES Structure

The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.

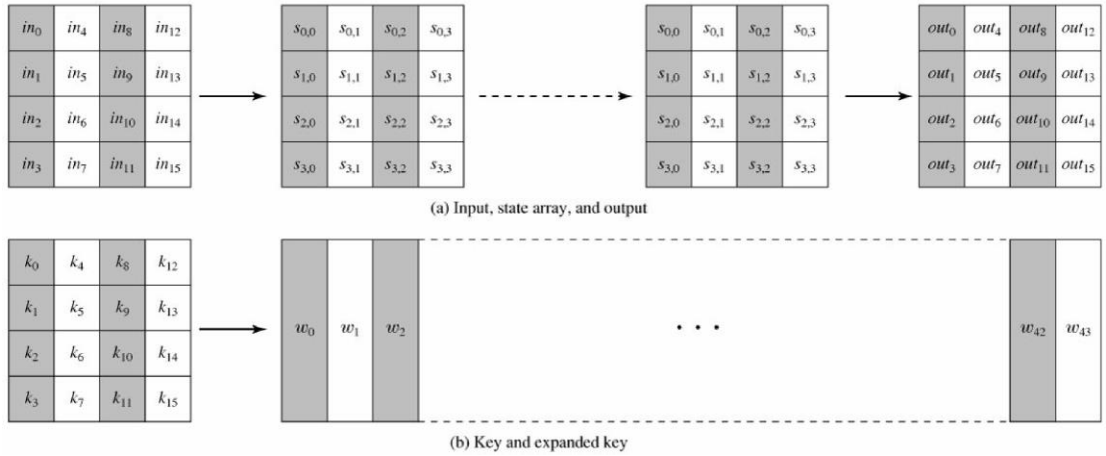


Fig: Key and Expanded Key

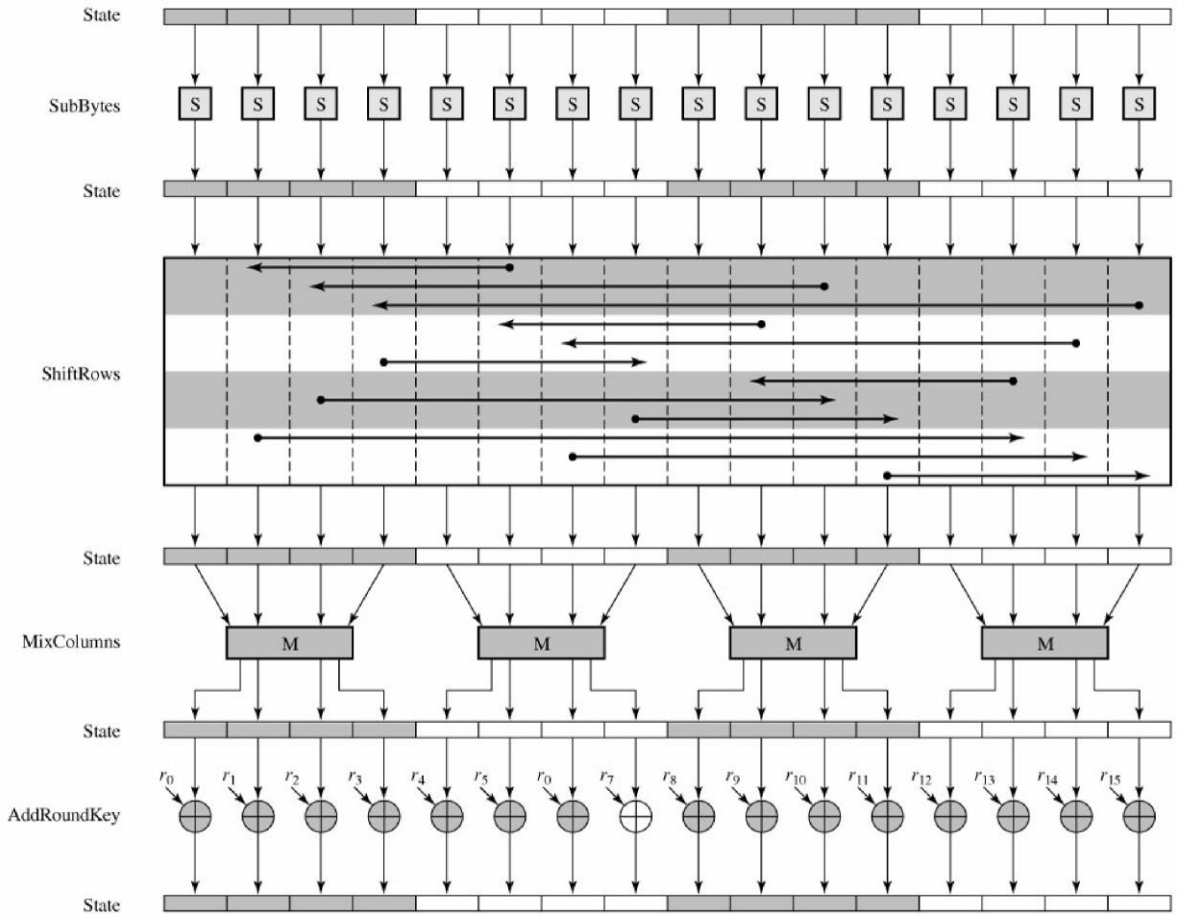
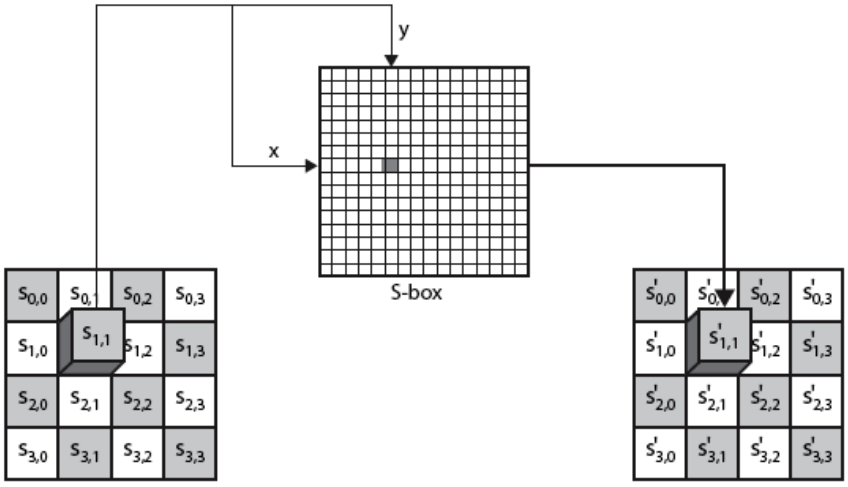


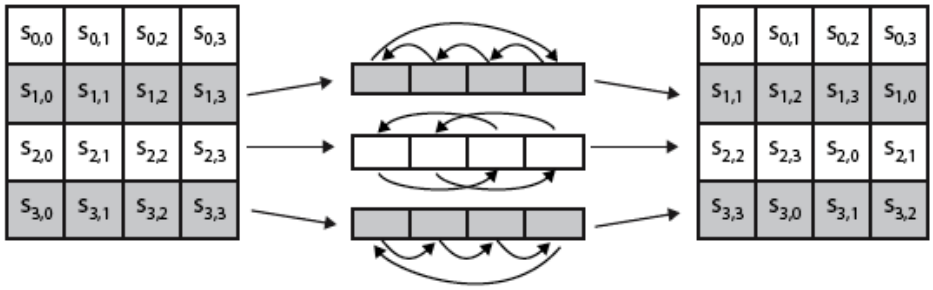
Fig: AES Encryption Round

1. Byte Substitution



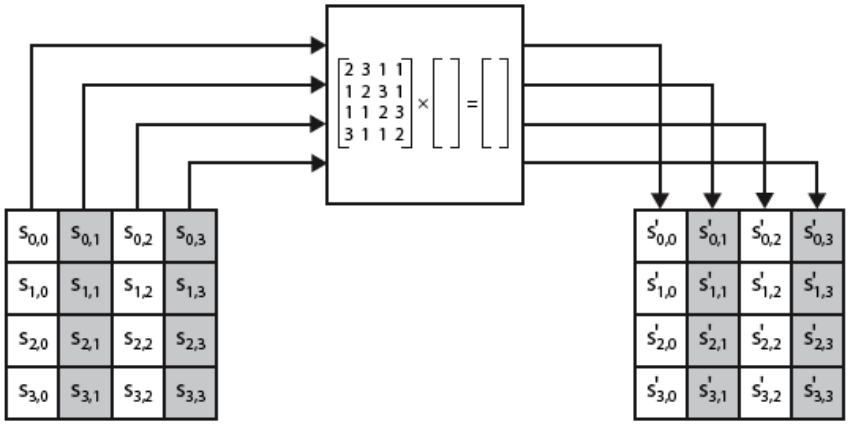
The **forward substitute byte transformation**, called SubBytes, is a simple table lookup table. AES defines a 16 x 16 matrix of byte values, called an S-box that contains a permutation of all possible 256 8-bit values. Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value.

2. Shift Rows



The **forward shift row transformation**, called ShiftRows. The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed.

3. Mix Columns



The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column.

$$s'_{0,j} = (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j})$$
$$s'_{3,j} = (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})$$

Where the symbol $\bullet$ to indicate multiplication over the finite field GF(2^8) and $\oplus$ to indicate bitwise XOR, which corresponds to addition in GF(2^8).

Multiplication Rule:

multiplication of a value by *x* (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1.

The following is an example of MixColumns

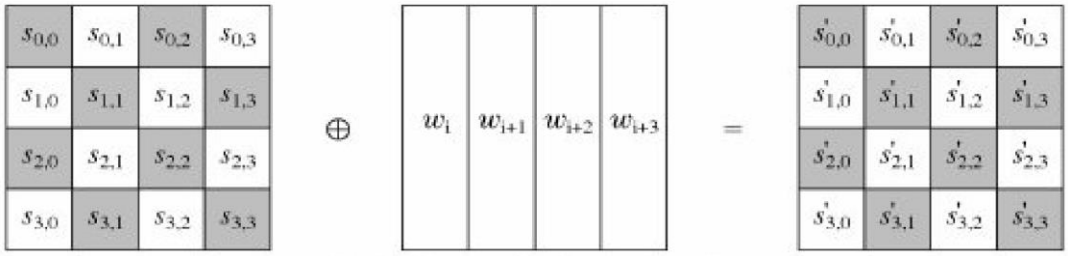
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

({02} · {87})⊕({03} · {6E}) ⊕ {46}⊕{A6}= {47}

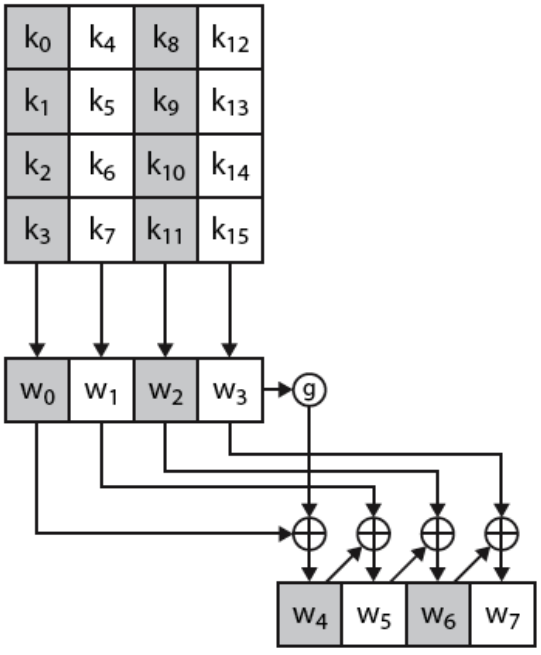
4. AddRoundKey



In the **forward add round key transformation**, called AddRoundKey, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column wise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation.

AES Key Expansion

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.



The following pseudocode describes the expansion:

```
KeyExpansion (byte key[16], word w[44])
{
        word temp
        for (i = 0; i < 4; i++)
```

```
        {
                w[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
        }
        for (i = 4; i < 44; i++)
        {
                temp = w[i-1];
                if (i mod 4 = 0)
                {
                        temp = SubWord (RotWord (temp)) ⊕ Rcon[i/4];
                        w[i] = w[i-4]≈temp
                }
        }
}
```

The generation of the first eight words of the expanded key, using the symbol g to represent that complex function. The function g consists of the following sub functions:

1. Rot Word performs a one byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. Sub Word performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.The round constant is different for each round and is defined as

Rcon[j]=(RC[j],0,0,0) with RC[1]=1,RC[j]=2·RC[j-1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are

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

Public-Key Cryptography and RSA

Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys, one a public key and one a private key. It is also known as public-key encryption.

Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext.

Asymmetric encryption can be used for confidentiality, authentication, or both.

The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption.

These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.
- Either of the two related keys can be used for encryption, with the other used for decryption

A public-key encryption scheme has six ingredients

1. **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.

3. **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
4. **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
5. **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

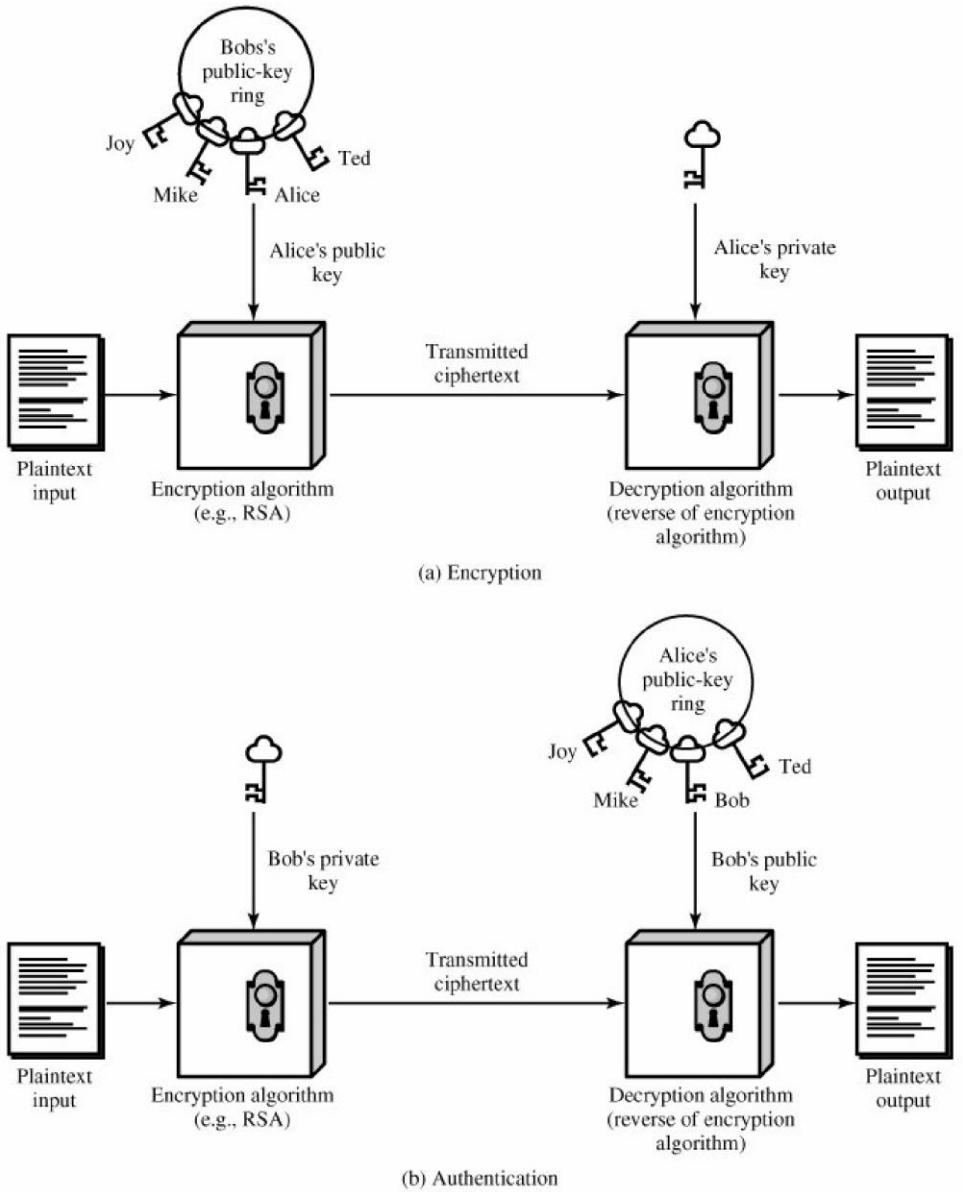


Fig: Public-Key Cryptography

Public-Key Cryptography provides

- **Confidentiality: $C = E_{KUb}(M)$; $M = D_{KRb}(C)$**
- **Authentication: $C = E_{KRa}(M)$; $M = D_{KUa}(C)$**

The following notation is used consistently throughout.

A secret key is represented by *Km*, where *m* is some modifier; for example, *Ka* is a secret key owned by user A.

A public key is represented by *PUa*, for user A, and the corresponding private key is *PRa*, Encryption of plaintext *X* can be performed with a secret key, a public key, or a private key, denoted by E(*Ka*, *X*), E(*PUa*, *X*), and E(*PRa*, *X*), respectively.

Similarly, decryption of ciphertext *C* can be performed with a secret key, a public key, or a private key, denoted by D(*Ka*, *X*), D(*PUa*, *X*), and D(*PRa*, *X*), respectively.

Conventional Encryption	Public-Key Encryption
The same algorithm with the same key is used for encryption and decryption.	One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.
The sender and receiver must share the algorithm and the key.	The sender and receiver must each have one of the matched pair of keys (not the same one).
The key must be kept secret.	One of the two keys must be kept secret.
It must be impossible or at least impractical to decipher a message if no other information is available. Cipher must be strong	It must be impossible or at least impractical to decipher a message if no other information is available. Cipher must be strong
Plaintext/ciphertext pairs must not weaken the security of the key	Plaintext/ciphertext pairs plus one of the keys must not weaken the other key

The RSA Algorithm

- It is developed by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978.
- The Rivest-Shamir-Adleman (RSA) scheme most widely used in general-purpose approach to public-key encryption.
- The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and n-1 for some n. A typical size for *n* is 1024 bits, or 309 decimal digits. That is, $n < 2^{1024}$

.

Description of the Algorithm

Key Generation

Select p, q	p and q both prime
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; \ 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

Steps:

1. Select 2 prime numbers p and q. $p \neq q$
2. Calculate n; n = p X q.
3. Calculate Euler's Totient function $\phi(n)$
   $\phi(n) = \phi(pq) = (p-1)(q-1)$; $\phi(n) < n$ and relatively prime to n
4. Select an integer (public key) e; such that e is relatively prime to $\phi(n)$ and $< \phi(n)$.
   ie, $1<n< \phi(n)$ and $GCD(\phi(n),e)=1$.
5. Determine d as a private key, by using Euclid's algorithm
   $ed \equiv 1 \bmod \phi(n)$
   $d = e^{-1} \bmod \phi(n)$
   this is equivalent to
   $ed = (k\ \phi(n)) + 1$
   $d = ((k\ \phi(n)) + 1) / e$
   where k = 1,2,3,… $\phi(n)$;
   $GCD(\phi(n), d) = 1$
6. The public key consists of KU = {*e*, *n*}.
   The private key consists of KR = {d,n}
7. Encryption
   Plaintext "M" and Ciphertext "C"
   $C = M^e \bmod n$
8. Decryption
   $M = C^d \bmod n$

Example: p=13,q=19,e=11,M=102

Step 1 : We pick 13 and 19.

Step 2 :Calculate N = p*q. N = 13*19 = 247.

Step 3 : $\phi(n)$ = (p-1)*(q-1).
For my p and q, (p-1)*(q-1) = 12*18 = 216.

Step 4 : Choose e = 11.

Step 5. Find private key. Find a number d so the e*d = 1 mod (p-1)*(q-1). For me this means that 11*d = 1 mod 216. It turns out that d = 59 works.
I keep my d = 59 secret!

Step 6: public key {11,247} and private key {59,247}

Step 7 : m = 102. The sender encrypts m by the formula
c = $\mathbf{m}^e$ mod N. For this message, using my public key we need to compute **c** = 102^{11} mod 247
computation:
$102^1 = 102 \bmod 247$
$102^2 = 30 \bmod 247$
$102^4 = 30^2 = 159 \bmod 247$
$102^8 = 59^2 = 87 \bmod 247$
$102^{11} = 102^8 * 102^2 * 102^1 = 87 * 30 * 102 = 201 \bmod 247$
The encrypted message sent to me is C = 201.

Step 8 : Use the formula **m** = $\mathbf{c}^d$ mod N. I need to find **m** = 201^{59} mod 247
computation:
$201^1 = 201 \bmod 247$
$201^2 = 140 \bmod 247$
$201^4 = 140^2 = 87 \bmod 247$
$201^8 = 87^2 = 159 \bmod 247$
$201^{16} = 159^2 = 87 \bmod 247$
$201^{32} = 87^2 = 159 \bmod 247$

$201^{59} = 201^{32} * 201^{16} * 201^{8} * 201^{2} * 201^{1} = 159 * 87 * 159 * 140 * 201 \bmod 247$

M= 102 mod 247, so my message is 102.

Advantages of RSA:

- Cryptanalysis is not possible for large values of 'n'
  - It would take more than 70 years to find p and q, if n is a 100 digit number.

Attacks on RSA Algorithm

Four possible approaches to attacking the RSA algorithm are as follows:

1. **Brute force:** This involves trying all possible private keys.
2. **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
3. **Timing attacks**: These depend on the running time of the decryption algorithm.
4. **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

1. Brute Force attack:

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, use a large key space. Thus, the larger the number of bits in *d*, the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

2. Mathematical attacks:

3 approaches:

- Factor n into its prime factors p and q (Factoring problem)
  - Factor n=p X q, hence allows calculation of $\phi(n)$, which allows determination of d. $d = e^{-1} \pmod{\phi(n)}$
- Determine $\phi(n)$ directly from n, without determining p and q and find d.
  - Equivalent to factoring
- Determine $d = e^{-1} \pmod{\phi(n)}$ directly, without first determining $\phi(n)$
  - Seems to be as hard as factoring

Avoid factoring problem

1. p and q should differ in length by only a few digits. Thus, for a 1024 bits key (309 decimal digits), both p and q should be on the order of magnitude of 10^{75} to 10^{100}.
2. Both (p-1) and (q-1) should contain a large prime factor.
3. gcd(p-1,q-1) should be small.

3. Timing Attacks

- Big integer multiplication take a long time
- The timing attack is a cipher text only attack.

Counter measures:

- Constant exponentiation time
  - Ensure that all exponentiations take the same amount of time, before returning the result.
  - Simple fix, but degrade the performance
- Random delay
  - Add a random delay to the exponentiation algorithm to confuse the timing attack
- Blinding
  - Multiply the cipher text by a random number before performing the exponentiation.
  - This process prevents the attacker from knowing what cipher text bits are being processed inside the computer and therefore prevents the bit by bit analysis to the timing attack.

4. Chosen ciphertext attacks

The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA). CCA is defined as an attack in which adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key.

Thus, the adversary could select a plaintext, encrypt it with the target's public key and then be able to get the plaintext back by having it decrypted with the private key.