Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude [†]

Paul C. van Oorschot and *Michael J. Wiener*

Bell-Northern Research, P.O. Box 3511 Station C, Ottawa, Ontario, K1Y 4H7, Canada
{paulv,wiener}@bnr.ca

1996 May 22

Abstract. Meet-in-the-middle attacks, where problems and the secrets being sought are decomposed into two pieces, have many applications in cryptanalysis. A well-known such attack on double-DES requires 2^{56} time and memory; a naive key search would take 2^{112} time. However, when the attacker is limited to a practical amount of memory, the time savings are much less dramatic. For n the cardinality of the space that each half of the secret is chosen from (n=2^{56} for double-DES), and w the number of words of memory available for an attack, a technique based on parallel collision search is described which requires $O(\sqrt{n/w})$ times fewer operations and $O(n/w)$ times fewer memory accesses than previous approaches to meet-in-the-middle attacks. For the example of double-DES, an attacker with 16 Gbytes of memory could recover a pair of DES keys in a known-plaintext attack with 570 times fewer encryptions and 3.7×10^6 times fewer memory accesses compared to previous techniques using the same amount of memory.

Key words. Meet-in-the-middle attack, parallel collision search, cryptanalysis, DES, low Hamming weight exponents.

1. Introduction

Many cryptographic techniques are susceptible to meet-in-the-middle attacks. Two well-known examples are double-DES encryption [5] and discrete logarithms with limited Hamming weight exponents [8]. A third example is an attack on a scheme for using an untrusted server to perform most of the work in an RSA computation [2]. A reduction in the run-time of meet-in-the-middle attacks is thus of wide-ranging interest. Such a reduction is possible by solving meet-in-the-middle problems using an algorithm based on collision search, and is the subject of this note.

Parallel collision search [13] based on Pollard's rho-methods [11, 12], was introduced as a means of efficiently parallelizing search problems. By formulating a meet-in-the-middle attack as a collision search problem, the run-time of the attack may be decreased. These ideas are explored in the remainder of this paper, organized as follows. In Section 2, a general meet-in-the-middle attack is described and formulated as a collision search problem, which is solved in Section 3 using parallel collision search. Section 4 compares the attack time of the collision search based technique to previous meet-in-the-middle methods. Section 5 concludes the paper.

[†] *Proceedings of Crypto '96* (to appear), Springer-Verlag LNCS, August 1996.

2. Formulating Meet-in-the-Middle Attacks as Collision Search Problems

A general meet-in-the-middle attack involves two functions, f_1 and f_2, for which there are two inputs, a and b, such that $f_1(a) = f_2(b)$. The objective is to find a and b. There may be other pairs of inputs which also satisfy this equation, but typically only one particular pair is the solution being sought. We begin by showing how one would construct f_1 and f_2 for three example cryptanalytic problems. Then a single function f suitable for a variant of parallel collision search is constructed from f_1 and f_2.

The first example is a mode of DES [3] called double-DES where data is DES-encrypted twice with two independent keys (k_1, k_2). Diffie and Hellman [5] showed that this is susceptible to a meet-in-the-middle attack which finds k_1 and k_2. Suppose that we are given a plaintext-ciphertext pair (P, C) such that P maps to C under double encryption with the unknown pair of keys (k_1, k_2). In this case, function f_1 is encryption of the constant P with a DES key, and f_2 is decryption of the constant C with a DES key. Note that $f_1(k_1) = f_2(k_2)$; here P and C are implicit constants in f_1 and f_2 (see Section 4 for a discussion of previous methods for recovering double-DES keys using f_1 and f_2). There may be other false key pairs which map P to C, but only one pair of keys is correct. One additional plaintext-ciphertext pair generally suffices to uniquely determine the correct pair of keys.

The second example is the discrete logarithm problem in the special case where exponents have low Hamming weight. Given a generator α of a cyclic group and an element $y = \alpha^x$ where x has bitlength m and Hamming weight t, we wish to find x. This problem can be solved with a meet-in-the-middle attack (e.g., see Heiman [8] or Pfitzmann and Waidner [10]). Observe that all possible values of x can be written as the sum of two m-bit values, each with Hamming weight $t/2$ (assume t is even). Let $n = \binom{m}{t/2}$ and let h map integers in the interval $[0, n)$ to m-bit values with Hamming weight $t/2$. Then using $f_1(i) = \alpha^{h(i)}$, and $f_2(j) = y/\alpha^{h(j)}$ $(= \alpha^{x-h(j)})$, there exist inputs a and b such that $f_1(a) = f_2(b)$. Finding inputs a and b (e.g., by a meet-in-the-middle attack) gives x because then $x = h(a) + h(b)$. Coppersmith observed that this attack could be made more efficient by trying (until success) to partition the exponent bits into two groups of $m/2$ bits each with Hamming weight $t/2$. By the Mean Value Theorem there exists a set of $m/2$ contiguous bits of the exponent with Hamming weight exactly $t/2$. Therefore, the attack can be completed in at most $m/2$ trials with $n = \binom{m/2}{t/2}$, so that each trial takes less time than the above version. This approach is guaranteed to give a solution in a fixed period of time, but we note that the expected number of trials can be reduced to significantly less than $m/2$ as follows. If one randomly partitions the bits in the exponent into two sets of size $m/2$, the probability that each group will have Hamming weight exactly $t/2$ is $\binom{m/2}{t/2}^2 \Big/ \binom{m}{t}$. Therefore, by executing independent trials which partition the exponent bits at random, one expects to complete the discrete logarithm in $\binom{m}{t} \Big/ \binom{m/2}{t/2}^2 \approx \sqrt{\pi t\,(1 - t/m)\,/2}$ trials.

The last example is a scheme for using an untrusted server to speed up RSA computations on a smart card [2]. In this scheme, an RSA private exponent is represented as

$d = \Sigma_{i=1}^{m} a_i d_i$, where d_i is public and a_i is a small secret, for i=1,…,m. To compute x^d, the untrusted server computes x^{d_i} for i=1,…,m and the smart card computes $\Pi_{i=1}^{m}(x^{d_i})^{a_i}$. Let $A=(a_1,\ldots,a_{m/2})$, $B=(a_{m/2+1},\ldots,a_m)$, $D=(d_1,\ldots,d_{m/2})$, and $E=(d_{m/2+1},\ldots,d_m)$. Then $d = A{\cdot}D + B{\cdot}E$. For RSA, $h = h^{ed} \bmod n$, where e is the RSA public exponent, n is the RSA modulus, and h is some positive integer less than n. This can be rewritten as $h = h^{e(A\cdot D + B\cdot E)} \bmod n$ or $h^{e(A\cdot D)} \bmod n = h^{1-e(B\cdot E)} \bmod n$. Using $f_1(x) = h^{e(x\cdot D)} \bmod n$ and $f_2(x) = h^{1-e(x\cdot E)} \bmod n$ gives $f_1(A) = f_2(B)$, which allows a meet-in-the-middle attack.

Returning to discussion of the general attack, f_1 and f_2 must have the same range, but need not have the same domain. It is not difficult to handle different domains, but to simplify the discussion below, we assume the domains are equal (as in all examples above). The problem is to take f_1: $D \rightarrow R$ and f_2: $D \rightarrow R$ and find pairs of inputs, i and j, such that $f_1(i) = f_2(j)$ until the correct pair of inputs (a and b) is found. If many pairs of inputs give a collision between f_1 and f_2, it may be necessary to have a test to determine whether the “correct” pair (a and b) has been found. In the case of double-DES, this can be done by verifying the candidate key pair using a second plaintext-ciphertext pair.

To use parallel collision search, we require a single function f such that (1) its domain and range are equal; and (2) there are two particular inputs to f which give the same output and which, if found, leads to a solution to the problem at hand. Let g: $R \rightarrow D{\times}\{1,2\}$ be a function which maps an element of the range of f_1 (and f_2) to an element of D along with a bit which is used to select between f_1 and f_2. We assume here that $|R| \geq 2|D|$. Now define f: $D{\times}\{1,2\} \rightarrow D{\times}\{1,2\}$ as $f(x, i) = g(f_i(x))$, for i=1,2. Because $f_1(a) = f_2(b)$, it follows that $g(f_1(a)) = g(f_2(b))$ and $f(a, 1) = f(b, 2)$; this is the collision which is sought.

3. Solving the Collision Search Problem

In this section we show how to use parallel collision search to solve the collision problem constructed from the general meet-in-the-middle attack in Section 2. An important point about this use of parallel collision search in the three applications given earlier is that there are many pairs i, j such that $f(i, 1) = f(j, 2)$, but among them is a unique collision pair, $f(a, 1) = f(b, 2)$ solving the meet-in-the-middle problem. Typically, a very large number of collisions in f must be found in order to find the one particular meaningful collision that is sought, which we call the *golden collision*. For the example of double-DES, the collision sought is $f(k_1, 1) = f(k_2, 2)$ for the correct key pair (k_1, k_2). In contrast, for the hashing and discrete logarithm[1] applications of parallel collision search considered by van Oorschot and Wiener [13], there were many collisions which solved the original problem and typically a useful collision was found after only a small number of collisions.

For reference, we briefly describe parallel collision search before considering how it should be modified to find a golden collision. Given a function f: $S \rightarrow S$, choose a

[1] This previous paper considered the general problem of finding a discrete logarithm in a cyclic group as opposed to the Section 2 example of the special case of exponents with restricted Hamming weight.

distinguishing property[1] which distinguishes a proportion θ of the elements of S (e.g., $\theta = 2^{-10}$ when elements with 10 leading zero bits are distinguished). Choose an element $x_0 \in S$ and produce the sequence (trail) of points $x_i = f(x_{i-1})$, for $i = 1, 2, \ldots$ until a distinguished point x_d is reached. Store the triple (x_0, x_d, d) in a table. Repeat this process for many x_0 values. The occurrence in the memory of two triples with the same x_d value indicates their trails have collided. By stepping the trails forward again from their respective x_0 values, one can find two inputs, u and v, to f such that $f(u) = f(v)$. Let $N = |S|$. One expects to perform $\sqrt{\pi N/2}$ iterations of f (possibly spread across multiple processors) before one trail collides with another [13]. As the available memory fills, the probability of finding a collision grows and the number of collisions found grows quadratically. Finding k collisions is expected to take $\sqrt{\pi k N/2}$ iterations of f [13].

Solving the meet-in-the-middle problem requires finding the golden collision out of the many available collisions. Because there are $\binom{N}{2} \approx N^2/2$ pairs of inputs and the probability that both inputs are mapped by f to the same output is 1 in N (if f behaves randomly), one expects that there are about $N/2$ collisions for a given random function f.

One may incorrectly reason that collisions will be found at random (with replacement) and that, on average, about $k = N/2$ collisions are required before locating the golden collision, requiring $\sqrt{\pi k N/2} = \sqrt{\pi}\, N/2$ iterations of f. However, this faulty analysis ignores two important facts. The first is that although the expected time between detected collisions drops as the memory fills, the expected time required to locate each detected collision by stepping the two trails forward to the collision point does not decrease. The second is that, generally, not all collisions are equally likely to occur; thus some collisions will be found many times while others will never be found.

To understand this latter point, consider a directed graph whose vertices are the elements of the set S, with a directed edge from each vertex x to the vertex corresponding to element $f(x)$. A collision is a pair of elements whose edges end at a common third element. The likelihood that a particular collision will be detected is a function of the sizes of the predecessor trees of the pair of elements involved in the collision. There is considerable variation in the sizes of predecessor trees in random mappings; see Flajolet and Odlyzko [7]. In the worst case, the elements a and b involved in the golden collision may have no predecessors at all. The probability of this occurring is about $1/e^2 \approx 14\%$. In this case, the golden collision will not be detected until both a and b are selected as starting points for trails and both are in memory at the same time.

A solution to these complications in practice is to limit the number of collisions sought using a particular function f. If the golden collision is not found after a fixed period of time, construct a new version of f known to contain a golden collision and repeat. Because

[1] The idea of using a distinguishing property was attributed to Rivest by Denning [4, p.100] as a means of improving Hellman's time-memory trade-off for attacking block ciphers [9].

f was constructed with a mapping g, one could simply choose a new mapping g to make a new version of f.

It remains to be determined what proportion θ of points to distinguish, how long to continue using each version of f, and how long it is expected to take to find the golden collision. Another important statistic in highly parallelized attacks is the number of memory accesses required. Proposition 1 gives an empirical result for these parameters.

Proposition 1 (heuristic): Let n be the cardinality of the domain of functions f_1 and f_2 above, so that the cardinality of the domain and range of f is $N = 2n$. For a memory which can hold w triples, the (conjectured) optimum proportion of distinguished points is $\theta \approx 2.25\sqrt{w/N}$, and one should generate about $10w$ trails per version of f. The expected number of iterations of f required to complete a meet-in-the-middle attack using these parameters is $2.5N^{3/2}/w^{1/2} \approx 7n^{3/2}/w^{1/2}$, and the expected number of memory accesses is $4.5N = 9n$.

Justification: Let us begin with a simple, but flawed, run-time analysis. If the memory is full with w distinguished points, then the total number of points on the trails leading to those distinguished points is about w/θ. For each trail point generated with f in the space of size N, the probability of producing a point on one of the existing trails is $w/(N\theta)$. The required number of generated points per collision found is then $N\theta/w$. To locate a collision, each trail involved must be retraced from its start to the colliding point requiring a total of $2/\theta$ steps on average. The total cost per collision detected is $(N\theta/w) + (2/\theta)$ steps. This is minimized at $\sqrt{8N/w}$ steps when $\theta = \sqrt{2w/N}$. The expected number of collisions generated before the golden collision is found is $N/2$ giving a total run-time of $(N/2)\sqrt{8N/w} = \sqrt{2N^3/w}$ function evaluations.

The flaws in this analysis are as follows. The memory for holding distinguished points is empty at the start of the algorithm, and thus not full all of the time. Not all collisions are equally likely to occur. Not all distinguished points in the memory are equally likely to produce a collision. However, we may hypothesize from the flawed analysis that $\theta = c\sqrt{w/N}$ is the optimum proportion for some constant c, and that the overall run-time is $O(\sqrt{N^3/w})$ function evaluations. This hypothesis was confirmed empirically. For various values of θ, w, and N, simulations were performed to determine the number of distinct collisions found when using a version of f for various lengths of time. (These simulations were for the general technique as opposed to the specific examples of Section 2.) For multiple simulations with the same parameters (but different random input), the results showed very little variation. The number of evaluations of f per distinct collision found was a minimum for $\theta = 2.25\sqrt{w/N}$, and $10w$ trails generated per version of f. Because $10w$ triples are written to a memory which can hold only w triples, after the memory fills up, triples are simply overwritten. Using the parameters above in simulations, for $2^{10} \le w \le N/2^{10}$ the expected run-time to find the golden collision was found to be $2.5N^{3/2}/w^{1/2}$ iterations of f, and the expected number of accesses to the memory was $4.5N$. ❑

For double-DES, n is the size of the DES key space ($n = 2^{56}$). For limited Hamming weight exponents, $n = \binom{m}{t/2}$ for the preliminary version, and $n = \binom{m/2}{t/2}$ for the improved version. For the case of speeding up RSA computations using an untrusted server, n is the size of the space that the half-secret A (or B) is chosen from. Typical values of w depend on available memory. (Table 1 in Section 4 considers attacking double-DES with values of w implying memory size ranging from 2^{24} to 2^{44} bytes.)

4. Comparison to Previous Techniques

A simple approach to performing a meet-in-the-middle attack proceeds as follows. Compute $f_1(x)$ for all $x \in D$ and store the $(f_1(x), x)$ pairs in a table (using standard hashing on the $f_1(x)$ values to allow lookup in constant time). For each $y \in D$, compute $f_2(y)$ and look it up in the table. If there is a match, then the candidate pair of inputs x and y are tested to see if they are the correct inputs (a and b). This method requires, on average, $1.5n$ function evaluations and memory for n pairs, where $n = |D|$. For double-DES, this is $(1.5)2^{56}$ DES operations and 2^{56} stored pairs. Obviously, this is not a practical amount of memory. Suppose that available memory can hold only w pairs $(f_1(x), x)$. The attack can be modified as described by Even and Goldreich [6] (Amirazizi and Hellman [1] also consider this problem). Partition the space D into subsets of size w. For each subset, compute and store the pairs $(f_1(x), x)$ for all x in this subset. Then for each $y \in D$, compute $f_2(y)$ and look it up. The expected run-time for this memory-limited version of the attack is $(1/2)(n/w)(w + n) \approx n^2/(2w)$ function evaluations. A memory access is required after each function evaluation, and so the expected number of memory accesses is also about $n^2/(2w)$.

Comparing the run-time of this previous technique to $7n^{3/2}/w^{1/2}$ function iterations and $9n$ memory accesses (Proposition 1), the parallel collision search method of performing a meet-in-the-middle attack requires $0.07\sqrt{n/w}$ times fewer function evaluations and $n/(18w)$ times fewer memory accesses.

For concreteness, consider attacking double-DES where $n=2^{56}$ and the amount of memory needed for each triple in memory is 16 bytes. A comparison for different memory sizes is shown in Table 1.

Table 1. Example Improvement of Parallel Collision Search Method over Previous Techniques

Memory Size	Ratio of Encryptions previous techniques / new method	Ratio of Memory Accesses previous techniques / new method
$w=2^{20}$ (2^{24} bytes)	2^{91} / $2^{76.8}$ = 18000	2^{91} / $2^{59.2}$ = 3.8×10^9
$w=2^{25}$ (2^{29} bytes)	2^{86} / $2^{74.3}$ = 3200	2^{86} / $2^{59.2}$ = 1.2×10^8
$w=2^{30}$ (2^{34} bytes)	2^{81} / $2^{71.8}$ = 570	2^{81} / $2^{59.2}$ = 3.7×10^6
$w=2^{35}$ (2^{39} bytes)	2^{76} / $2^{69.3}$ = 100	2^{76} / $2^{59.2}$ = 1.2×10^5
$w=2^{40}$ (2^{44} bytes)	2^{71} / $2^{66.8}$ = 18	2^{71} / $2^{59.2}$ = 3.6×10^3

When a small number of processors is used, the total run-time is determined by the number of encryptions required as per Proposition 1. However, for a high degree of parallelism, the main limitation becomes accessing the memory which is common to all processors (particularly for the previous techniques which require a memory access after every function evaluation). Optimum performance for a given investment requires a balance between the memory size and number of processors; for larger memories, more processors should be used. Finding such an optimum for a given budget and fixed costs of processors, memory, etc., requires a detailed engineering design tailored for a particular problem, and is beyond the scope of the present paper.

For smaller memories, the amount of improvement is determined by the number of encryptions required; for large memories, the amount of improvement is determined by the number of memory accesses required. For the case where $w=2^{30}$ (or 16 Gbytes, which is considerable for an amateur, but not for a determined effort), the new method will be somewhere between 570 and 3.7×10^{6} times faster depending upon the type of processors and memory used to mount the attack.

5. Conclusion

Meet-in-the-middle attacks involve splitting an operation into two halves with a different secret quantity involved in each half of the operation. If each secret is chosen from a set of size n, and w memory elements are available to mount an attack, then a parallel collision search based method can be used to complete the attack in an expected heuristic time of $7n^{3/2}/w^{1/2}$ operations. This is $0.07\sqrt{n/w}$ times faster than previous techniques for meet-in-the-middle attacks. For the illustrative case of double-DES and an attacker with available memory for $w=2^{30}$ entries, the new method is between three and six orders of magnitude faster.

Acknowledgments

We would like to thank Burt Kaliski for early discussions regarding multiple encryption, Don Coppersmith for an observation that allowed golden collisions to be found about $\sqrt{2}$ times faster, and Andrew Odlyzko for reminding us about reference [10]. We would also like to thank anonymous members of the Crypto ’96 Program Committee whose comments contributed to an improved presentation of this material.

References

[1] H.R. Amirazizi and M.E. Hellman, “Time-Memory-Processor Trade-Offs”, *IEEE Transactions on Information Theory*, vol. 34, no. 3, May 1988.

[2] J. Burns and C.J. Mitchell, “Parameter Selection for Server-Aided RSA Computation Schemes”, *IEEE Transactions on Computers*, vol. 43, no. 2, Feb. 1994, pp. 163-174.

[3] “Data Encryption Standard”, National Bureau of Standards (U.S.), Federal Information Processing Standards Publication (FIPS PUB) 46, National Technical Information Service, Springfield, Virginia, 1977.

[4] D.E. Denning, *Cryptography and Data Security,* Addison Wesley, 1982.

[5] W. Diffie and M. Hellman, “Exhaustive cryptanalysis of the NBS Data Encryption Standard”, *Computer* vol.10 no.6 (June 1977) pp. 74-84.

[6] S. Even and O. Goldreich, “On the Power of Cascade Ciphers”, *ACM Transactions on Computer Systems*, vol. 3, no. 2, May 1985.

[7] P. Flajolet and A.M. Odlyzko, “Random Mapping Statistics”, *Lecture Notes in Computer Science 434: Advances in Cryptology - Eurocrypt ’89 Proceedings*, Springer-Verlag, pp. 329-354.

[8] R. Heiman, “A note on discrete logarithms with special structure”, *Lecture Notes in Computer Science 658: Advances in Cryptology - Eurocrypt ’92*, Springer-Verlag, pp. 454-457.

[9] M.E. Hellman, “A cryptanalytic time-memory trade-off”, *IEEE Transactions on Information Theory*, vol.6 (1980), pp. 401-406.

[10] B. Pfitzmann and M. Waidner, “Attacks on Protocols for Server-Aided RSA Computation”, *Lecture Notes in Computer Science 658: Advances in Cryptology - Eurocrypt ’92*, Springer-Verlag, pp. 153-162.

[11] J.M. Pollard, “A Monte Carlo method for factorization”, *BIT*, vol. 15 (1975), pp. 331-334.

[12] J.M. Pollard, “Monte Carlo Methods for Index Computation (mod p)”, *Mathematics of Computation*, vol. 32, no. 143, July 1978, pp. 918-924.

[13] P.C. van Oorschot and M.J. Wiener, “Parallel Collision Search with Application to Hash Functions and Discrete Logarithms”, *2nd ACM Conference on Computer and Communications Security*, Fairfax, Virginia, November 1994, pp. 210-218.