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

Presented by Ohad Lutzky, University of Haifa

Paul C. van Oorschot Michal J. Wiener

Cryptanalysis of Block Ciphers Seminar, Spring 2013

Outline

1. Double-DES and introduction to Meet-in-the-middle attacks
2. Formulating Meet-in-the-Middle attacks as Collision Search Problems
   - Additional examples for Meet-in-the-Middle attacks
   - Meet-in-the-middle as a collision search
3. Solving the Collision Search Problem
   - Parallel collision search
   - Comparison to previous techniques

DES and its short keys

- DES is considered weak, mainly because of its 56-bit key size
- A DES key took under 24 hours to break in 1999 (EFF's *Deep Crack*)
- Advanced chosen-plaintext attacks can provide even faster results

Double-DES

- Solution - use two keys k_1, k_2. Encrypt using $DES_{k_1}\left(DES_{k_2}(P)\right)$, decrypt using $DES_{k_2}^{-1}\left(DES_{k_1}^{-1}(C)\right)$.
- Key size is now $|k_1| + |k_2| = 112$ bits.
- Brute force attack now takes a reasonable 2^{113} DES encryptions (2^{112} keys, 2 encryptions each).

Basic meet-in-the-middle attack

Consider the following attack with one known plaintext pair (P, C):

1. For all possible keys k_1, compute $C_{k_1} = DES_{k_1}(P)$. Save the mapping $C_{k_1} \to k_1$ in a lookup table. [2^{56} time, 2^{56} space.]
2. For all possible keys k_2, compute $DES^{-1}_{k_2}(C)$. If this value is in the lookup table, use it to find the complementary k_1. [2^{56} time, no additional space].

Total cost: 2^{57} time (practical), 2^{56} space (about 64 petabytes, impractical.

Outline

1. Double-DES and introduction to Meet-in-the-middle attacks
2. Formulating Meet-in-the-Middle attacks as Collision Search Problems
   - Additional examples for Meet-in-the-Middle attacks
   - Meet-in-the-middle as a collision search
3. Solving the Collision Search Problem
   - Parallel collision search
   - Comparison to previous techniques

General description

A meet-in-the-middle attack involves:

- Two functions f_1, f_2
- Two inputs a, b s.t. $f_1(a) = f_2(b)$.
- The objective is to find a and b.
- There may be multiple solutions, but typically only one particular pair is "correct".

Double-DES, again

In this new notation,

- P, C are implicit constants
- $f_1(a) = DES_a(P)$
- $f_2(b) = DES_b^{-1}(C)$
- There may be other false pairs, but only one is correct. One additional pair (P', C') will generally suffice to determine correctness.

Discrete logarithm with low Hamming weight [1]

Solving the discrete logarithm can allow attacks on Diffie-Hellman key exchange, ElGamal encryption and DSA signatures.

- Let α be the generator of a cyclic group
- Let $y = \alpha^x$ be an element of that group, where $|x| = m$ bits and the Hamming weight of x is t.
- We wish to find x.

Discrete logarithm with low Hamming weight [2]

- Any x can be written as a sum of two m-bit values, each with Hamming weight $t/2$ (assume t is even).
- Let $n = \binom{m}{t/2}$ be the number of such values.
- Let $h : [0, n) \to 0, 1^m$ map integers to such values with Hamming weight $t/2$.
- Using $f_1(i) = \alpha^{h(i)}$, $f_2(j) = \alpha^{x-h(j)}$, we can describe this is a meet-in-the-middle attack.

Discrete logarithm with low Hamming weight [3]

- Efficiency can be improved by trying 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 $m/2$ trials, but with $n = \binom{m/2}{t/2}$.
- Randomly partitioning the bits, the expectation of amount of required trials can be reduced to $\sim \sqrt{\pi t(1 - t/m)/2}$.

Assisted RSA computations [1]

- Reminder: In RSA there is a private exponent d, a public exponent e, a public modulus $n = pq$. A message M is encrypted $M^e \mod n$, and a ciphertext C is decrypted $C^d \mod n$.
- Assume we need to compute an RSA decryption of x, i.e. $x^d \mod n$.
- Further assume our CPU is underpowered (e.g. a smart card), and requires assistance from an untrusted party.

Assisted RSA computations [2]

- We break down d as $\sum_{i=1}^{m} a_i d_i$, where $\{d_i\}_{i=1}^{m}$ is public and $\{a_i\}_{i=1}^{m}$ is a small secret.
- The untrusted server computes $\left\{x^{d_i}\right\}_{i=1}^{m}$ and sends it to the smart card.
- The smart card computes $\prod_{i=1}^{m} \left(x^{d_i}\right)^{a_i}$.

Assisted RSA computations [3]

- Let $A = (a_1, \ldots, a_{m/2})$, $B = (a_{m/2+1}, \ldots, a_m)$, $D = (d_1, \ldots, d_{m/2})$, $E = (d_{m/2+1}, \ldots, d_m)$,
- Then $d = A \cdot D + B \cdot E$.
- For RSA, $h = h^{ed} \mod n$ (encryption and decryption). This can be rewritten $h \equiv_n h^{e(A \cdot D + B \cdot E)}$, or $h^{e(A \cdot D)} \equiv_n h^{1-e(B \cdot E)}$.
- Using $f_1(x) = h^{e(x \cdot D)} \mod n$ and $f_2(x) = h^{1-e(x \cdot E)} \mod n$, this gives us a meet-in-the-middle attack.

Outline

1. Double-DES and introduction to Meet-in-the-middle attacks
2. Formulating Meet-in-the-Middle attacks as Collision Search Problems
   - Additional examples for Meet-in-the-Middle attacks
   - Meet-in-the-middle as a collision search
3. Solving the Collision Search Problem
   - Parallel collision search
   - Comparison to previous techniques

- Assume f_1, f_2 have the same domain (they only really need to have the same range)
- We have $f_1 : D \to R, f_2 : D \to R$, and seek pairs of inputs (i,j) s.t. $f_1(i) = f_2(j)$.
- If many pairs of inputs give collisions between f_1 and f_2, we need to test each to see if it's the "correct" pair we're looking for (a, b).

- For parallel collision search, we require a *single* function f that has equal range and domain, and has an ordinary collision which leads to the solution.
- Let $g : R \to D \times \{1, 2\}$ be a function which maps an element from R to D, along with a bit selecting between f_1, f_2. (Assume $|R| \geqslant 2|D|$).
- Define $f : D \times 1, 2 \to D \times 1, 2$ as $f(x, i) = g(f_i(x))$.
- Therefore, $f_1(a) = f_2(b)$ is equivalent to $f(a, 1) = f(b, 2)$, which is the collision we seek.

Outline

1. Double-DES and introduction to Meet-in-the-middle attacks
2. Formulating Meet-in-the-Middle attacks as Collision Search Problems
   - Additional examples for Meet-in-the-Middle attacks
   - Meet-in-the-middle as a collision search
3. Solving the Collision Search Problem
   - Parallel collision search
   - Comparison to previous techniques

Golden collisions

- There are many pairs i, j s.t. $f(i,1) = f(j,2)$
- Among them is a unique collision pair $f(a,1) = f(b,2)$ which solves the meet-in-the-middle problem.
- A very large number of collisions in f must be found so that $((a,1),(b,2))$ is among them, therefore we call it the *golden collision*.
- For double-DES, we seek the correct key pair $(k_1, 1), (k_2, 2)$.
- For the discrete logarithm search shown before, many collisions solve the original problem, and such a collision is usually found quickly.

Ordinary parallel collision search

1. Given a function $f : S \to S$, choose a *distinguishing property* which distinguishes a proportion θ of the elements of S.
   - e.g., $\theta = 2^{-10}$ when elements with 10 leading zero bits are distinguished
2. Choose an element $x_0 \in S$ and produce the sequence of points $x_i = f(x_{i-1})$. Label these $x_1, x_2, \ldots$.
3. Keep iterating until a distinguished point x_d is reached.
4. Store the triple (x_0, x_d, d) in a table.
5. Repeat for many x_0 values.
6. If the same x_d value shows up twice, their trails have collided.
7. Step the trails forward from their respective x_0 to find the collision.

Analysis

- Let $N = |S|$
- One expects to perform $\sqrt{\pi N/2}$ iterations of f before a collision.
- This can occur over multiple processors
- As available memory fills, the probability of finding a collision grows quadratically.
- Finding k collisions is expected to take $\sqrt{\pi k N/2}$ iterations of f.

Modification for golden collisions [1]

- Solving the meet-in-the-middle requires finding the *golden* collision
- There are $\binom{N}{2} \approx N^2/2$ pairs of inputs
- If f behaves randomly, one expects that there are about $N/2$ collisions
- One might expect that this means you need $\sqrt{\pi N/2}$ iterations of f
- However, iterations required to locate each detected collision by stepping the trails do not decrease
- Furthermore, not all collisions are equally likely
- Solution: Limit the number of collisions sought using a particular f. If the golden collision is not found, recreate f (by choosing a new g reverse-mapping[3]) and repeat.

Modification for golden collisions [2]

- Assume given (shared) memory can hold w triples.
- Heuristically, $\theta \approx 2.25\sqrt{w/N}$ is optimal, and one should generate $10w$ trails per version of f.
- The expected number of f iterations is $7n^{3/2}/w^{1/2}$
- The expected number of memory accesses is $4.5N = 9n$.
- For double-DES, n is the size of the DES key space (2^{56})
- For limited Hamming weight exponents, $n = \binom{m/2}{t/2}$ (for the improved version)
- For untrusted-assisted RSA, n is the size of the space that A is chosen from.

Outline

1. Double-DES and introduction to Meet-in-the-middle attacks
2. Formulating Meet-in-the-Middle attacks as Collision Search Problems
   - Additional examples for Meet-in-the-Middle attacks
   - Meet-in-the-middle as a collision search
3. Solving the Collision Search Problem
   - Parallel collision search
   - Comparison to previous techniques

Memory-limited basic meet-in-the-middle

- The original meet-in-the-middle attack shown is not feasible with respect to memory usage. Solution:
  1. Partition the space D into subsets of size w.
  2. For each subset, compute and store the pairs $(f_1(x), x)$ for all x in this subset.
  3. For each $y \in D$, compute $f_2(y)$ and look it up.
- Expected run-time is $(1/2)(n/w)(w+n) \approx n^2/(2w)$ function evaluations and memory accesses.

Comparison

- The improved technique takes $7n^{3/2}/w^{1/2}$ function iterations ($0.07\sqrt{n/w}$ times fewer than the memory-limited simple MITM).
- ...and $9n$ memory accesses. ($n/(18w)$ times fewer).

Comparison

Table: Example improvement of Parallel Collision Search Method over previous techniques

Memory Size	f iteration ratio	Mem. access ratio
$w = 2^{20}$ (2^{24} bytes)	$2^{91}/2^{76.8} = 18000$	$2^{91}/2^{59.2} = 3.8 \times 10^9$
$w = 2^{25}$ (2^{29} bytes)	$2^{86}/2^{74.3} = 3200$	$2^{86}/2^{59.2} = 1.2 \times 10^8$
$w = 2^{30}$ (2^{34} bytes)	$2^{81}/2^{71.8} = 570$	$2^{81}/2^{59.2} = 3.7 \times 10^6$
$w = 2^{35}$ (2^{39} bytes)	$2^{76}/2^{69.3} = 100$	$2^{76}/2^{59.2} = 1.2 \times 10^5$
$w = 2^{40}$ (2^{44} bytes)	$2^{71}/2^{66.8} = 18$	$2^{71}/2^{59.2} = 3.6 \times 10^3$

Practical notes

- When a small number of processor is used, total run-time is determined by number of f iterations.
- For high parallelism, the main limitation is shared memory amount and speed.
- Finding the optimum requires detailed engineering, tailored for a particular problem.

Summary

- Meet-in-the-middle attacks involve splitting an operation into two halves, with two secret quantities.
- When each secret is chosen from a set of size n, and w memory is available, a parallel collision search can be used with much higher efficiency than “classic” meet-in-the-middle attacks.
- This method can make good use of practical memory availability and parallel processing.

Created with

- LaTeX-BEAMER
  - Using `mathserif`, math font is `eulervm`
- Vim