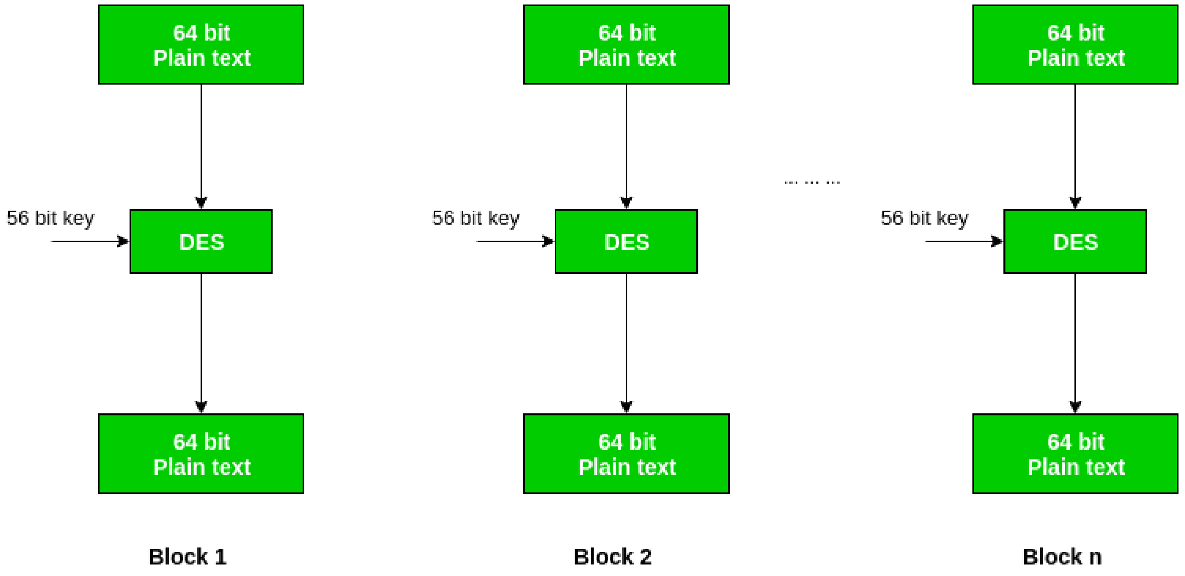
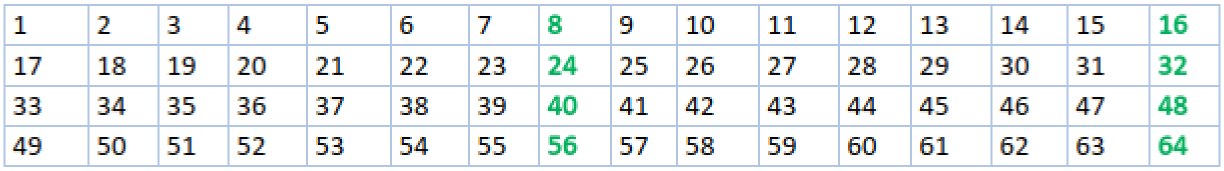
Data encryption standard (DES)

Data encryption standard (DES) has been found vulnerable against very powerful attacks and therefore, the popularity of DES has been found slightly on decline.

DES is a block cipher, and encrypts data in blocks of size of 64 bit each, means 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is show in figure.



We have mention that DES uses a 56 bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56 bit key. That is bit position 8, 16, 24, 32, 40, 48, 56 and 64 are discarded.



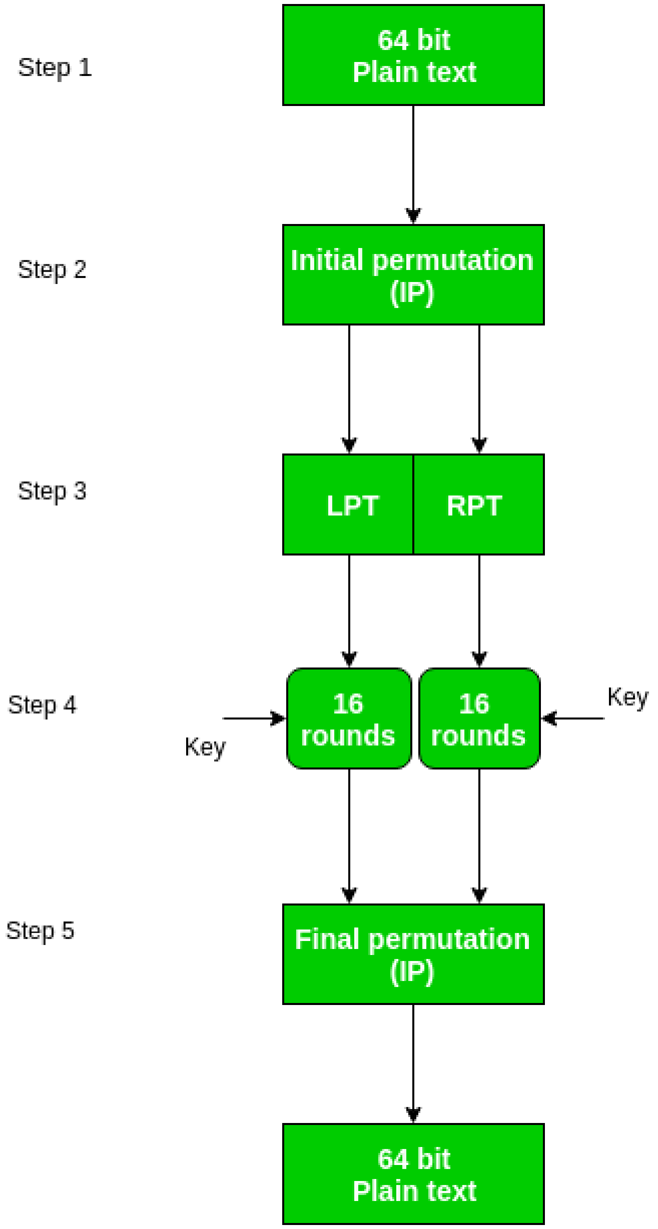
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Figure - discording of every 8th bit of original key

Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key.

DES is based on the two fundamental attributes of cryptography: substitution (also called as confusion) and transposition (also called as diffusion). DES consists of 16 steps, each of which is called as a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

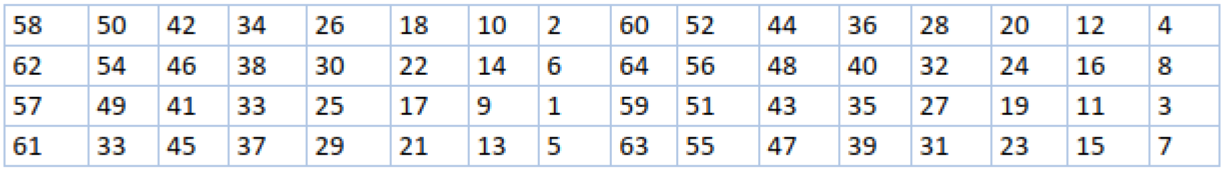
1. In the first step, the 64 bit plain text block is handed over to an initial Permutation (IP) function.
2. The initial permutation performed on plain text.
3. Next the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now each LPT and RPT to go through 16 rounds of encryption process.
5. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
6. The result of this process produces 64 bit cipher text.



Initial Permutation (IP) –
As we have noted, the Initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as show in figure.

For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block and so on.

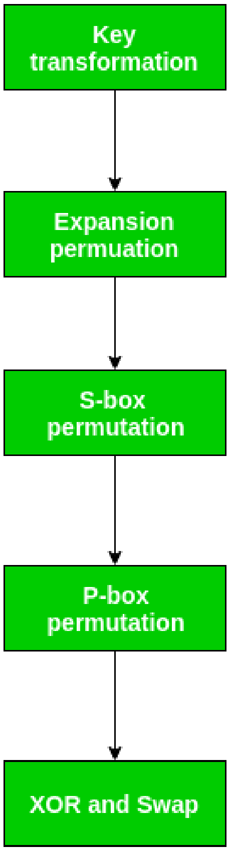
This is nothing but jugglery of bit positions of the original plain text block. The same rule applies for all the other bit positions which shows in the figure.



58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	33	45	37	29	21	13	5	63	55	47	39	31	23	15	7

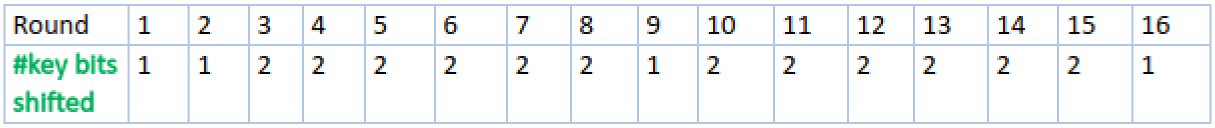
Figure - Initial permutation table

As we have noted after IP done, the resulting 64-bit permuted text block is divided into two half blocks. Each half block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad level steps outlined in figure.



Step-1: Key transformation –
We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called as key transformation. For this the 56 bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

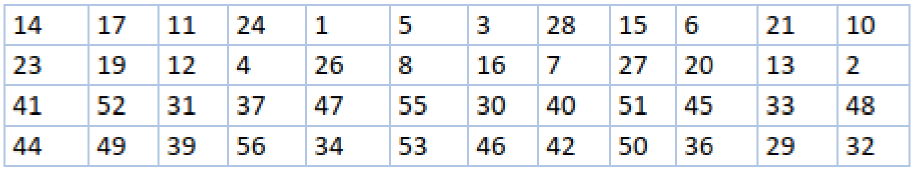
For example, if the round number 1, 2, 9 or 16 the shift is done by only position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is show in figure.



Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

After an appropriate shift, 48 of the 56 bit are selected. for selecting 48 of the 56 bits the table show in figure given below. For instance, after the shift, bit number 14 moves on the first position, bit number 17 moves on the second position and so on. If we observe the table carefully, we will realize that it contains only 48 bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as selection of a 48-bit sub set of the original 56-bit key it is called Compression Permutation.



14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Figure - compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That's make DES not easy to crack.

Step-2: Expansion Permutation –
Recall that after initial permutation, we had two 32-bit plain text areas called as Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called as expansion permutation. This happens as the 32 bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4 bit block of the previous step is then expanded to a corresponding 6 bit block, i.e., per 4 bit block, 2 more bits are added.

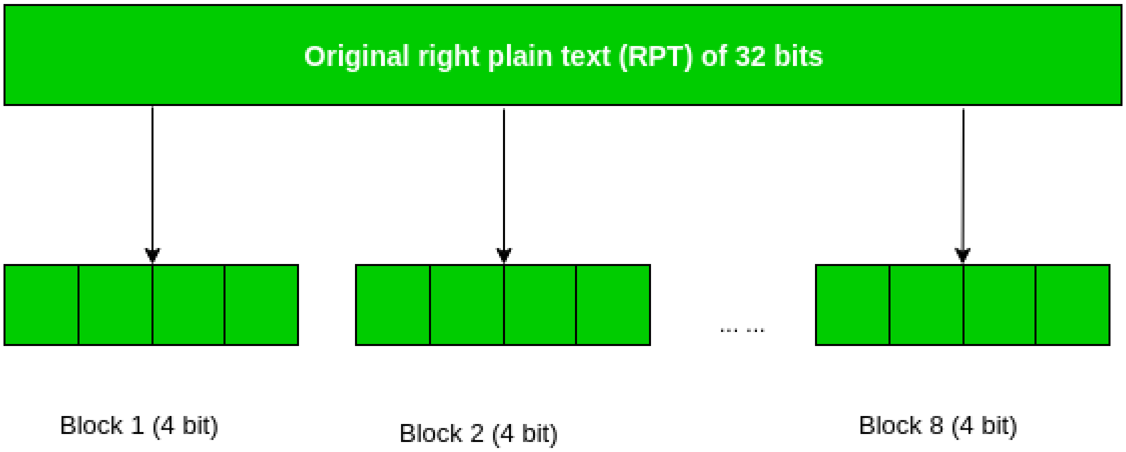


Figure - division of 32 bit RPT into 8 bit blocks

This process results into expansion as well as permutation of the input bit while creating output. Key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the 32-bit RPT to 48-bits. Now the 48-bit key is XOR with 48-bit RPT and resulting output is given to the next step, which is the S-Box substitution.

Iterated DES:

A block cipher that "iterates a fixed number of times of another block cipher, called round function, with a different key, called round key, for each iteration".

Most block ciphers are constructed by repeatedly applying a simpler function. This approach is known as iterated block cipher. Each iteration is termed a round, and the repeated function is termed the round function; anywhere between to 32 rounds are typical.

DES-X:

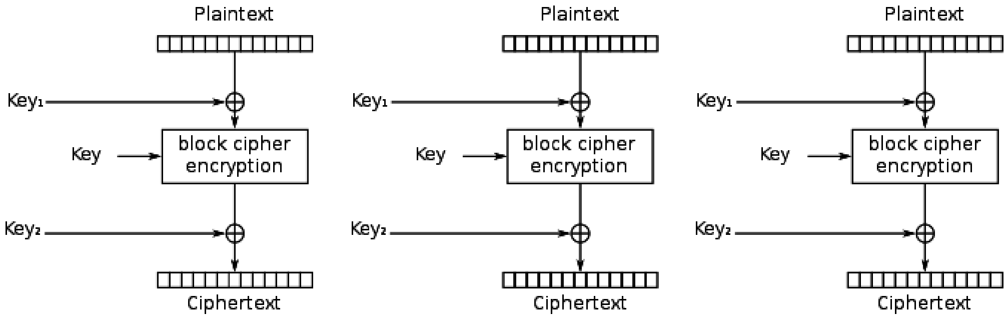
In cryptography, DES-X (or DESX) is a variant on the DES (Data Encryption Standard) block cipher intended to increase the complexity of a brute force attack using a technique called key whitening.

The algorithm was included in RSA Security's BSAFE cryptographic library since the late 1980s.DES-X augments DES by XORing an extra 64 bits of key (K1) to the plaintext before applying DES, and then XORing another 64 bits of key (K2) after the encryption:

The key size is thereby increased to 56 + 2 × 64 = 184 bits.

However, the effective key size (security) is only increased to 56+64-1- lg(M) =119 - lg(M) = ~119 bits, where M is the number of known plaintext/ciphertext pairs the adversary can obtain,and lg() denotes the binary logarithm. (Because of this, some implementations actually make K2 a strong one way function of K1 and K.)

DES-X also increases the strength of DES against differential cryptanalysis and linear cryptanalysis, although the improvement is much smaller than in the case of brute force attacks. It is estimated that differential cryptanalysis would require 261 chosen plaintexts (vs. 247 for DES), while linear cryptanalysis would require 260 known plaintexts (vs. 243 for DES.) Note that with 264 plaintexts (known or chosen being the same in this case), DES (or indeed any other block cipher with a 64 bit block size) is totally broken via the elementary codebook attack.



Xor Encrypt Xor (XEX) mode encryption

Advanced Encryption Standard (AES):

In cryptography, the Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used worldwide, as was the case with its predecessor, the Data Encryption Standard (DES).

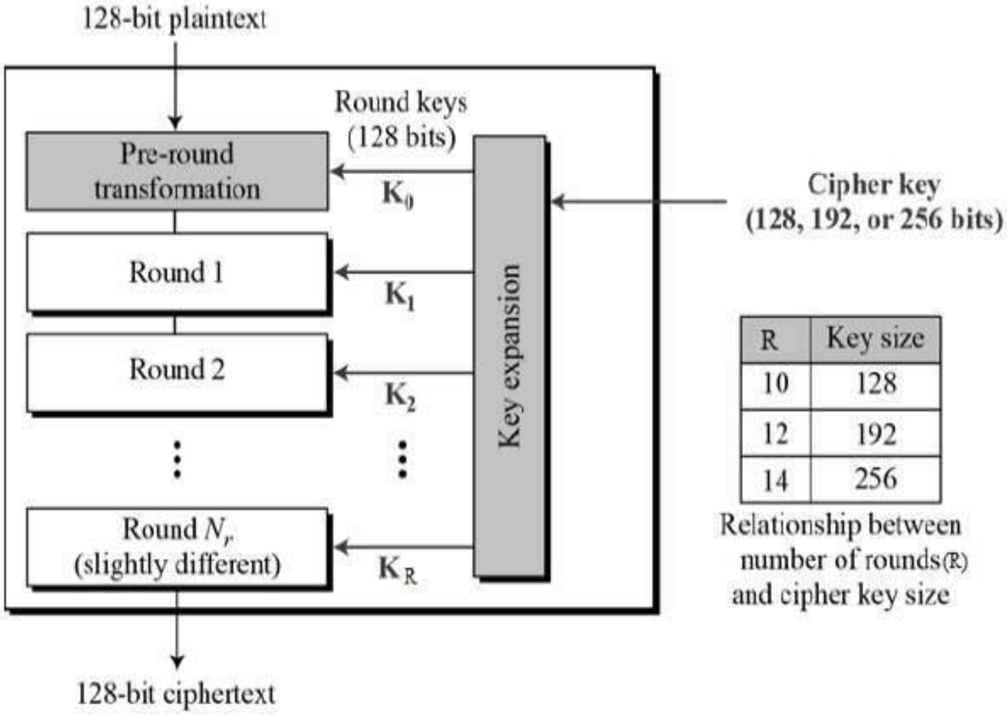
AES is one of the most popular algorithms used in symmetric key cryptography. It is available by choice in many different encryption packages. This marks the first time that the public has had access to a cipher approved by NSA for top secret information.

AES is fast in both software and hardware, is relatively easy to implement, and requires little memory. As a new encryption standard, it is currently being deployed on a large scale.

Operation of AES

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

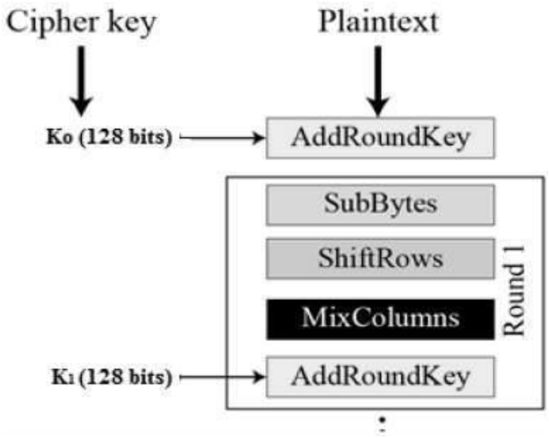
The schematic of AES structure is given in the following illustration −



R	Key size
10	128
12	192
14	256

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

Pseudorandom Function

In cryptography, a pseudorandom function family, abbreviated PRF, is a collection of efficiently-computable functions which emulate a random oracle in the following way: No efficient algorithm can distinguish (with significant advantage) between a function chosen randomly from the PRF family and a random oracle (a function whose outputs are fixed completely at random). Pseudorandom functions are vital tools in the construction of cryptographic primitives, especially secure encryption schemes.

Pseudorandom functions are efficient and deterministic functions which return pseudorandom output indistinguishable from random sequences. They are made based on pseudorandom generators but contrary to them, in addition to the internal state, they can accept any input data. The input may be arbitrary but the output must always look completely random.

A pseudorandom function, which output is indistinguishable from random sequences, is called a secure one.

Definition: The pseudorandom function (PRF) defined over (K, X, Y) is an efficient and deterministic function which returns a pseudorandom output sequence:

F: K x X -> Y

A pseudorandom function family can be constructed from any pseudorandom generator, using, for example, the construction given by Goldreich, Goldwasser, and Micali.

Psedorandom Permutations

In cryptography, a pseudorandom permutation, abbreviated PRP, is an idealized block cipher. It means the cipher that cannot be distinguished from a random permutation (that is, a permutation selected at random with uniform probability, from the family of all permutations on blocks of that size) with less computational effort than specified by the cipher's security parameters (this usually means the effort required should be about the same as a brute force search through the cipher's key space). If a distinguishing algorithm exists that achieves significant advantage with less effort than the security parameter specifies, the cipher is considered broken at least in a certificational sense, even if such a break doesn't immediately lead to a practical security failure.

Pseudorandom permutations can be defined in a similar way. They create output data indistinguishable from random sequences.

Definition: The pseudorandom permutation (PRP) defined over (K, X) is an efficient and deterministic function which returns a pseudorandom output sequence:

E: K x X -> X

- the function E (k, .) in one-to-one
- there exists an efficient inversion algorithm D (k, x)

Pseudorandom permutations which produce output sequences that are indistinguishable from random sequences, are called secure PRPs. It can be proved that secure PRP defined over large enough X is also a secure PRF (according to the PRF Switching Lemma).

Birthday attack

A birthday attack is a type of cryptographic attack that exploits the mathematics behind the birthday problem in probability theory. This attack can be used to abuse communication between two or more parties. The attack depends on the higher likelihood of collisions found between random attack attempts and a fixed degree of permutations (pigeonholes), as described in the birthday problem/paradox.

Mathematics

Given a function f, the goal of the attack is to find two inputs x1, x2 such that f(x1) = f(x2). Such a pair x1, x2 is called a collision. The method used to find a collision is to simply evaluate the function f for different input values that may be chosen randomly or pseudorandomly until the same result is found more than once. Because of the birthday paradox this method can be rather efficient. Specifically, if a function f(x) yields any of H different outputs with equal probability and H is sufficiently large, then we expect to obtain a pair of different arguments x1 and x2 with f(x1) = f(x2) after evaluating the function for about different arguments on average.

Understanding the problem

As an example, consider the scenario in which a teacher with a class of 30 students (n = 30) asks for everybody's birthday (for simplicity, ignore leap years) to determine whether any two students have the same birthday (corresponding to a hash collision as described further). Intuitively, this chance may seem small. If the teacher picked a specific day (say, 16 September), then the chance that at least one student was born on that specific day is $1-(364/365)^{30}$, about 7.9%. However, counter-intuitively, the probability that at least one student has the same birthday as *any* other student on any day is around 70% (for n = 30), from the formula

$1-365!/((365-n)!.365^{n})$.

Number-Theoretic Primitives

Number theory is a source of several computational problems that serve as primitives in the design of cryptographic schemes. Asymmetric cryptography in particular relies on these primitives. As with other beasts that we have been calling “primitives,” these computational problems exhibit some intractability features, but by themselves do not solve any cryptographic problem directly relevant to a user security goal. But appropriately applied, they become useful to this end. In order to later effectively exploit them it is useful to first spend some time understanding them.

This understanding has two parts. The first is to provide precise definitions of the various problems and their measures of intractability. The second is to look at what is known or conjectured about the computational complexity of these problems.

There are two main classes of primitives. The first class relates to the discrete logarithm problem over appropriate groups, and the second to the factoring of composite integers.

10.1 Discrete logarithm related problems

Let G be a cyclic group and let g be a generator of G. Recall this means that $G = \{g^0, g^1, \ldots, g^{m-1}\}$, where $m = |G|$ is the order of G. The discrete logarithm function $\mathrm{DLog}_{G,g}: G \to \mathbf{Z}_m$ takes input a group element a and returns the unique $i \in \mathbf{Z}_m$ such that $a = g^i$. There are several computational problems related to this function that are used as primitives.

10.1.1 Informal descriptions of the problems

The computational problems we consider in this setting are summarized in Fig. 10.1. In all cases, we are considering an attacker that knows the group G and the generator g. It is given the quantities listed in the column labeled "given," and is trying to compute the quantities, or answer the question, listed in the column labeled "figure out."

The most basic problem is the discrete logarithm (DL) problem. Informally stated, the attacker is given as input some group element X, and must compute $\mathrm{DLog}_{G,g}(X)$. This problem is conjectured to be computationally intractable in suitable groups G.

Problem	**Given**	**Figure out**
Discrete logarithm (DL)	g^x	x
Computational Diffie-Hellman (CDH)	g^x, g^y	g^{xy}
Decisional Diffie-Hellman (DDH)	g^x, g^y, g^z	Is $z \equiv xy \pmod{\lvert G\rvert}$?

Figure 10.1: An informal description of three discrete logarithm related problems over a cyclic group G with generator g. For each problem we indicate the input to the attacker, and what the attacker must figure out to "win." The formal definitions are in the text.

One might imagine "encrypting" a message $x \in \mathbf{Z}_m$ by letting g^x be the ciphertext. An adversary wanting to recover x is then faced with solving the discrete logarithm problem to do so. However, as a form of encryption, this has the disadvantage of being non-functional, because an intended recipient, namely the person to whom the sender is trying to communicate x, is faced with the same task as the adversary in attempting to recover x.

The Diffie-Hellman (DH) problems first appeared in the context of secret key exchange. Suppose two parties want to agree on a key which should remain unknown to an eavesdropping adversary. The first party picks $x \stackrel{\$}{\leftarrow} \mathbf{Z}_m$ and sends $X = g^x$ to the second party; the second party correspondingly picks $y \stackrel{\$}{\leftarrow} \mathbf{Z}_m$ and sends $Y = g^y$ to the first party. The quantity g^{xy} is called the DH-key corresponding to X, Y. We note that

$$Y^x = g^{xy} = X^y . \tag{10.1}$$

Thus the first party, knowing Y, x, can compute the DH key, as can the second party, knowing X, y. The adversary sees X, Y, so to recover the DH-key the adversary must solve the Computational Diffie-Hellman (CDH) problem, namely compute g^{xy} given $X = g^x$ and $Y = g^y$. Similarly, we will see later a simple asymmetric encryption scheme, based on Equation (10.1), where recovery of the encrypted message corresponds to solving the CDH problem.

The obvious route to solving the CDH problem is to try to compute the discrete logarithm of either X or Y and then use Equation (10.1) to obtain the DH key. However, there might be other routes that do not involve computing discrete logarithms, which is why CDH is singled out as a computational problem in its own right. This problem appears to be computationally intractable in a variety of groups.

We have seen before that security of a cryptographic scheme typically demands much more than merely the computational intractability of recovery of some underlying key. The computational intractability of the CDH problem turns out to be insufficient to guarantee the security of many schemes based on DH keys, including the secret key exchange protocol and encryption scheme mentioned above. The Decisional Diffie-Hellman (DDH) problem provides the adversary with a task that can be no harder, but possibly easier, than solving the CDH problem, namely to tell whether or not a given group element Z is the DH key corresponding to given group elements X, Y. This problem too appears to be computationally intractable in appropriate groups.

We now proceed to define the problems more formally. Having done that we will provide more specific discussions about their hardness in various different groups and their relations to each other.

10.1.2 The discrete logarithm problem

The description of the discrete logarithm problem given above was that the adversary is given as input some group element X, and is considered successful if it can output $\mathrm{DLog}_{G,g}(X)$. We would like to associate to a specific adversary A some advantage function measuring how well it does in solving this problem. The measure adopted is to look at the fraction of group elements for which the adversary is able to compute the discrete logarithm. In other words, we imagine the group element X given to the adversary as being drawn at random.

Definition 10.1.1 Let G be a cyclic group of order m, let g be a generator of G, and let A be an algorithm that returns an integer in $\mathbf{Z}_m$. We consider the following experiment:

```
Experiment $\mathbf{Exp}^{\mathrm{dl}}_{G,g}(A)$
    $x \stackrel{\$}{\leftarrow} \mathbf{Z}_m$ ; $X \leftarrow g^x$
    $\overline{x} \leftarrow A(X)$
    If $g^{\overline{x}} = X$ then return 1 else return 0
```

The *dl-advantage* of A is defined as

$$\mathbf{Adv}^{\mathrm{dl}}_{G,g}(A) \quad = \quad \Pr\left[\mathbf{Exp}^{\mathrm{dl}}_{G,g}(A) = 1\right] \; .$$ ▮

Recall that the discrete exponentiation function takes input $i \in \mathbf{Z}_m$ and returns the group element g^i. The discrete logarithm function is the inverse of the discrete exponentiation function. The definition above simply measures the one-wayness of the discrete exponentiation function according to the standard definition of one-way function. It is to emphasize this that certain parts of the experiment are written the way they are.

The discrete logarithm problem is said to hard in G if the dl-advantage of any adversary of reasonable resources is small. Resources here means the time-complexity of the adversary, which includes its code size as usual.

10.1.3 The Computational Diffie-Hellman problem

As above, the transition from the informal description to the formal definition involves considering the group elements X, Y to be drawn at random.

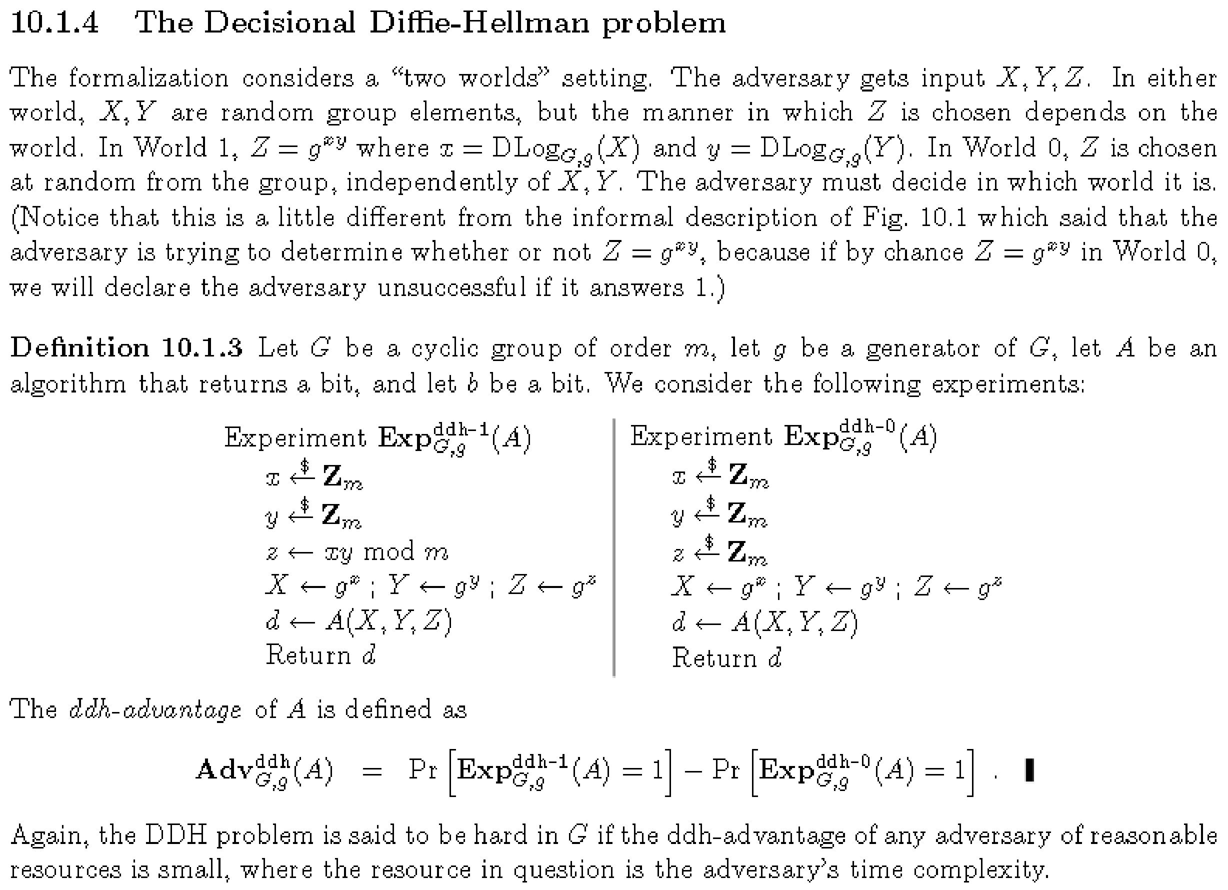
Definition 10.1.2 Let G be a cyclic group of order m, let g be a generator of G, and let A be an algorithm that returns an element of G. We consider the following experiment:

```
Experiment $\mathbf{Exp}^{\mathrm{cdh}}_{G,g}(A)$
    $x \stackrel{\$}{\leftarrow} \mathbf{Z}_m$ ; $y \stackrel{\$}{\leftarrow} \mathbf{Z}_m$
    $X \leftarrow g^x$ ; $Y \leftarrow g^y$
    $Z \leftarrow A(X, Y)$
    If $Z = g^{xy}$ then return 1 else return 0
```

The *cdh-advantage* of A is defined as

$$\mathbf{Adv}^{\mathrm{cdh}}_{G,g}(A) \quad = \quad \Pr\left[\mathbf{Exp}^{\mathrm{cdh}}_{G,g}(A) = 1\right] \; .$$ ▮

Again, the CDH problem is said to be hard in G if the cdh-advantage of any adversary of reasonable resources is small, where the resource in question is the adversary's time complexity.



10.1.4 The Decisional Diffie-Hellman problem

The formalization considers a "two worlds" setting. The adversary gets input X, Y, Z. In either world, X, Y are random group elements, but the manner in which Z is chosen depends on the world. In World 1, $Z = g^{xy}$ where $x = \mathrm{DLog}_{G,g}(X)$ and $y = \mathrm{DLog}_{G,g}(Y)$. In World 0, Z is chosen at random from the group, independently of X, Y. The adversary must decide in which world it is. (Notice that this is a little different from the informal description of Fig. 10.1 which said that the adversary is trying to determine whether or not $Z = g^{xy}$, because if by chance $Z = g^{xy}$ in World 0, we will declare the adversary unsuccessful if it answers 1.)

Definition 10.1.3 Let G be a cyclic group of order m, let g be a generator of G, let A be an algorithm that returns a bit, and let b be a bit. We consider the following experiments:

```
Experiment $\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A)$
    $x \xleftarrow{\$} \mathbf{Z}_m$
    $y \xleftarrow{\$} \mathbf{Z}_m$
    $z \leftarrow xy \bmod m$
    $X \leftarrow g^x$ ; $Y \leftarrow g^y$ ; $Z \leftarrow g^z$
    $d \leftarrow A(X, Y, Z)$
    Return $d$
```

```
Experiment $\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A)$
    $x \xleftarrow{\$} \mathbf{Z}_m$
    $y \xleftarrow{\$} \mathbf{Z}_m$
    $z \xleftarrow{\$} \mathbf{Z}_m$
    $X \leftarrow g^x$ ; $Y \leftarrow g^y$ ; $Z \leftarrow g^z$
    $d \leftarrow A(X, Y, Z)$
    Return $d$
```

The *ddh-advantage* of A is defined as

$$\mathbf{Adv}^{\text{ddh}}_{G,g}(A) \;=\; \Pr\left[\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A) = 1\right] . \quad ▮$$

Again, the DDH problem is said to be hard in G if the ddh-advantage of any adversary of reasonable resources is small, where the resource in question is the adversary's time complexity.

Fermat's Little Theorem

Fermat's little theorem is a fundamental theorem in elementary number theory, which helps compute powers of integers modulo prime numbers. It is a special case of Euler's theorem, and is important in applications of elementary number theory, including primality testing and public-key cryptography.

The result is called Fermat's "little theorem" in order to distinguish it from Fermat's last theorem.

Fermat's little theorem

Fermat's little theorem states that if p is a prime number, then for any integer a, the number

a^p –a is an integer multiple of p.

Here p is a prime number
$a^p \equiv a \pmod{p}$.

Special Case: If a is not divisible by p, Fermat’s little theorem is equivalent to the statement that a^{p-1}-1 is an integer multiple of p.

$a^{p-1} \equiv 1$ (mod p)
OR
a^{p-1} % p = 1
Here a is not divisible by p.

Take an Example How Fermat’s little theorem works
Examples:

P = an integer Prime number
a = an integer which is not multiple of P
Let a = 2 and P = 17

According to Fermat's little theorem
$2^{17-1} \equiv 1$ mod(17)
we got 65536 % 17 ≡ 1
that mean (65536-1) is an multiple of 17

Use of Fermat’s little theorem

If we know m is prime, then we can also use Fermats’s little theorem to find the inverse.

$a^{m-1} \equiv 1$ (mod m)
If we multiply both sides with a-1, we get

$a^{-1} \equiv a^{m-2}$ (mod m)

Euler's Theorem

The generalization of Fermat’s theorem is known as Euler’s theorem. In general, Euler’s theorem states that, “if *p* and *q* are relatively prime, then $p^{\varphi(q)} = 1(\mathrm{mod}\, q)$”, where *φ* is Euler’s totient function for integers. That is, $\varphi(q)$is the number of non-negative numbers that are less than *q* and relatively prime to *q*.

Proof of Euler’s theorem:
Consider the set of non-negative numbers,
$P = \{n_1, n_2, n_3, \cdots, n_{\varphi(q)}\}(\mathrm{mod}\, q)$
These elements are relatively (co-prime) to *q*.

Consider another set of non-negative numbers,

$$P_1 = \{pn_1, pn_2, pn_3, \cdots, pn_{\varphi(q)}\} (\bmod q) \text{ where } (p,q) = 1$$

Since the sets are congruent to each other,

$$n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)} \equiv pn_1 \cdot pn_2 \cdot pn_3 \cdots pn_{\varphi(q)} (\bmod q)$$

$$n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)} \equiv p\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right) (\bmod q)$$

$$p^{\varphi(q)} \cdot \left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right) \equiv \left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right) (\bmod q)$$

Since the set of numbers are relatively prime to *q*, dividing by the term $\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)$is permissible.

$$\frac{p^{\varphi(q)} \cdot \left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)}{\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)} \equiv \frac{\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)}{\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)} (\bmod q)$$

$$p^{\varphi(q)} \equiv \frac{\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)}{\left(n_1 \cdot n_2 \cdot n_3 \cdots n_{\varphi(q)}\right)} (\bmod q)$$

$$p^{\varphi(q)} \equiv 1 (\bmod q)$$

Hence proved.

RSA Algorithm (Rivest-Shamir-Adleman)

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977 [RIVE78].

The RSA cryptosystem is the most widely-used public key cryptography algorithm in the world. It can be used to encrypt a message without the need to exchange a secret key separately.

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Let us learn the mechanism behind RSA algorithm :

>> **Generating Public Key :**

- Select two prime no's. Suppose **$P = 53$ and $Q = 59$**.
- Now First part of the Public key : **$n = P*Q = 3127$**.
- We also need a small exponent say **e** :
- But e Must be
  - An integer.
  - Not be a factor of n.
  - **$1 < e < \Phi(n)$** [$\Phi(n)$ is discussed below],
  - Let us now consider it to be equal to 3.
- Our Public Key is made of n and e

>> **Generating Private Key :**

- We need to calculate $\Phi(n)$:
- Such that **$\Phi(n) = (P-1)(Q-1)$**
- so, $\Phi(n) = 3016$
- Now calculate Private Key, **d** :
- **$d = (k*\Phi(n) + 1) / e$** for some integer k
- For $k = 2$, value of d is 2011.
- Now we are ready with our – Public Key ($n = 3127$ and $e = 3$) and Private Key($d = 2011$)
- Now we will encrypt **"HI"** :
- Convert letters to numbers : H = 8 and I = 9
- Thus **Encrypted Data $c = 89^e \text{ mod } n$**.
- Thus our Encrypted Data comes out to be 1394
- Now we will decrypt **1394** :
- **Decrypted Data = $c^d \text{ mod } n$**.
- Thus our Encrypted Data comes out to be 89
- **8 = H and I = 9 i.e. "HI".**