Cryptography Report

Tom Potter

There is currently a range of attack methods against the Data Encryption Standard (DES) which exploit its multiple vulnerabilities. These range from linear cryptanalysis which creates a linear approximation of the encryption process to find biases in plaintext-ciphertext combinations for varying keys, to the improved Davies' attack, which analyses the non-uniform distribution produced by substitution boxes during an analysis phase to drastically reduce the key search space. These approaches aim to reduce the complexity of cracking DES, however they are known to be impractical to implement due to their reliance on a large collection of known plaintexts and ciphertexts: 2^{43} for an accurate linear cryptanalysis approximation, and 2^{50} to achieve a 51% success rate in the Davies' attack. This makes these attacks incredibly memory intensive, and can negate the reduction in DES evaluations by relying upon a (possibly slow and unreliable) oracle to generate plaintext-ciphertext pairs. Therefore, the most feasible attack on DES remains an exhaustive key search (brute-force), whereby we test all 2^{56} possible variations of the (64-bit) key against a single known plaintext P and ciphertext C^* to find the corresponding target key K^* used in this encryption. Therefore, this is the approach that my implementation will focus on in the following description and evaluation.

In order to mount an effective brute-force attack on the DES key space, the implementation must optimise its two core elements (the cipher and cracker) to as many rapid DES evaluations as possible - where we encrypt the input plaintext P using a given test key K - in the shortest time-frame. The first element is the implementation of a DES cipher used to conduct each encoding $E_K(P) = C'$. To maximise the throughput of the cipher, the pre-computation step (i.e. the computation of subkeys k from K, and the application of the initial bit permutation IP), 16 distinct rounds of the DES Feistel cipher, and post-computation step (swapping the left and right blocks, and applying the final bit permutation $FP = IP^{-1}$), can be split up into distinct functions (allocated to different registers) which each form part of a full composite pipeline of the entire encryption process. The path of single test key K' thus starts with the input of K' and P into the pipeline, where step 0 (pre-computation) is applied, generating the initial output (L_0, R_0) and a set of subkeys which are stored in a global 16×16 matrix, a single row holding all values of k such that each column indicates the subkey k_i used in DES round i. The 1st-16th step of this process then implement the sequential rounds of DES, with each round i taking input of subkey k_i and the output of the last round (L_{i-1}, R_{i-1}). The final round (post-computation) subsequently outputs the corresponding ciphertext C', which we can quickly evaluate against the goal ciphertext C^*, outputting the found key $K' = K^*$ if $C' = C^*$.

Furthermore, as in this context we have access to an oracle returning the corresponding ciphertext to any plaintext (formatted as a hexadecimal string with length a multiple of 8), we can also ensure our value of P to be optimally fast to encrypt. Therefore, we pick P (and produce its corresponding C^*) such that it only contains 8 hexadecimal characters, and thus only requires one pass through the pipeline per encryption (whereas we would have to use more resources to encrypt the multiple blocks present in longer messages in ECB mode).

The full capacity of this pipeline can then be utilised by not only performing a single encoding $E_{K'_j}(P)$, but instead entering a new pair (P, K'_{j+1}) into the pipeline as soon as the previous encryption $E_{K'_j}(P)$ has completed its initial step (and stored its subkeys in row $j \pmod{16}$ of the subkey matrix), maximising the rate at which keys can be tested. This can be implemented by maintaining a constant global clock, whereby at each clock tick a new encryption enters the pipeline and all ongoing internal encryptions advance through another round of the cipher. Thus, there are 18 encryptions taking place simultaneously in the pipeline, with one encryption completing at each clock tick. We can then increase the frequency of the clock as high as possible, maximising the throughput of the pipeline whilst ensuring that each step is completed at every stage.

The other main element of the attack is the cracking process itself through which we take input of our test pair (P, C^*) and try all 2^{56} possible 56-bit values of K' until one satisfies $E_{K'}(P) = C^*$. Our implementation also takex input of an integer t specifying the number of available threads/compute nodes upon which we can run our attack in parallel. The cracker initially creates an array of length t, where each element i specifies the start key ($\approx \frac{i*2^{56}}{t}$th possible key from the all-zero start key) and maximum number of evaluations ($\approx \frac{2^{56}}{t}$) such that each thread has an evenly balanced, non-overlapping interval of the key search space. We then initialise t parallel cracking threads, where each thread i operates on a separate domain, taking input of (P, C^*), the start key and number of evaluations assigned to i. Each thread iterates through its allocated interval of string keys, utilising the aforementioned DES pipeline to iteratively encrypt P using each key K'_j. The process of checking outputs of the pipeline can further be parallelised by not checking for equality between the output ciphertext C'_j and goal C^* in the pipeline, or even in thread i itself. Instead, we employ a hierarchical system where on the lower level are the cracking threads, and on a higher level are a set of 'master' threads into which the pipeline output C'_j is fed after each clock tick, allowing thread i to immediately return to processing encryptions in its pipeline, and leaving the evaluation of whether $C'_j = C^*$ to the master thread, which can compute multiple of these computationally inexpensive comparisons in one step. The number of master threads $m \geq 1$ can be optimised to take maximal usage of the resources available, where in the case of multiple master threads ($m > 1$) each evaluate the outputs of a distinct group of $\approx \frac{t}{m}$ cracker threads. If a master thread identifies that a key K' (from any cracker thread) satisfies the evaluation, it sends a signal to close and join all threads throughout the system whatever state they are currently in (returning to the main thread) and outputs the value of the found key $K' = K^*$. If the threads are closed for any other reason, for example an error, the current key furthest through the encryption pipeline for each thread should be output and stored, such that the key search can be resumed again by setting those keys as the start key for each thread, and the adjusting number of evaluations accordingly.

As with all brute-force attacks, the time taken T to find the target key K^* is proportional to the number of bits in the key. In the case of DES, the 64-bit key has an effective size of 56 bits (as 8 are used as odd parity bits), and thus we have $T \propto 56$. As previously stated, there is precisely 2^{56} different combinations of these bits, meaning to find K^* with 100% certainty we would have to try all 2^{56} variations of the key value. However, to give a majority success rate of 51% we can assume that we need to test around 2^{55} values (half of the key space). It is clear that the speed at which this key space can be searched depends upon both the number of computing nodes (threads) and the speed at which these compute. For each instance of the DES pipeline, we complete one DES evaluation per clock tick - including the comparison against the target ciphertext (which we model as happening simultaneously due to multiple comparisons happening in parallel to each evaluation). Therefore, the speed at which we can make these evaluations is directly reliant upon the frequency F of our implementation of the clock, which is a variable parameter given the resources of the machine being used. The number of parallel cracking threads t (and master threads m) can also be adjusted in line with the available resources (maximising the total $t + m$ without overflowing the available compute/memory space), theoretically improving the time to find K^* from 2^{55} clock ticks to $\frac{2^{55}}{t}$. Thus, the time in seconds is given by $(\frac{2^{55}}{t})/F = \frac{2^{55}}{t*F}$, where t is an integer and F is measured in *Hz*. Note that the time taken to compute $C' = C^*$ for a single ciphertext is negligible compared to the necessary time to compute a single round of DES, and thus we can implement $m \gg t$; therefore, in subsequent calculations I will simply refer to t. For example, on a state-of-the-art machine, our aim for optimal t would be around 1800 (the number of parallel chips used in the EFF implementation "Deep Crack" in 1998), paired with a desirable but reasonable frequency F of around 50MHz, which would allow us to evaluate $t * F = 1800 * 5 \times 10^7 = 9 \times 10^{10}$ keys per second, and thus it would take $\frac{2^{55}}{9 \times 10^{10}} = 400320$ seconds, or approximately 4.6 days. An even more powerful setup, such as that used to crack DES by D. Hulton and M. Marlinspike in 2012, would have $t = 1900$ and $F = 4 \times 10^8 Hz$, which using my evaluation would take $\frac{2^{55}}{1900 * 4 \times 10^8} = 47406$ seconds, or approximately 13 hours. However, it is not given that this frequency value would be reachable, as the maximum frequency depends upon the resources of the machine being used and thus should be determined experimentally after the implementation has been built.