Attacking Weak Symmetric Ciphers

HW1 - CNS Sapienza

Pietro Borrello

02 Nov 17

1 Introduction

Due to the continuous increasing of computational power, every encryption algorithm will eventually become obsolete. Any algorithm for which a bruteforce attack is computationally feasible, is considered weak, and must not be used. In this article we present an analysis on the bruteforce attack against such an algorithm as DES, for which we provide an implementation and performance results, where given a pair of plaintext and ciphertext, key will be discovered.

DES, first published in 1975 by IBM, provide symmetric encryption with a 56bits key. It has been considered obsolete since 1998, thanks to the EFF DES cracker [1], a 250'000$ computer that managed to broke a DES encrypted message in 56 hours exploiting the limited key space given by such a small key. Apart form bruteforce, more advanced techniques exist as Differential and Linear Cryptanalysis on DES [2] that can break the cipher in less than the complexity of the exhaustive search in the key space. Such attacks are not considered in this article, that has the aim to show that DES is breakable only with a dumb bruteforce attack. Thus providing the thesis that using DES should be foolish.

2 Resources Used

To test the implementation a 2 GHz Intel Core i7 (quad core, with 8 virtual cores thanks to hyper-threading) with 8 GB of RAM had been used.

The choice of the technology to implement the cracker was driven by the will to use *libgcrypt*. We consider this library the state of the art in cryptographic libraries, with continuous updates and bug fixes on it, giving us the confidence that all the operation would be implemented in the most efficient way. *Libgcrypt* is a C library developed as part of GnuPG, in the GNU Project, approved in Federal Information Processing Standard Publication 140-2 by NIST.

3 Implementation

As previously said, we based our implementation on *libgcrypt* [3] library, providing as a proof of concept a C program, that has the aim to crack a key of a given complexity, from the knowledge of a pair of plaintext and ciphertext. A DES key is composed by 56bits, split in 8 groups of 7bits, and for each group of bits, a parity bits is added, resulting in 8 groups of 8bits, maintained in a 64bit vector.

To make the approach scalable, we limited the key space to test the implementation on machines that are not so powerful. So we defined the complexity of the key for the cracker as the total number of unfixed bytes in the key vector. Thus $8 - COMPLEXITY$ provides the number of the bytes in the vector we fixed to zero, for testing purpose. For example a key of complexity 5 is:

x_0	x_1	x_2	x_3	x_4	0	0	0

and only the unknown bytes set to x_i have to be cracked.

Therefore a $COMPLEXITY$ set to 8 means a full DES crack. All the unfixed bytes are randomized, to provide a performance estimation on the average case and not only in the worst case, that would be obtained fixing the bytes at $0xff$.

The key enumeration in the bruteforce as to be made carefully: to be noticed is that not all the libraries (and neither *libgcrypt*) check parity bits, simplifying the code of the key enumeration. So a careless designer could think that the key enumeration could be simply done increasing a long integer index in a 64bit environment. This naive implementation would increase key space from 2^{56} to 2^{64}. Our implementation, considered not naive, is a simple reimplementation of carry addition considering only the most significant 7 bits out of 8. The snippet of code that follows describes the function incrementing the key:

```
int next_key(unsigned char key[8]) {
  unsigned char to_sum[8] = {2, 0, 0, 0, 0, 0, 0, 0};
  for (int i = 0; i < 8; i++) {
    unsigned char temp = key[i];
    key[i] += to_sum[i];
    if (key[i] < temp) {
      if (i < 7) to_sum[i + 1] += 2;
      else return -1;
    }
  }
  return 0;
}
```

The rest of the code consist simply in iterating in the key space, until a match for $plaintext == decrypt(key, cipher)$ is found. To speed up the key space search, we provide a multithread implementation, for which each of the 16 threads has to search a match in an independent key subspace.

4 Evaluation

Due to restriction on time and on the performance of the machine used, we were able to find a DES key up to a complexity of 6 out of 8, with 50 hours of computation. Therefore performance considerations and analysis have been possible up to a complexity of 5.

The program doesn't provide any pre-computation, nor rainbow tables, so space consumption is constant in respect to complexity, and linear in respect to plaintext size. Time cost is expected to be $O(n2^{7C})$ where n is the plaintext length and C is the complexity upper bounded by 8, that reduces to $O(n2^{56}) \simeq O(n)$ for complexity equals to 8.

We ran the program 10 times, with random key of complexity 5 and with a plaintext size of 40 bytes. Given the randomness of the key generation, each time the program ran, had a really different duration, depending on the distance from the starting point in the thread subspace. The sample set had a mean of $\mu = 58$ minutes and 20 seconds and a standard deviation $\sigma = 35$ minutes and 52 seconds. The minimum in the sample was 1 minute and 12 seconds, and the maximum was 124 minutes and 38 seconds. The worst case is represented by the key being all $0xff$ bytes, due to the order of exploration in the key space, and it took in average 155 minutes to be computed.

5 Conclusion

There is no surprise that DES can be cracked by bruteforce attack, but what should make the reader to think, is the fact that DES can be attacked just with the aid of a consumer laptop. Any key of such a small length, for similar algorithms has to be considered insecure, and never used.

References

[1] S. Landau, *Standing the test of time: The data encryption standard*, Notices of AMS, 2000.

[2] Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Full 16-Round DES*, Springer, 1993.

[3] W. Koch, M. Schulte, *The libgcrypt reference manual*, Free Software Foundation Inc, 2005.