How easy is collision search? Application to DES

Jean-Jacques Quisquater   Jean-Paul Delescaille

Philips Research Laboratory Belgium

Avenue Albert Einstein, 4

B-1348 Louvain-la-Neuve, Belgium

jjq@prlb.philips.be — jpdesca@prlb.philips.be

(Extended summary)

1 About collisions

Given a cryptographic algorithm f (depending upon a fixed message m and a key k), a pair of keys with collision k_1 and k_2 (in short, a *collision*) are keys such that

$$f(m, k_1) = f(m, k_2).$$

The existence of collisions for a given cryptographic algorithm means that this algorithm is not *faithful* in a very precise technical sense (see [3]). It is important to know if it is *easy* to find collisions for a given cryptographic algorithm. Indeed, the existence of such easy-to-find collisions means that this algorithm (or, maybe, its mode of use) is not secure for many applications related to hashing functions used in the context of digital signatures.

While there is a large probability that DES, in its basic mode, has collisions, nobody has found a collision for DES until now. It is thus a challenging problem to find only one. We found 21 collisions with the same plaintext (= identical m).

The used algorithm is based on the so-called theory of *distinguished points* (see the abstract in the proceedings of CRYPTO '87 by the same authors). The result was obtained thanks to efficient implementations of DES on VAXes and SUN's and intensive use of the idle time from 35 workstations at our laboratory.

2 Algorithms

2.1 A naive algorithm

Here we will use DES(m, k) for denoting DES in its basic mode, with m as the input message (64 bits), k as the key (56 bits); the obtained result has 64 bits.

If we suppose that DES can be modelled as a *random mapping*, the following algorithm works. Given a fixed message m, compute about 2^{32} values DES(m, k_i), where the k_i's are all different. Sort the obtained values. With a high probability, we will obtain one collision. The problem with such a method is the need of a very large memory (disk and RAM) for storing the values. The associated problem of sorting a lot of data is not so simple. This method is not feasible as an annex task in a network of workstations.

2.2 Algorithm without memory

There exist very efficient algorithms to find cycles in periodic functions mapping some finite domain D into D (see [1]). If we take a random element x from the finite set D and generate the infinite sequence $f^0(x) = x, f^1(x) = f(x), f^2(x) = f(f(x)), ...$, then we know that the sequence becomes cyclic. That is, there exists some value l that $f^{l+c}(x) = f^l(x)$ (the *point of contact* common to the leader and the cycle) and $f^{l+c-1}(x) \neq f^{l-1}(x)$ (one value on the leader and one on the cycle). That is, by definition, we found a collision for such a f.

It is simple to modify such algorithms to find cycles when the domain D has less elements than the codomain. That is, the input k has less elements than the output. We need some projection function g for mapping the output onto the next input. However, we have a new problem. The common point is not necessary the result of a collision. In fact, for DES, the probability of having found a collision is one out of 256. We now call a pair of antecedent points of such a common point a *pseudo-collision*.

So the algorithm becomes the following one. Given m and an initial value x_0, find a first pseudo-collision. Verify if it is a *true* collision. If no, try again with a new initial value x_1, aso. A new problem with such an algorithm is that we compute many times the same values due to the fact that we are computing values on the same cycle with high probability (see the paper by Flajolet and Odlyzko, in these proceedings). An effective technique to overcome this problem is the use of distinguished points.

2.3 Effective algorithm in use

Figure 1 describes the algorithm we are using. The two variables *pseudo_collision* and *pseudo_cycle* are first set to false. The variable i is a counter used for the number of the current initial value. At each call of *new_init*, a new and different value for y is chosen. The counter k is used for computing the number of computed values since the last call of *add*, that is, since the last time we found a distinguished point. The procedure *distinguished_point* is true if the input y is a distinguished point, that is a value with some attribute fast to compute (for instance, we used the attribute that the value y had 20 bits set to 0 at the left). The procedure *add* puts the value y into TABLE by checking if there is another entry already there with the same value y. It is a fast operation (comparisons with elements in a small table). If yes, then *add* puts the variable *pseudo_collision* to true. We have then detected a pseudo-collision but we do not know its exact value. We will find that in a next phase. The variable *limit_k* is used for avoiding the problem of looping due to a cycle without any distinguished point. After some time, TABLE contains a large number of values indicating pseudo-collisions. We are now in position to find out if there are some collisions in this set. For that we compute the effective values of the pseudo-collisions. Sometimes it is a collision.

3 Results

The first collision has been found January 13, 1989, the birthday of the first author (another application of the birthday paradox!) after 3 weeks of computation.

Here is this first one:

PLAIN = 0404040404040404 (in hexadecimal)

k_1 = 4A5AA8D0BA30585A (idem)

k_2 = 46B2C8B62818F884 (idem)

RESULT = F02D67223CEAF91C (for the two keys)

```
$pseudo\_collision \leftarrow$ false;
$pseudo\_cycle \leftarrow$ false;
$i \leftarrow 0$;
repeat
    $y \leftarrow new\_init(i)$;
    $k \leftarrow 0$;
    repeat
        $y \leftarrow f(y)$;  $k \leftarrow k + 1$;
        if ($distinguished\_point(y)$) then
            begin
                $add(y,\ pseudo\_collision)$;  $k \leftarrow 0$ ;
            end
        if $k > limit\_k$ then $pseudo\_collision \leftarrow$ true ;
    until $pseudo\_collision$ or $pseudo\_cycle$;
until $i >$ limit_$i$;
```

Figure 1: The pseudo-collision detecting algorithm

The workstations worked in parallel on the same problem (= same m) with distinct initial points and distinct projection functions g. In this context, the number of found collisions is proportional to the square of the used time if we consider that the studied cryptographic function acts as a random mapping. The projection functions were simply different sets of 56 bits out of the outputs of 64 bits.

We have found 21 collisions for DES (March 13, 1989). The table at the end of this paper gives the complete values.

An algorithm was implemented to draw the mappings resulting from these DES computations (see an example in the paper by Flajolet and Odlyzko, in these proceedings).

References

[1] Robert Sedgewick, Thomas G. Szymanski and Andrew C. Yao, *The complexity of finding cycles in periodic functions*, SIAM J. Comput., vol. 11, 2, pp. 376–390, 1982.

[2] Jean-Jacques Quisquater and Jean-Paul Delescaille, *Other cycling tests for DES*, Springer Verlag, Lecture notes in computer science **293**, Advances in cryptology, Proceedings of CRYPTO '87, pp. 255–256.

[3] Burton Kaliski, Ronald Rivest and Alan Sherman, *Is the Data Encryption Standard a group? (Results of cycling experiments on DES)?*, J. Cryptology, vol. 1, 198, pp. 3–36.

PLAIN	KEY 1	KEY 2	CIPHER
0404040404040404	4a5aa8d0ba30585a	46b2c8b62818f884	f02d67223ceaf91c
	d296c2ca66be3c60	1680b00c1c22c6b4	e20332821871eb8f
	6edaa03254d2a298	22a64edc20e07032	7237f9e44466059f
	cc3adc3616cc1c32	620e08e886aa8c1c	345d8975676ffde0
	a2aa9adc56a60ad6	b41ebe7a88c4a8c8	301c9a64b903048d
	5888c640ee3016d4	8654a2b862a82486	8f4a67da0852722d
	1e620c46682e325c	0ed86014328cf2da	96f0faf4f80b6b29
	780a76586c7c0ca4	92f69c5aa2c84ee8	1d901196097a93f4
	46f422a832ac0c18	1680f2049484b4b2	85795a73b4af5d78
	3eb8406c969c9c84	e4f06aaea2022e02	46184d44b739a147
	28e8161878343ea0	36a0f03afe48c226	c5ed963b29a48bf6
	060c0e048614bc42	5c4afa4ae0c62a84	c931dab489f515a1
	d0e4aa90baba681c	d8fc6cba3c0a946c	a3c7d6d33eb1400d
	36da7e6010d6a07e	2c2c5a243cd882fa	6a5d431ed4863421
	7aac9c602e9854b6	ac78ca74c6a0ea6e	2edeaaa86e5141af
	ce806eee7cfcd2ec	ae8838904874c606	150e0b6ff35b4f0e
	366cf4baa8cc6c80	76f6527c54447ade	77964b1e86be688e
	6ece1e20bef2b0f8	be827240c8bc3e6a	f29fdbc8dc6c174a
	5e301c2452d88476	5406c60cb4d6f0c8	c6120f53b62eed0d
	0e5ebe562c961274	b45e08326ea40e10	ef5293f14f84fc4f
	624e36aa48926a2e	a862d2aef0c06c54	7dd3c3d34ea30c2f

Table of known collisions for DES.