$O(\log(N)\log(M))$ like multiplication, though it is slower.

If $n = O(\log(N))$ and $p(n)$ is a polynomial, then an algorithm that runs in time $c^{p(n)}$ for some constant c is said to run in exponential time (in the length of N). So $O(N\log(N))$ and $O(\sqrt[3]{N})$ are exponential. The current running time for finding a factor of N is $k\sqrt[3]{\log(N)(\log\log(N))^2}$ which is slower than polynomial but faster than exponential. Factoring a 20 digit number using trial division (which is exponential) would take longer than the age of the universe. In 1996, a 130-digit RSA challenge number was factored in 500 MIPS years.

The set of problems whose solutions have polynomial time algorithms is called P. There's a large set of problems for which no known polynomial time algorithm exists for solving them (though you can check that a given solution is correct in polynomial time) called NP. Many of the solutions differ from each other by polynomial time algorithms. So if you could solve one in polynomial time, you could solve them all in polynomial time. It is known that, in terms of running times, P$\leq$ NP $\leq$ exponential.

One NP problem: find simultaneous solutions to a system of non-linear polynomial equations mod 2. Like $x_1x_2x_5 + x_4x_3 + x_7 \equiv 0(\text{mod}2)$, $x_1x_9 + x_2 + x_4 \equiv 1(\text{mod}2)$, ... If you could solve this problem quickly you could crack DES quickly.

Another NP problem is the following: given a fixed, finite set of points in the plane, find the shortest path starting at one and going through each of the rest exactly once and returning to the original point.

DES

The U.S. government in the early 1970's wanted an encryption process on a small chip that would be widely used and safe. In 1973 and 1974 the National Bureau of Standards solicited data security systems from business and academia. I.B.M. submitted the Data Encryption Standard (DES) and it was accepted and published in 1975. DES is widely used in business in the United States: PIN numbers, phone conversations, bank transactions, and many other types of data are encrypted with DES. The DES algorithm is somewhat complicated to describe, for that reason, I have invented a similar, simpler algorithm I call Baby DES. I will first describe that, then I will explain who to expand all the parameters to get the real DES. In the cryptanalysis course we will apply linear and differential cryptanalysis to Baby DES for simplicity.

First Baby DES

You and your addressee have a shared 10 bit key. From that key you will make subkeys. You break your plaintext message into blocks of 8 bit binary numbers, like 10111101. There are then 2^8 possible plaintext blocks. Baby DES will encrypt one block at a time. Identical blocks will be encrypted identically.

Encryption is by $IP^{-1} \circ \Pi_{T_2} \circ \Theta \circ \Pi_{T_1} \circ IP$ which is the composition of 5 maps which will be described below. Recall, the above notation means that you do IP first, Π_{T_1} second, etc.

All additions are bit-by-bit mod 2 additions (XOR). So

$$\begin{array}{cc} & 10101 \\ + & 11001 \\ - & ----- \\ = & 01100 \end{array}$$

There are 3 kinds of maps:
i) IP is the initial permutation. It is (1,5,2,0,3,7,4,6); it is known. When I say known I mean that it is always the same and everybody knows what it is. Let $m_i \in \{0,1\}$. Then

$$\text{IP}(m_0m_1m_2m_3m_4m_5m_6m_7) = (m_1m_5m_2m_0m_3m_7m_4m_6) = (n_0n_1n_2n_3n_4n_5n_6n_7)$$

where $m_1 = n_0$, $m_5 = n_1$, $m_2 = n_2$, $m_0 = n_3 \ldots$. We have IP^{-1} =(3,0,2,4,6,1,7,5). So

$$\text{IP}^{-1}(n_0n_1n_2n_3n_4n_5n_6n_7) = (n_3n_0n_2n_4n_6n_1n_7n_5) = (m_0m_1m_2m_3m_4m_5m_6m_7)$$

ii) Θ switches the first four bits for the last four bits.

$$\Theta(m_0m_1m_2m_3m_4m_5m_6m_7) = (m_4m_5m_6m_7m_0m_1m_2m_3)$$

Note Θ^2 is the trivial map, so $\Theta^{-1} = \Theta$.
iii) Let's define Π_T, where T is some map (not necessarily one-to-one) from 4 bit binary numbers to 4 bit binary numbers.

$$\Pi_T(X, X') = (X + T(X'), X')$$

where X and X' are 4 bit binary numbers. Notice that Π_T^2 is the trivial map, because applying it twice is adding $T(X')$ twice to X and mod 2 that's adding 0000. Example: say you have (10111101) and T is some function for which $T(1101) = 1110$. Now 1011 + 1110 = 0101 so then $\Pi_T(10111101) = (01011101)$.

Decryption is by $(IP^{-1} \circ \Pi_{T_2} \circ \Theta \circ \Pi_{T_1} \circ IP)^{-1}$. Now recall that $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$. So decryption is by $IP^{-1} \circ \Pi_{T_1} \circ \Theta \circ \Pi_{T_2} \circ IP$.

Keys

Now the maps T_i are key-controlled so let's discuss how to make the two subkeys. Let's say that the agreed upon 10 bit key is $(r_0r_1 \ldots r_9)$ where $r_i \in \{0,1\}$. There are 2 known permutations: $P10$ = (2,4,1,6,3,9,0,8,7,5) and $P8$ = (5,2,6,3,7,4,9,8) and a shifting sequence (1,2).

First you apply $P10$ (which is only ever used once) to the key and get

$$(r_2r_4r_1r_6r_3r_9r_0r_8r_7r_5) = (s_0s_1s_2s_3s_4s_5s_6s_7s_8s_9)$$

Break this into two and shift each 5-tuple to the left 1 (since 1 is the first number in the shift sequence). So

$$(s_0s_1s_2s_3s_4)(s_5s_6s_7s_8s_9)$$

gets shifted to

$$(s_1 s_2 s_3 s_4 s_0 s_6 s_7 s_8 s_9 s_5) = (t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9)$$

Now apply P8 to pick out 8 of the 10 bits $(t_5 t_2 t_6 t_3 t_7 t_4 t_9 t_8)$. This is key 1.

Break the last 10 bit number into 2 pieces $(t_0 t_1 t_2 t_3 t_4)(t_5 t_6 t_7 t_8 t_9)$ and shift each left 2 (since 2 is the second number in the shift sequence).

$$(t_2 t_3 t_4 t_0 t_1 t_7 t_8 t_9 t_5 t_6) = (u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9)$$

Now apply P8 to pick out 8 of the 10 bits $(u_5 u_2 u_6 u_3 u_7 u_4 u_9 u_8)$. This is key 2.

The maps T_1 and T_2

We will begin with T_1. Take a 4 bit number $n_4 n_5 n_6 n_7$ with $n_i \in \{0, 1\}$. (I call them 4 through 7 because you apply T_i to bits 4 through 7 when doing Π_{T_i}). Make a diagram

$$\begin{array}{c|cc|c} n_7 & n_4 & n_5 & n_6 \\ n_5 & n_6 & n_7 & n_4 \end{array}$$

add key 1 (because we are doing T_1).

$$\begin{array}{c|cc|c} n_7 + t_5 & n_4 + t_2 & n_5 + t_6 & n_6 + t_3 \\ n_5 + t_7 & n_6 + t_4 & n_7 + t_9 & n_4 + t_8 \end{array}$$

we will rename these 8 bits (recall they are all 0's and 1's)

$$\begin{array}{c|cc|c} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \end{array}$$

There are two known S-boxes, $S[0]$ and $S[1]$, shown below. We have labelled the rows and columns 0 to 3.

$$S[0] = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{array}\right] \end{array} \qquad S[1] = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \left[\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{array}\right] \end{array}$$

and one known permutation P4=(1,3,2,0). Consider $(p_{00}p_{03})$ and $(p_{01}p_{02})$ as numbers between 0-3 (00=0, 01=1, 10=2, 11=3). In matrix S[0] look in row $(p_{00}p_{03})$ and column $(p_{01}p_{02})$ and find the entry, which is a number between 0-3. Write that number as a base 2 number $(q_0 q_1)$.

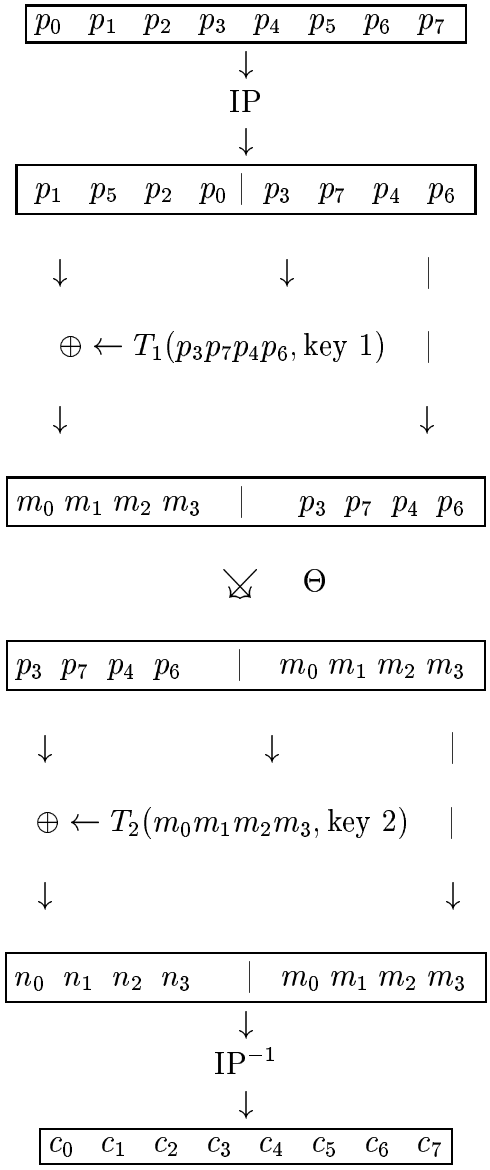
Similarly in matrix S[1] look in row $(p_{10}p_{13})$ and column $(p_{11}p_{12})$ and find the entry between 0-3. Write it as a base 2 number $(q_2 q_3)$. Now concatenate them and you have $(q_0 q_1 q_2 q_3)$, a 4 bit binary number. Apply P4 to it and get $(q_1 q_3 q_2 q_0)$. That's it.

So $T_1(n_4 n_5 n_6 n_7) = (q_1 q_3 q_2 q_0)$. Recall that this is just part of doing Π_{T_1}. So during encryption, if after the initial permutation, the message is now $(n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7)$, then Π_{T_1} will turn that into

$$(n_0 + q_1, n_1 + q_3, n_2 + q_2, n_3 + q_0, n_4, n_5, n_6, n_7)$$

T_2 is identical except that you use key 2. The S[0], S[1] and P4 are the same. It may seem odd to leave the last 4 bits alone, but Θ comes next and then Π_{T_2}.

Review



The real DES

In real life the blocks of plaintext are 64 bits long and so there are 2^{64} possible plaintext blocks. The encryption is actually by

$$IP^{-1} \circ \Pi_{T_{16}} \circ \Theta \circ \Pi_{T_{15}} \circ \Theta \circ \ldots \circ \Theta \circ \Pi_{T_1} \circ IP$$

The key is 56 bits long but comes with 8 parity-check bits. The subkeys have 48 bits. So instead of P10 and P8 there is P56 and P48. There are 16 subkeys since there are 16 T_i's. The shift sequence is actually $(1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)$. The initial permutation

IP is a permutation of the 64 bits. Now instead of T_i acting on $(n_4 n_5 n_6 n_7)$ it really acts on $(n_{32} \dots n_{63})$. The diagrams that you put those in actually look like

$$\begin{array}{c|cccc|c} n_{63} & n_{32} & n_{33} & n_{34} & n_{35} & n_{36} \\ n_{35} & n_{36} & n_{37} & n_{38} & n_{39} & n_{40} \\ \vdots & & \vdots & & & \vdots \\ n_{59} & n_{60} & n_{61} & n_{62} & n_{63} & n_{32} \end{array}$$

which has 8 rows and 6 columns (hence the 48 bit subkeys). Then there must be 8 S-boxes S[0], ...,S[7] (since there are 8 rows in the diagram) each having 4 rows and 16 columns (since $(n_{63} n_{36})$ can represent 4 numbers and $(n_{32} n_{33} n_{34} n_{35})$ can represent 16 numbers). In the real DES, each row of an S-box contains each of the numbers 0 through 15 exactly once. Also it has a P32 not a P4 (half the message length). Θ and IP are permutations and the Π_{T_i}'s are substitutions so DES is a product cipher.

Analysis of Baby DES The enemy intercepts a matched plaintext/ciphertext pair and wants to solve for the key. Let's say the plaintext is $P_0, \dots, P_7$, the ciphertext is $C_0, \dots, C_7$ and the key is $K_0, \dots, K_9$. There are 8 equations of the form

$$f_i(P_0, \dots, P_7, K_0, \dots K_9) = C_i$$

where f_i is a polyonomial in 18 variables, with coefficients in $\mathbf{F}_2$ which can be expected to have 2^{17} terms on average. Once we fix the C_i and P_i we get 8 non-linear equations in the 10 unknowns K_i. On average, the equations should have about 2^9 terms.

All of the permutations and additions are linear maps. The non-linearity comes from the S-boxes. Let us consider how they operate. For clarity, let us rename $(p_{00}, p_{01}, p_{02}, p_{03}) = (a, b, c, d)$ and $(p_{10}, p_{11}, p_{12}, p_{13}) = (w, x, y, z)$. Then the operation of the S-boxes can be computed with the following equations

$$\begin{aligned} q_0 &= abcd + ab + ac + b + d \\ q_1 &= abcd + abd + ab + ac + ad + a + c + 1 \\ q_2 &= wxyz + wxy + wyz + wy + wz + yz + w + x + z \\ q_3 &= wxz + wyz + wz + xz + yz + w + y \end{aligned}$$

where all additions are modulo 2. Alternating the linear maps with these non-linear maps leads to very complicated polynomial expressions for the ciphertext bits.

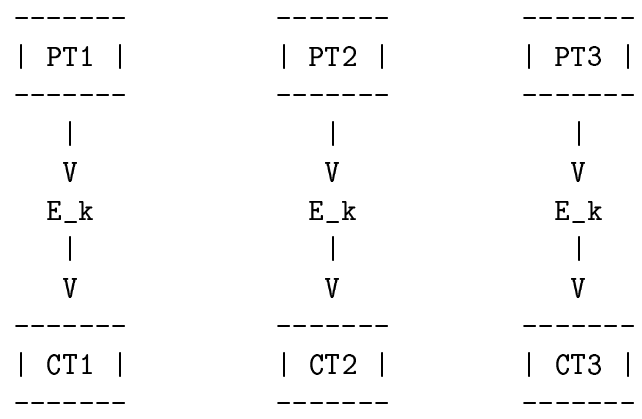
In the real DES, a pair $\Theta \circ \Pi$ is called a round. After 5 rounds, every (partial) ciphertext bit depends on every plaintext bit. Solving many non-linear equations in many unknowns over $\mathbf{F}_2$ is a problem in NP.

More complicated ways of using DES

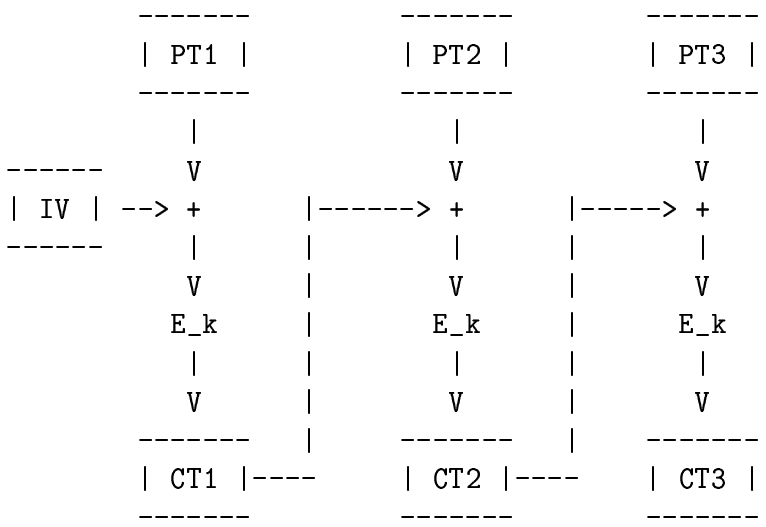
Many consider the key to be too short now. In 1997, a group of users on the Internet tried all possible DES keys on a challenge PT/CT pair (from RSA). In 1997, one expects to be able to exhaustively try different keys on a one million dollar machine until reaching the right one in under two hours. It turns out that using DES twice, one after the other, with

two different keys is not much safer than single DES with 1 key. Nowadays many use triple DES with 2 keys. Let E_k denote encrypting with DES and key k. Let D_k denote decrypting with DES and key k. Triple DES is $CT = E_{\text{key 1}}(D_{\text{key 2}}(E_{\text{key 1}}(PT)))$.

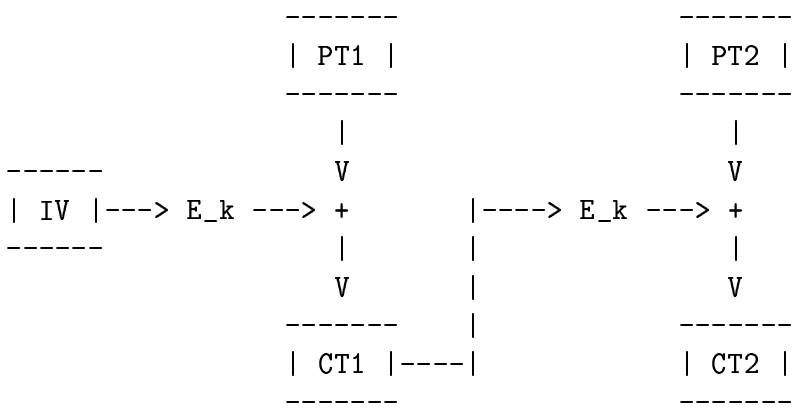
There are four modes on a DES chip. The standard mode is the *electronic code book* (ECB) mode. It is the most straightforward but has the disadvantage that for a given key, two indentical plaintexts will correspond to identical ciphertexts.



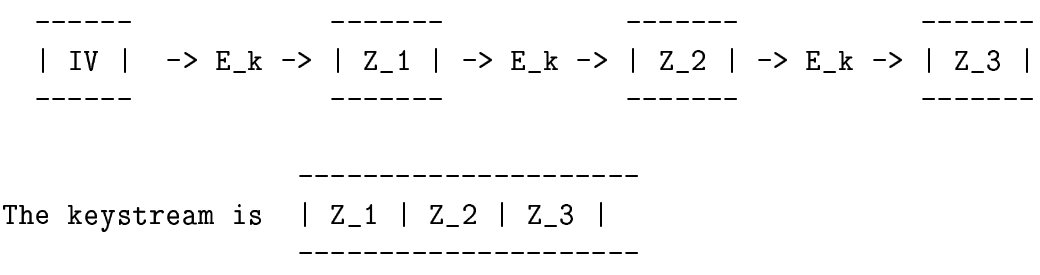
The next mode is the *cipherblock chaining* (CBC) mode. IV denotes an initialization vector. It is a random 64-bit string that the two users must agree upon ahead of time.



The next mode is the *cipher feedback* (CFB) mode. IV again denotes a 64-bit initialization vector that the two users must agree upon ahead of time.



The last mode is the *output feedback* (OFB) mode. This is a modern stream cipher. You XOR (sum mod 2) the PT bitstream with the keystream to get the CT bitstream. Below is how you create the keystream. IV again denotes a 64-bit initialization vector that the two users must agree upon ahead of time.



Public Key Cryptography

In a secret key cryptosystem, if you know the enciphering transformation and the enciphering key you can find the deciphering transformation and key very quickly (polynomial time). This is true with $C \equiv aP + b(\text{mod}26)$, modern stream ciphers and DES.

Public key cryptography A cryptosystem where everyone knows the enciphering transformation and everyone's enciphering key but no known polynomial time algorithm will get deciphering keys from those.

One way function $f : X \to Y$. Given $x \in X$, it is easy to compute $f(x)$. Given $y \in Y$ it is hard to find x such that $f(x) = y$. So it is hard to compute f^{-1}, which might be the deciphering transformation.

Trapdoor function A one way function where computing f^{-1} is fast, when known.

Often, to store a password, there is a file with f(password) where f is a one-way function. You log in, enter your password and the computer finds f(password) and compares with that file.

RSA

Recall that if $a \equiv 1(\text{mod}\phi(n))$ and $\gcd(m, n) = 1$ then $m^a \equiv m(\text{mod}n)$.