# DES and and the Simplified DES Algorithm

Justin Wyss-Gallifent

March 14, 2022

## 0.1 Introduction

The DES (Data Encryption Standard) was developed in the 1970s at IBM and adopted by the US in 1977. It was then adopted worldwide.

It had issues from the outset including concerns about NSA backdoors and the short key length. Originally IBM wanted 64 bits and the NSA wanted 48 and they settled on 56. Really it's a 64-bit key with 8 bits used for parity checking.

In the 1990s DES started to be cracked easily by brute force and various cryptanalysis techniques and was effectively thrown out and replaced by AES (Advanced Encryption Standard).

There are other versions of DES including one called Triple-DES (3DES) which involves three keys. They are all certainly more secure than DES but are still problematic. All these versions were officially retired in about 2018.

## 0.2 Overview

At a high level DES works as follows:

1. The key is used to generate 16 subkeys $K1$ through $K16$.
2. The input is split in half to $L$ and $R$.
3. $R$ is altered via $K1$ and permutations.
4. The result is XORed with $L$ and then joined with $R$ to create the output.
5. Steps 2,3,4 are repeated with $K2$ through $K16$ with the output as the new input.
6. To decrypt we use the exact same process but with the keys in reverse order.

## 0.3 Simplified DES

Simplified DES, or SDES, was developed by Edward Schaefer at Santa Clara University to help understand the basic structure of DES. This is a simplified version which has the major components of DES. Note that there are other simplified DES versions around, including one in Trappe and Washington's book, but the one presented here is the most popular.

## 0.4 Key

The key $K$ is a 10-bit number.

## 0.5 Input/Output

The input/output are bytes.

## 0.6 Permutation Notation Note

The permutation notation here is a bit non-standard and should be thought of as a selection function. If we apply the permutation $(2, 4, 3, 1)$ to 1011 it means we select the bits in positions 2,4,3,1 in that order, yielding 0111. This allows us to select bits multiple times, so applying the expansion permutation $(2, 4, 3, 1, 2, 4)$ to 1010 would yield 001100. Similarly we can apply reduction permutations.

## 0.7 Subkey Generation

1. Take the original key $K$ and apply the permutation $P10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)$.
2. Split in half and left-rotate each half by 1 bit.
3. Join and apply the reduction permutation $P8 = (6, 3, 7, 4, 8, 5, 10, 9)$ to get the subkey $K1$.
4. Go back after the previous left-rotate and left-rotate each half by another 2.
5. Join and apply the reduction permutation $P8 = (6, 3, 7, 4, 8, 5, 10, 9)$ to get the subkey $K2$.
6. The result is subkey $K2$.

**Example:** Suppose we have the key 1101001101.

1. We permute this to get 0011111010.
2. Split in half to 00111 and 11010 and left-rotate each half by 1 bit to get 01110 and 10101.
3. Join to 0111010101 and apply the reduction permutation $P8 = (6, 3, 7, 4, 8, 5, 10, 9)$ to get the subkey $K1 = 11011010$.
4. Go back to 01110 and 10101 and left-rotate each half by another 2 to get 11001 and 10110.
5. Join to 1100110110 and apply the reduction permutation $P8 = (6, 3, 7, 4, 8, 5, 10, 9)$ to get the subkey $K2 = 10001101$.

## 0.8 Encryption

1. Take the plaintext byte $P$ and apply the permutation $IP = (2, 6, 3, 1, 4, 8, 5, 7)$.
2. Split to nibbles $L$ and $R$ and apply the expansion permutation $EP = (4, 1, 2, 3, 2, 3, 4, 1)$ to $R$.
3. XOR this with $K1$.
4. Split to nibbles $n_1n_2n_3n_4$ and $n_5n_6n_7n_8$ and use the lookup tables, called S-boxes:

$$S0 = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 01 & 00 & 11 & 10 \\ 01 & 11 & 10 & 01 & 00 \\ 10 & 00 & 10 & 01 & 11 \\ 11 & 11 & 01 & 11 & 10 \end{array} \quad \text{and} \quad \text{S1} = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 00 & 01 & 10 & 11 \\ 01 & 10 & 00 & 01 & 11 \\ 10 & 11 & 00 & 01 & 00 \\ 11 & 10 & 01 & 00 & 11 \end{array}$$

   Find the entry in row $n_1n_4$ and column $n_2n_3$ in $S0$ and find the entry in row $n_5n_8$ and column $n_6n_7$ in $S1$. Join these.
5. Apply the permutation $P4 = (2, 4, 3, 1)$.
6. XOR this with $L$.
7. Join this on the left with $R$ on the right. This is the end of Round 1.
8. Swap the two nibbles.
9. Repeat steps 2-7 with $K2$. This is the end of Round 2.
10. Apply the permutation $IP^{-1} = (4, 1, 3, 5, 7, 2, 8, 6)$. The result is the ciphertext byte.

**Example:** Suppose we have the plaintext byte 1010 1010.

1. Apply $IP = (2, 6, 3, 1, 4, 8, 5, 7)$ to get 0011 0011.
2. Split to nibbles $L = 0011$ and $R = 0011$ and apply $EP = (4, 1, 2, 3, 2, 3, 4, 1)$ to $R = 0011$ to get 1001 0110.
3. XOR this with $K1 = 1101\ 1010$ to get 0100 1100.
4. Split to nibbles 0100 and 1100 and use $S1$ and $S2$ respectively to get 11 and 01. Join to get 1101.
5. Apply $P4 = (2, 4, 3, 1)$ to get 1101.
6. XOR this with $L = 0011$ to get 1110.
7. Join this on the left with $R = 0011$ on the right to get 1110 0011. This is the end of Round 1.
8. Swap the two nibbles to get 0011 1110.
9. Split to nibbles $L = 0011$ and $R = 1110$ and apply $EP = (4, 1, 2, 3, 2, 3, 4, 1)$ to $R = 1110$ to get 0111 1101.
10. XOR this with $K2 = 1000\ 1101$ to get 1111 0000.
11. Split to nibbles 1111 and 0000 and use $S1$ and $S2$ respectively to get 10 and 00. Join to get 1000.
12. Apply $P4 = (2, 4, 3, 1)$ to get 0001.
13. XOR this with $L = 0011$ to get 0010.
14. Join this on the left with $R = 1110$ on the right to get 0010 1110. This is the end of Round 2.
15. Apply $IP^{-1} = (4, 1, 3, 5, 7, 2, 8, 6)$ to get 0011 1001. This is the ciphertext byte.

## 0.9 Decryption

Decryption is exactly the same as Encryption except we switch the roles of $K2$ and $K1$.

**Example:** Suppose we have the ciphertext byte 0011 1001.

1. Apply $IP = (2, 6, 3, 1, 4, 8, 5, 7)$ to get 0010 1110.
2. Split to nibbles $L = 0010$ and $R = 1110$ and apply $EP = (4, 1, 2, 3, 2, 3, 4, 1)$ to $R = 1110$ to get 0111 1101.
3. XOR this with $K2 = 1000\ 1101$ to get 1111 0000.
4. Split to nibbles 1111 and 0000 and use $S1$ and $S2$ respectively to get 10 and 00. Join to get 1000.
5. Apply $P4 = (2, 4, 3, 1)$ to get 0001.
6. XOR this with $L = 0010$ to get 0011.
7. Join this on the left with $R = 1110$ on the right to get 0011 1110. This is the end of Round 1.
8. Swap the two nibbles to get 1110 0011.
9. Split to nibbles $L = 1110$ and $R = 0011$ and apply $EP = (4, 1, 2, 3, 2, 3, 4, 1)$ to $R = 0011$ to get 10010110.
10. XOR this with $K1 = 1101\ 1010$ to get 0100 1100.
11. Split to nibbles 0100 and 1100 and use $S1$ and $S2$ respectively to get 11 and 01. Join to get 1101.
12. Apply $P4 = (2, 4, 3, 1)$ to get 1101.
13. XOR this with $L = 1110$ to get 0011.
14. Join this on the left with $R = 0011$ on the right to get 00110011. This is the end of Round 2.
15. Apply $IP^{-1} = (4, 1, 3, 5, 7, 2, 8, 6)$ to get 1010 1010. This is the plaintext byte.

## 0.10 Notes and Comparison to DES

A few notes about DES and SDES.

1. DES uses sixteen rounds rather than just the two we see here in SDES.
2. Consequently in DES there are sixteen subkeys instead of just two but they are created by a similar *key schedule* which involves splits and left rotates.
3. The splitting, expanding, and XORing process in DES is more complicated with larger binary expressions.
4. There are eight S-boxes in DES rather than just two, and they are much larger.
5. In DES of course there are $2^{56}$ possible keys whereas in SDES there are only $2^{10} = 1024$.
6. It is not entirely clear what role $IP$ nad $IP^{-1}$ play either here or in DES. I asked some cryptographers and they don't know either. It's been suggested that permuting the initial data made it easier to load into older chips but it's not clear how this would be the case.
7. During the encryption/decryption process we have effectively encrypted $R$ and XORed it with $L$. This is a design model of many cryptographic systems including SDES, DES, and others. It is often called a *Feistel Cipher* but this name is confusing because it's not the whole cipher, it's really just a component. It turns out that this component allows an arbitrary number of rounds all of which are invertible.
8. The output is highly sensitive on the input in the sense that changing even one bit of the input will change several bits of the output. This makes it harder to do *differential cryptanalysis* which is a type of cryptanalysis whereby small changes are made to the input and the changes to the output are analyzed in order to gain insight into the underlying functioning.
9. The various permutations (especially the S-Boxes) make the encyption seem random. Of course it's not, but seeming random makes cryptanalysis trickier.
10. Brute force involves knowing one plaintext/ciphertext pair and trying every possible key. For $2^{56}$ keys this seems like a lot but is computationally manageable. Notice that for every extra bit added to the keyspace we double the number of possible keys, loosely doubling the computation time for a brute force attack.