Codes and Cryptography Summative

Module Name: Codes and Cryptography

Date: 08/12/22

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

1 Introduction

A feistel cipher is a variant of a block cipher that iteratively executes a function, referred to as the feistel function, on half of the plaintext. Due to the structure of the cipher, the feistel function does not need to be invertible to be able to encrypt and decrypt data. This allows for the designer of the cryptosystem to select a feistel function that will protect the system against popular block cipher attacks, such as meet-in-the-middle and linear cryptanalysis. Feistel functions have been used widely in different applications such as the DES cryptosystem.

This report will describe the process of designing and implementing an algorithm for cracking the feistel cipher. More specifically, the problem is laid out as the following: given a single plaintext ciphertext pair and a known number of rounds, is it possible to efficiently compute a key such that '*feistel(plaintext, key, rounds) == ciphertext*'? This problem is enclosed under the assumption that the feistel function used in the cipher is bitwise AND. Finally, this report will use the results found to evaluate whether bitwise AND is an appropriate choice of feistel function.

2 Method

The naïve approach to solving this problem would be by brute force. Given a 2t-bit input, the size of a given key is rt where r is the number of rounds supplied to the feistel function. This results in a total search space of *2^rt*, or on average *2^rt-1* keys to be searched. For large plaintext-ciphertext pairs, this is intractable.

The key insight that leads to the development of an improved algorithm lies within the properties of the feistel function used in the cipher. Bitwise AND is particularly vulnerable to a simplified version of differential cryptanalysis. It is entirely predictable how changes to the plaintext input will affect the ciphertext output, which can be used to determine additional information about the key. Feistel can be expressed as a sequence of two operations: the feistel function (bitwise AND) and bitwise XOR. Each function and their possible compositions, act independently on each bit of the input. A balanced feistel cipher will only perform these operations on one half of the input per-round before swapping the halves around. Therefore, changing a single bit in the plaintext will change at most two bits in the ciphertext. This enables the problem to be decomposed into a linear combination of sub-problems that can be concatenated to form a solution to the main problem in significantly less time than a brute force search.

To formalise this, let there exist a plaintext-ciphertext pair of a feistel cipher on r rounds, split into halves of length t. Label these values P_L, P_R, C_L and C_R where $P = (P_L, P_R)$ and $C = (C_L, C_R)$. It holds that there exists at least one key, K, of length rt such that:

$$feistel(P, K, r) = C \quad (1)$$

Let K_i denote the ith set of t bits in K for some $i \leq r$. Therefore, $K = (K_1, \ldots, K_{i-1}, K_i)$. Expressing P, C and K as tuples of t length elements yields the following:

$$feistel((P_L, P_R), (K_1, \ldots, K_{r-1}, K_r), r) = (C_L, C_R) \quad (2)$$

By applying the insight that feistel contains only bitwise operations and swapping P_L and P_R, we can

simplify the problem as follows. For the *xth* bit of each P_L, P_R, C_L, C_R, and all K_i, this equality holds true:

$$feistel((P_{L[x]}, P_{R[x]}), (K_{1[x]}, \ldots, K_{r-1[x]}, K_{r[x]}), r) = (C_{L[x]}, C_{R[x]}) \quad (3)$$

Therefore, the original problem (1) can be expressed as a concatenation of the set of *t* sub-problems:

$$feistel((P_{L[1]}, P_{R[1]}), (K_{1[1]}, \ldots, K_{r-1[1]}, K_{r[1]}), r) = (C_{L[1]}, C_{R[1]})$$

$$feistel((P_{L[2]}, P_{R[2]}), (K_{1[2]}, \ldots, K_{r-1[2]}, K_{r[2]}), r) = (C_{L[2]}, C_{R[2]})$$

$$\ldots$$

$$feistel((P_{L[t]}, P_{R[t]}), (K_{1[t]}, \ldots, K_{r-1[t]}, K_{r[t]}), r) = (C_{L[t]}, C_{R[t]}) \quad (4)$$

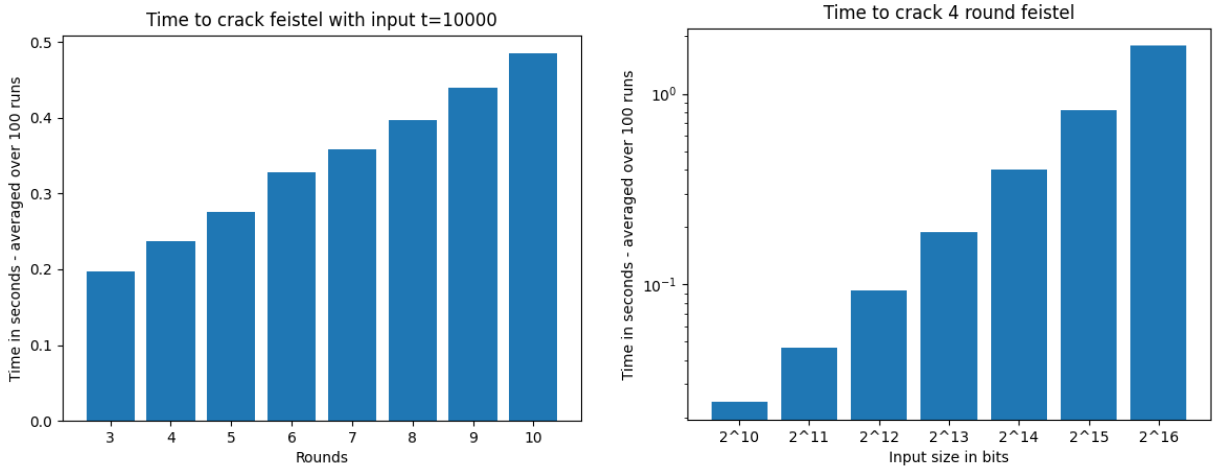
Each sub-problem now can be efficiently solved. The search space for cracking a feistel cipher on single bit inputs (the case *t = 1)* is *2^r*. As there are t sets of such problems, the big-O time complexity of cracking an r round cipher becomes *t*(2^r)*.

For *r > 3*, the worst-case time complexity is much tighter. For each plaintext-ciphertext combination, there exists at least one key of the form $K_{[x]} = (1, \ldots, 1, K_{r-2[x]}, K_{r-1[x]}, K_{r[x]})$ for all $x \leq t$. This is due to the feistel function being bitwise as well as the fact that applying bitwise AND on a key containing all ones has no effect. There are only three bits to search for each pair of bits in the plaintext and the ciphertext. This results in a worst-case time complexity of *t*(2^3)* or O(t).

Furthermore, if the number of rounds is known beforehand such as the case where *r = 3* or *r = 4*, then performance can be optimised by precomputing all single bit combinations of P_L, P_R, C_L, C_R that yields the key *K*. This can then be stored and accessed in a lookup table, to avoid searching all 8 possible values of *K* and testing whether $feistel((P_L, P_R), K, r) == C_L, C_R$. If the number of rounds is not known beforehand, then dynamic programming can be utilised to avoid recomputing the same combinations of plaintext and ciphertext repeatedly. The size of this lookup table will be at most *2^4* making it an effective optimisation that does not come at the cost of increased memory complexity.

3 Results

The performance of the algorithm outlined in the previous section was evaluated in two tests: the first aimed to measure how the number of rounds impacted runtime and the other aimed to measure the impact of input size. Each test was repeated 100 times with random inputs and run on a Ryzen 7 4800HS with 16GB of memory.



As expected, the size of the input was linearly proportional to runtime. Increasing the input size introduces additional subproblems to be solved at the same rate. Despite this, the performance of the cracker was still acceptable, solving 2^16 bit inputs in ~1 second.

However, what wasn't expected was there also being a linear relationship between the number of

rounds and runtime. Each additional round increased the runtime of the 10000 bit *crackrround* approximately 0.06 seconds. This is most likely due to memory limitations of the implementation, rather than flaws in the design of the algorithm. This will likely have little impact on real life performance.

4 Evaluation

As demonstrated in this report, bitwise AND is not a suitable feistel function. This is mostly due to the bit invariance between inputs and outputs that has been leveraged to reduce the time complexity of cracking the cipher from exponential to linear. In addition to this, using bitwise AND as the feistel function results in keys being predictably non-unique. That being, there are many different keys that map the same set of plaintexts to the same set of ciphertexts, further reducing the total search space. Bitwise AND is also a linear function, making the feistel cipher vulnerable to linear cryptanalysis: a process in which plaintext-ciphertext pairs are used to closely approximate the feistel function. To protect against this type of attack, the feistel function should be highly nonlinear.

A feistel cipher using bitwise AND is also not suitable for the creation of a hashing function using the Davies-Meyer construction and Merkle-Damgård transform. This is because zeros propagate through the function upon repeated application, resulting in the hash value to contain a disproportionately more zeros than on average. This significantly increases the amount of expected collisions in practice.

Bitwise AND has the positive property that it is not bijective. The input to the function cannot be fully determined from its outputs alone. An invertible feistel function would further reduce the amount of computation required to determine the key.

5 Conclusion

To summarise, not all non-invertible functions are suitable to be used as the feistel function. A secure choice of feistel function should protect against similar attacks to the one outlined in this report by having changed in the input masked by an unpredictable change in the output. In addition to this, a feistel function should be highly nonlinear so it couldn't be effectively approximated by a linear function. To solve this, the feistel function should perform a lookup to a table of pseudorandom input-output pairs known as a S-box.