Volume : 2 | Issue : 1 | January 2013 ISSN - 2250-1991

Research Paper Computer Science

Efficient Black-Box Collision Search in cryptanalysis

*Kalpesh R. Rakholiya ** Dr. Dhaval Kathiriya

* Professor, Research Scholor of C.M.J. University Shilong, Meghalaya

** Research Guide in computer science, C.M.J. University ,Meghalaya

ABSTRACT

The collision search problem is, given a function F with a finite range, to find distinct inputs x and x' such that F(x) equals F(x'). Collision search is an important tool in cryptanalysis, most notably for computing discrete logarithms, making meet-in-the-middle attacks, or finding hash function collisions. After a brief review of historical results, this section describes the state-of the-art serial and parallel methods for searching collisions.

Keywords: random, theorem, negligible memory, eliminate, asymptotically ,probability, meet-in-middle

Tails and Cycles

Let Fn denote the set of all functions from a domain D of size n to a codomain of size n, with n finite. Let F be random element of Fn (that is, a random mapping from and to n-element sets). The range of F is expected to contain $n(1-1/e) \approx 0.63n$ distinct elements. Therefore , F is expected to have collisions F(x)=F(x'),x ¹ x'. Efficient methods for finding such collision exploit the structure of F as collection of cycles.

Consider the infinite sequence $\{x_i = F(x_i - 1)\}0 < i$, for some arbitrary starting value x_0 .Because D is finite, this sequence will eventually begin to cycle. Hence, there exist two smallest integers $\mu \geq 0$ (the tail length) and $\lambda \geq 1$(the cycle length) such that $x_i = x_i + \lambda$ for every $i \geq \mu$. Such a structure then yields a collision at the point where the cycle begins: $F(x_{\mu-1}) = F(x_{\mu+\lambda-1}) = x_\mu$.

The birthday paradox illustrates well the above structure: in a sequence of random numbers in{1,…,n}, the expected number of draws before a number occurs twice is asymptotically $\sqrt{\pi n/2}$. This is because the expected values of the tail length m and of the cycle length l sum to $\sqrt{\pi n/8} + \sqrt{\pi n/8} = \sqrt{\pi n/2}$. This value is sometimes called the rho length, because of the rho shape of the sequence, as noticed by Pollard[1].

A trivial collision search algorithm repeats the following: pick random x and x' , return them as a collision if F(x) equals F(x') , otherwise continue the search. About n trials are required, since x and x' collide with probability $1/n$. A less trivial algorithm exploits the existence of cycles by storing a sequence $\{x_i = F(x_{i-1})\}_{0<i<\sqrt{\pi n/2}}$, sort it and look for a collision. State-of-the-art methods eliminate the large memory requirements and the cost of sorting a large list. In the following we review these methods , starting with explicit cycle-detection methods, then presenting modern techniques that tailored to supercomputers. Finally, we explain how to apply those methods to concrete cryptanalytic problems.

Cycle Detection Based Methods

The low-memory cycle-detection method of Floyd is at the base of Pollard's rho method for factoring and computing discrete logarithms . It is based on the following observation from [2]:

Theorem 1 : For a periodic sequence x0,x1,x2,…, there exists a unique i>0 such that xi=x2i and the smallest such $\lambda \leq j \geq \lambda + \mu$.

Based on Theorem 1 , Floyd's method picks a starting value $x_0 = x_0'$ and compares the values $x_i = F(x_{i-1})$ and $F(F(x'_{i-1})), i \geq 1$.The expected number of iterations before reaching a match is [19] $\sqrt{\pi^5 n/288}$.

Floyd's algorithm detects that the sequence has reached a cycle , but does not give the values of l and m , nor a collision for F. this can be done as follows, once xi = x2i is found: generate xj and $x_{j+i}, j \leq 0$, until finding $x_j = x_{i+i}$; at the first equality we have j=m. If none of the values x_{j+i}equals xi then l=m. If none on average $\sqrt[2]{\pi n/2}$ evaluations of F. Finally, detecting the cycle and locating the collision with the above method costs

$$3\sqrt{\pi^5 n/288} + 2\sqrt{\pi n/2} \approx (3.09 + 2.51)\sqrt{n} = 5.60\sqrt{n}$$

evaluations of F, and requires negligible memory (storage of a few xi's). Slightly more efficient variants of Floyd's algorithm were proposed by Brent [3] and Teske [4]. Sedgewick et al. showed how to eliminate the redundant computations by using a small amount of memory [5], but their algorithm is not as general as Floyd's (in particular, it cannot be combined with Pollard's rho factoring method).

Parallel Search with Distinguished Points

A disadvantage of Floyd's algorithm (and thus of Pollard's rho method) is that it cannot be parallelized efficiently: m processors don't provide a 1/m reduction of complexity. This is because one has to wait for a given invocation of F to end before the next can begin. Efficient parallelization of collision search takes a different approach, by using the idea of distinguished points, The idea of using distinguished points (i.e., points that have some predefined easily checkable property, like having ten leading zero bits) was proposed by Quisquater and Delescaille [6] for searching DES collisions, and earlier noted by Rivest [7, p.100] in the context of Hellman's time-memory tradeoff. Below we describe a simple method for efficient parallelization of collision search using distinguished points, and due to van Oorshot and Wiener [8].

Let m be the number of processors available, and consider some easily checked property $\mathcal{P} \subsetneq \mathcal{D}$ that a random point satisfies with probability θ <1. To perform the search, each processor

1. Selects a starting value x0 .
2. Computers $x_i = F(x_{i-1}), i > 0$, until a distinguished point $x_d \in \mathcal{P}$ is reached;

3. Add xd (along with x0 and d) to a common list for all processors;
4. Repeats the process.

The algorithm halts when a same distinguished point appears twice in the common list, which means that two distinct sequences (x0,…,xi) and $(x'_0,\dots,x'_j)$ lead to same value $x_i = x'_j$(one should ensure that a same starting value is not used twice). With high probability , one will easily deduce a collision from these two sequences (if the first sequence leads to the starting point of the second, then no collision will be found).

The above algorithm runs in time about $\sqrt{\pi n/2/m} + 2.5/\theta$ to locate a collision, hence parallelization provides a linear speed-up of the search.

Application to Meet-in-the-Middle

Parallel collision search using distinguished points can be directly applied to find collisions for hash functions. It can also be adapted to compute discrete logarithms in cyclic groups. Here we show how it can be used to perform meet-in-the-middle (MITM) attacks, which are used for computing preimages of MD5 and HAVAL.

The problem considered is, given two functions F1 and F2 in Fn , to find x and x' (not necessarily distinct) such that F1(x) equals F2(x'). A solution can be found by defining an easily checked property P , and by considering the function

$$F(x) = \begin{cases} F_1(x) & if\ x \in p \\ F_2(x) & otherwise \end{cases}$$

Under reasonable assumptions on F1 and F2 , and assuming that a random x satisfies P with probability 1/2, a collision F(x) = F(x') will be useful as soon as x satisfies P but not x. When the cost of computing F1 and F2 significantly differs (for example if one of them represents a shortcut preimage attack on some component), the property P can be adapted to optimize the complexity of the attack, so that F1 is called more often than F2.

Conclusion:

Note that the MITM problem considered here, and often encountered in cryptanalysis, differs from what is called MITM in [8]. Indeed, the latter attack looks for a single "golden value", and its complexity heavily depends on the domain size, whereas in the former complexity only depends on the range size. Example of applications of memory less MITM are our attacks on the SHA-3 candidate MCSSHA-3 and on MD5.

REFERENCES

[1] John M. Pollard. Monte-Carlo methods for index computation mod p. Mathematics of Computation, 32(143), 1978. | [2] Donald E. Knuth. The Art of Computer Programming. Addison-Wesley, second edition edition, 1981. | [3] Shi Bai and Richard P. Brent. On the efficiency of Pollard's rho method for discrete logarithms. In CATS, 2008. | [4]Edlyn Teske. Speeding up Pollard's rho method for computing discrete logarithms. In ANTS, 1998. | [5] Robert Sedgewick, Thomas G. Szymanski, and Andrew Chi-Chih Yao. The complexity of finding cycles in periodic functions. SIAM Journal on Computing, 11(2), 1982. | [6] Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search? Application to DES (extended summary). In EUROCRYPT, 1989. | [7] Dorothy E. Robling Denning. Cryptography and Data Security. Addison-Wesley, 1982. | [8]Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. Journal of Cryptology, 12(1), 1999.