Generic Cryptanalytic Attacks

Orr Dunkelman

Computer Science Department
University of Haifa, Israel

January 28th, 2014

Outline

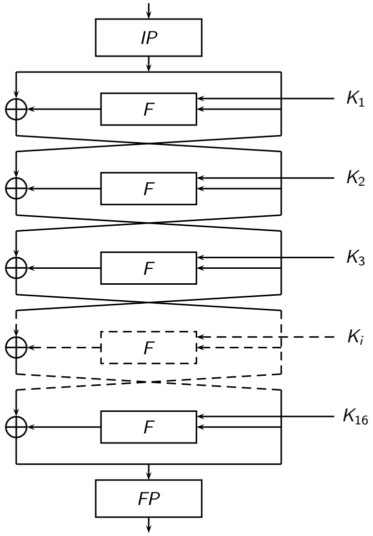
Block Ciphers

- One of the most basic cryptographic algorithms.
- A symmetric key algorithm (both sides hold secret information).
- Is a transformation of blocks of bits (of size n) into new blocks of bits (usually of the same size). Formally: $E : \{0,1\}^n \times \{0,1\}^k \mapsto \{0,1\}^n$ or $E_k : \{0,1\}^n \mapsto \{0,1\}^n$.
- To deal with more (or less) data, some mode of operation is used (ECB, CBC, counter mode, etc.).

The Data Encryption Standard

- Designed by IBM at the mid 70's.
- Feistel block cipher with 16 rounds.
- 64-bit block size, 56-bit key size.
- The round function accepts 32-bit input and 48-bit subkey.

Outline of DES



IP and *FP*

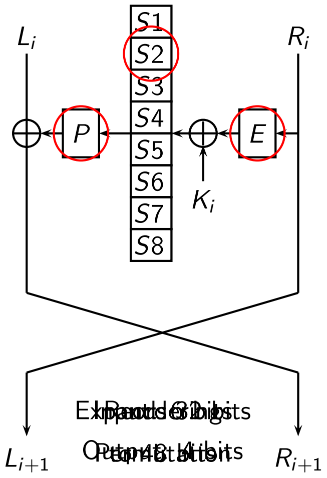
IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

FP $(=\mathrm{IP}^{-1})$

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

DES' F-function



Expands 32 bits [illegible]

Output 4 bits Permutation 48 bits [illegible]

DES' F-function (cont.)

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

E

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

DES' F-function (cont.)

$S1$

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S2$

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

DES' F-function (cont.)

$S3$

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$S4$

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

DES' F-function (cont.)

$S5$

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

$S6$

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

DES' F-function (cont.)

$S7$

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

$S8$

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

DES' F-function (cont.)

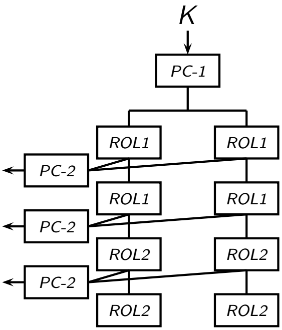
Beware! The S-boxes are given (as in the FIPS) in a very confusing manner. The MSB and the LSB of the input determine the row in the table, and the middle 4 bits determine the column. For example, this table shows where the entry corresponding to the input is:

Location of entries in the previous tables

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63

DES' Key Schedule Algorithm

- The key is divided into two registers C and D (28-bit each).
- Each round both registers are rotated to the left (1 or 2 bits).
- 24 bits from C are chosen as the subkey entering $S1, S2, S3, S4$.
- 24 bits from D are chosen as the subkey entering $S5, S6, S7, S8$.



Round	1	2	3	4	5	6	7	8
Rotation	1	1	2	2	2	2	2	2
Round	9	10	11	12	13	14	15	16
Rotation	1	2	2	2	2	2	2	1

Basic Inversion Attacks

- A function $f : \{1, 2, \ldots, N\} \mapsto \{1, 2, \ldots, N\}$ is fixed.
- A value $y = f(x)$ is given to the attacker who needs to find x.
- This problem can model finding keys of an encryption function.
- For example, $f(k) = E_k(P)$ for some pre-determined plaintext P.
- Or $h(\text{input})$ for hash functions.

Basic Inversion Attacks (cont.)

Simple approaches for solution:

- Exhaustive search — the attacker computes for each i the value of $f(i)$ and stops once $y = f(i)$.
- Table attack/Dictionary attack — the attacker precomputes once all $f(i)$, and stores in a table $(f(i), i)$ sorted according to $f(i)$.

- Exhaustive search — Precomputation $= 0$; Memory $= 0$; Time $= N$.
- Table — Precomputation $= N$; Memory $= N$; Time $= 1$.

Some Variants of Exhaustive Search

- If P CPUs are available, we can let each CPU run over $1/P$ of the search space.

Time $= N/P$ (in real time).

- Sometimes, it is possible to evaluate $f(\cdot)$ on several points simultaneously (bit-slicing, same subkey in the first round, etc.)

Does not affect asymptotic time, but actual time.

- Sometimes, it is possible to partially-evaluate $f(\cdot)$, and only if the partial evaluation succeeds, compute the full evaluation.

Does not affect asymptotic time, but actual time.

- For a specific $f(\cdot)$, it is usually more efficient to build dedicated hardware (FPGA/ASIC).

Saves on the money/time ratio.

Diffie and Hellman's DES Machine

- Shortly after the introduction of DES, Diffie and Hellman analyzed the 56-bit key length.
- Machine with 1,000,000 chips (in 64 racks), each tests a DES key in a microsecond.
- Connecting it all (and taking some overhead), their machine could find a DES key every half a day on average for 20,000,000$.
- If you run the machine for 5 years, the cost of finding a key is expected to be 5000$.

For more information: W. Diffie, M. Hellman, *Exhaustive cryptanalysis of the NBS Data Encryption Standard*, Computer, Vol. 10, No. 6, pp. 74–84, June 1977.

Exhaustive Search Example — DES Challenges

- In 1997 RSA Labs started a DES challenge, they published a plaintext and a ciphertext, and offered a prize for the first one finding the key.
- The DESCALL project was used to solve the first challenge in a distributed manner (90 days).
- In 1998, the second DES challenge was launched.
- distributed.net project found the key in 39 days.
- The third challenge (and last) was cracked using the DES Cracker (by EFF).
- 22 hours to find a random key of 56-bit (full exhaustive search was expected to take 56 hours).

Exhaustive Search Example — DES Cracker

- DES cracker consisted of 1,536 custom-designed ASIC chips at a cost of material of around 250,000$ and could search 88 billion keys per second.
- That is more than 2^{36} keys per second.
- A full exhaustive search requires about 819,000 seconds (slightly less than 9.5 days).
- Actually, the majority of the cost is the design cost and fabrication of the first unit.
- A second machine would be significantly cheaper.
- Today, for the same price, one should expect 512-times the computational power for the same cost (or the same computational power for slightly less than 500$) following Moore's law.

Exhaustive Search Example — DES Cracker (cont)

- DES Cracker was used in the third challenge, and found the key in 56 hours.
- In the amended third challenge (issued two weeks later), the DES cracker was integrated into the Distributed.Net project. This challenge was solved in about 22 hours.
- Conclusion: 56-bit key is insecure (1997).
- Conclusion 2: Today, 64-bit key is insecure.
- Conclusion 3: For real security, move to 80-bit security.

Other Technologies

FPGA based DES-cracker: COPACOBANA ([G+07,G+08]).

- A board with 120 slots for FPGAs.
- Costs 10,000 Euros, searches 65.3 billion keys per second.

Sony PlayStation:

- Used for (the really cool) attacks on MD5.
- A PS3 machine at 400$ could perform 175 million MD5 computations per second.

GPU cards:

- For only 245$, an ATI HD 4850 X2 can compute 1634 million MD5 per second.

The Full Cost of Cryptanalytic Attacks

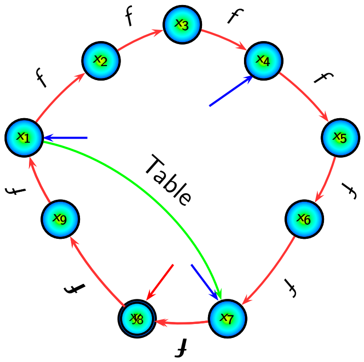
- One should be very careful when parallelizing algorithms.
- A problem broken to x small tasks, each using the same huge memory, actually do not yield a time improvement by a factor of x.
- Moreover, one should consider the cost of the switching and wiring between the CPUs and memory.
  - For exhaustive search — not really an issue.
  - For other cryptanalytic attacks: n processes, accessing a memory of n values **each cycle**, requires wiring of $\Omega(n^{3/2})$.

For more info: M. J. Wiener, *The Full Cost of Cryptanalytic Attacks*, Journal of Cryptology, Vol. 17, No. 2, pp. 105–124, 2004.

Hellman's Time-Memory Tradeoff Attack [H80]

Hellman suggested a method to trade the time and the memory complexities.

- Assume that f is a permutation, such that it has one huge cycle covering all values.
- Precomputation: pick at random point x_1, compute $x_{i+1} = f(x_i)$, and store the $\sqrt{N}$th values (i.e., $x_1, x_{\sqrt{N}+1}, x_{2\sqrt{N}+1}, \ldots$).
- Online phase: given y, compute $f^j(y)$ until a stored $x_{i\sqrt{N}}$ is encountered. Obtain $x_{(i-1)\sqrt{N}+1}$ from the table, and apply f to it until y is obtained



Hellman's Time-Memory Tradeoff Attack (cont.)

- The suggested attack has a precomputation of $P = N$, storage of $M = \sqrt{N}$, and online time, $T = \sqrt{N}$.
- But this works only if f induces a single cycle!
- Actually if f is a permutation a similar attack works, which might require even less online computation or even less memory (or both).
- The idea is to store every $\sqrt{N}$th point on the cycle.
- Once the cycle is smaller than $\sqrt{N}$ there is no need to store any point on it, as you can start from y, and find its preimage (predecessor in the graph) in less than $\sqrt{N}$ operations.

Hellman's TM Attack on Random Functions

- First trial:
  - Precomputation: Take $m = \sqrt{N}$ starting points x^i and from each such point, generate a sequence of $\sqrt{N}$ values, and store the obtained end points (y^i, x^i).
  - Online phase: Given y, start computing f on it, until hitting one of the end points. Retrieve from the table the value of the corresponding start point x^i, and compute forward until $x = f^{-1}(y)$ is found.

$$\begin{tikzcd}
x^1 \arrow[r, "f"] & x_1^1 \arrow[r, "f"] & x_2^1 \arrow[r, "f"] & \dots \arrow[r, "f"] & y^1 \\
x^2 \arrow[r, "f"] & x_1^2 \arrow[r, "f"] & x_2^2 \arrow[r, "f"] & \dots \arrow[r, "f"] & y^2 \\
x^3 \arrow[r, "f"] & x_1^3 \arrow[r, "f"] & x_2^3 \arrow[r, "f"] & \dots \arrow[r, "f"] & y^3 \\
 & & \vdots & & \\
x^m \arrow[r, "f"] & x_1^m \arrow[r, "f"] & x_2^m \arrow[r, "f"] & \dots \arrow[r, "f"] & y^m
\end{tikzcd}$$

Hellman's TM Attack on Random Functions (cont.)

- The function f is random. Thus, there are collisions between the chains!
- From the collision, both chains "evolve" together, and thus cover the same nodes (values).
- Thus, the chains are expected to cover much less than N nodes.
- Adding more chains, will not solve the problem (each new chain will cover very few new nodes before a collision is found).
- Finally, because f is random, some nodes do not have predecessors (about $1/e$ of the space).

Hellman's TM Attack on Random Functions (cont.)

- Hellman solved the problem by using **different functions!**
- Let f_i be some small tweak of f, such that inverting f_i is like inverting f (for example $f_i(x) = f(x) \oplus i$).
- For each of the t functions f_i, pick m random starting points, and compute chains of length t.
- For each function, store the values $(end, start)$ in a table.
- In the online phase — try to compute $f_i^j(y)$ for every $i = 1, \ldots t$, and $j = 1, \ldots, t$, until one of the end points is found. Go to the corresponding start point, and find the predecessor of y.

Hellman's TM Attack on Random Functions (cont.)

- Preprocessing — N. Memory — t tables, of m blocks each, total of mt. Online time — t^2 applications of f, and t^2 table accesses. As we want to cover $O(N)$ values, we need $mt^2 \approx N$, i.e.,

$$TM^2 = N^2.$$

- A common point on the curve is $M = T = N^{2/3}$.
- Of course, if $mt > N$ or $t^2 > N$, then the attack is inferior to other generic attacks.
- There are some small technicalities concerning the false alarms (hitting an end point, even though the value is not covered by the chain), but most of the time it is OK.

Choosing the Function f

- Consider the case of a block cipher.
- When suggesting a function to invert, the function often picked is $f(K) = E_K(P)$ from some pre-determined plaintext P.
- When the block size is equal to the key size, this function has the "right" size.
- But what if the block size is not equal to the key size?
- If $|P| > |K|$, then $E_K(P)$ is longer than K, and a simple solution is to drop some bits of the output (beware of false alarms!).
- If $|P| < |K|$, one can define the function $f(K) = E_K(P_1)||E_K(P_2)$.

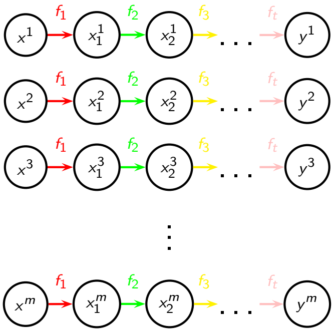
Reducing Storage Accesses using Distinguished Points

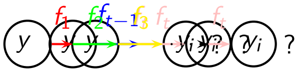
- Each table contains m pairs of (end point, start point).
- After each computation of f (or f_i), we need to access a table.
- Accessing large tables (especially if m is larger than the size of the RAM) takes time, sometimes greater than of actually computing f.
- A solution by Rivest, to use the concept of *distinguished points*.
- Instead of each chain ending always after t iterations of f_i, we let the chain continue until an easily identifiable point is achieved (e.g., $\log_2(t)$ least significant bits are 0).

Reducing Storage Accesses using Distinguished Points (cont.)

- On average, the same number of points is covered. But instead of t^2 table accesses, we can use only t of these (whenever a distinguished point is encountered).
- Note that the other parameters are the same! Specifically, the number of f invocations and/or memory size is the same.
- Really good for hardware acceleration and parallelization [SRQL02].

Rainbow Tables [O03]





- As noted before, it might be beneficial to reduce the number of accesses to the table.
- Oechslin suggested the concept of rainbow tables, without the need of distinguished points.
- Instead of having t multiple tables (each with m starting points), we start with mt starting points.
- For each point x_i, we evaluate $y_i = f_t(f_{t-1}(\ldots f_2(f_1(x_i))\ldots))$, and store (y_i, x_i).
- In the online phase: given y, check $f_t(y)$, $f_t(f_{t-1}(y))$, ..., as end

Rainbow Tables (Analysis)

- This method has the advantage of reducing false alarms, and it is claimed to achieve the curve $N^2 = 2TM^2$.
- This is partially true, but due to some technicalities, [BBS06] showed that rainbow tables are less favorable (mostly due to a larger memory block).

Estimation of success rate:

- Real coverage of each table — 80% [H80].
- Independence between tables — 55% success rate (for $mt^2 = N^2$).
- Finding optimal tables/coverage — [BPV98,SRQL02,KM96,KM99].
- Rainbow tables' success rate — 99% (but for a different problem).
- Stateful Random Graph model [BBS06]:

$$T \geq \frac{N^2}{128 M^2 \ln N}.$$

under randomness assumption of f.

The Fiat-Naor Time-Memory Attack

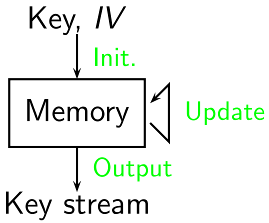
- Useful for non-random functions as well.
- A method with the tradeoff curve $TM^2 = N^3 \cdot q(f)$, where $q(f)$ is the probability that two randomly chosen inputs to f collide.
- Or a method for any function with the curve $TM^3 = N^3$ that works for any function.
- In both cases, the preprocessing is $O(N)$.

Time-Memory-Data Tradeoff Attacks

- Sometimes, the adversary is given several data points $y_1 = f(x_1), y_2 = f(x_2), \ldots$, and he needs to find only **one** of the preimages.
- In block cipher cryptanalysis — this means related-key attacks.
- For hash functions — in multiple target attacks.
- For stream ciphers — very interesting.

Stream Ciphers

- A stream cipher is a symmetric key encryption algorithm.
- It contains an internal memory, and is composed of three algorithms:
  - Initialize — Accepts a key and IV, and initializes the internal memory.
  - Update — Given the current memory state (and sometimes the plaintext) updates the internal memory.
  - Encrypt — Given the internal memory state and the plaintext, produces the ciphertext.



TMDTO Attacks on Stream Ciphers

Babbage & Golic independently developed an attack of

- Let N be the internal state space, and let $D = N/M$.
- Pre-processing: $O(M)$ operations.
- Online time: $T = D$ disk accesses.
- Tradeoff curve $D \cdot M = N$, i.e., $N = TM$, and $T, M, D < N$.

[Pick M internal states, compute the stream produced by it; after $D = N/M$ data points the stream can be found]

TMDTO Attacks on Stream Ciphers (cont.)

- Biryukov & Shamir showed that one can just apply Hellman's attack with $1/D$ of the coverage.
- Then, the attacker tries for each of the D given values to invert the function.
- Preprocessing: $P = O(N/D)$
- Online computation: T
- Obtained tradeoff curve: $N^2 = TM^2D^2$, assuming that $N > T \geq D^2$.
- Later [BSW00] showed that functions of low sampling resistance, can admit the curve $N^2 = TM^2D^2$ with $T > D$.

TMDTO Attacks on Stream Ciphers (cont.)

- It is also possible to analyze the function that maps (key, IV) pair into n-bit output stream.
- Each stream under a (key, IV) pair suggests one data point.
- Resulting curve: $N^2 = TM^2D^2$ and the restriction $T < N$, $MD < \sqrt{N}$, $T > D^2$.
- [DK08]: Out of the 2^{iv} possible IV, pick $2^{iv}/D$ IVs, and build for each chosen IV_i, the tables for the function $f : \{0,1\}^k \times IV_i \rightarrow$ output.
- Wait until one of the used IVs (in the real world) is one of the IV_i, and deduce the key using the time-memory attack. The obtained curve: $N^2 = TM^2D^2$. Restrictions: $N \geq T; N \geq M; N \geq D; T \geq D$.

Double DES

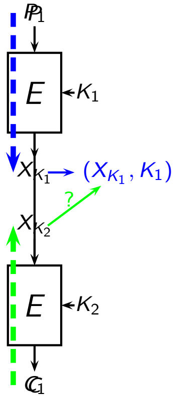
- DES has a key size of 56 bits, which (along with the complementation property) made DES look less secure than its predecessor Lucifer.
- One possible solution is to encrypt under two keys, i.e., define $DES^2_{K_1,K_2}(P) = DES_{K_2}(DES_{K_1}(P))$.
- The new key size is 112 bits, and exhaustive search now should take 2^{112} operations...

Meet-in-the-Middle Attack on Double-DES

- In 2DES, $C = DES_{K_2}(DES_{K_1}(P))$ which can be reformulated as

$$DES_{K_2}^{-1}(C) = DES_{K_1}(P).$$

- The attack:
  1. For all K_1, compute $X_{K_1} = DES_{K_1}(P)$, and store in a table (X_{K_1}, K_1).
  2. For all K_2, compute $X_{K_2} = DES_{K_2}^{-1}(C)$ and check whether X_{K_2} is in the table. If so, test (K_1, K_2) on a hash different plaintext/ciphertext pair.
  3. We expect $2^{56} \cdot 2^{56} \cdot 2^{-64} = 2^{48}$ additional tests.



Meet-in-the-Middle Attack on Double-DES (cont.)

- The time complexity of this attack is $2^{56} + 2^{56} + 2 \cdot 2^{48} \approx 2^{57}$ 1-DES encryptions.
- and the memory complexity is 2^{56}.
- There are easy tradeoffs for less memory (M) which takes time $2^{112}/M$.

Memoryless Meet in the Middle Attack

- Let (P_1, C_1) and (P_2, C_2) be two plaintext/ciphertext pairs for double encryption.
- Define $f_1(K) = E_K(P_1)$ and $f_2(K) = D_K(C_1)$.
- The correct key (K_1, K_2) is one for which $f_1(K_1) = f_2(K_2)$.
- To solve this problem, one can run a memoryless collision algorithm.
- If the block size is equal to the key size, **a** collision can be found in time $O(\sqrt{|K|})$ (and checked using (P_2, C_2)), and there are $O(|K|)$ collisions.
- Hence, after about $O(|K|^{3/2})$ one can find (K_1, K_2) with no additional memory.

For more information: P. C. van Oorschot, M. J. Wiener, *Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude*, CRYPTO 1996: 229–236.

Meet-in-the-Middle Attack on Triple-DES

- There are several variants which use three DES encryption. One of them is the following triple-DES:

$$TDES_{K_1,K_2,K_3}(P) = DES_{K_3}(DES^{-1}_{K_2}(DES_{K_1}(P)))$$

- Of course, the same meet-in-the-middle attack can still be applied:
  1. For all K_1, compute $X_{K_1} = DES_{K_1}(P)$, and store in a table (X_{K_1}, K_1).
  2. For all K_2, K_3, compute $X'_{K_2,K_3} = DES_{K_2}(DES^{-1}_{K_3}(C))$ and check whether X'_{K_2,K_3} is in the table. If so, test (K_1, K_2, K_3) on different plaintext/ciphertext pairs.
  3. We expect $2^{56} \cdot 2^{56} \cdot 2^{56} \cdot 2^{-64} = 2^{104}$ additional tests.
- The running time is 2^{112} and the memory requirements are 2^{56} blocks of memory.

Meet-in-the-Middle Attack on 2Key-Triple-DES

- Consider the 2K-3DES mode suggested by IBM:

$$2K - TDES_{K_1,K_2}(P) = DES_{K_1}(DES^{-1}_{K_2}(DES_{K_1}(P)))$$

- The MitM attack on it is a chosen plaintext one [MH81].
- The idea is to find a plaintext for which $DES_{K_1}(P) = 0$, and then play with the MitM attack a bit.

Meet-in-the-Middle Attack on 2K-Triple-DES (cont.)

- The attack is as follows:
    1. For any i, compute $P_i = DES_i^{-1}(0)$, and ask for its encryption.
    2. Let $C_i = DES_{K_1}(DES_{K_2}^{-1}(DES_{K_1}(P_i)))$.
    3. For each C_i compute $X_i = DES_i^{-1}(C_i)$.
    4. Store (X_i, i) in a table.
    5. For each K_2, compute $X' = DES_{K_2}^{-1}(0)$, and whether X' is in the table. If so, check (i, K_2) as the key.
- The attack can be optimized by noting that Steps 1 and 5, perform the same thing (using the correct data structure).

Other Resources

There are several very interesting papers you may wish to consider:

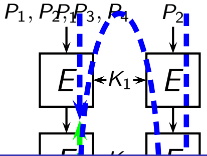
- Paul C. van Oorschot, Michael J. Wiener, *A Known Plaintext Attack on Two-Key Triple Encryption*, EUROCRYPT 1990: 318-325.
- Stefan Lucks, *Attacking Triple Encryption*, FSE 1998: 239-253.

Analyzing 4-Encryption

Consider the case of 4-Encryption:

$$C = E_{K_4}(E_{K_3}(E_{K_2}(E_{K_1}(P))))$$

Standard MitM attack can take 2^{3n} time with 2^n memory, or 2^{2n} time with 2^{2n} memory.



Analysis

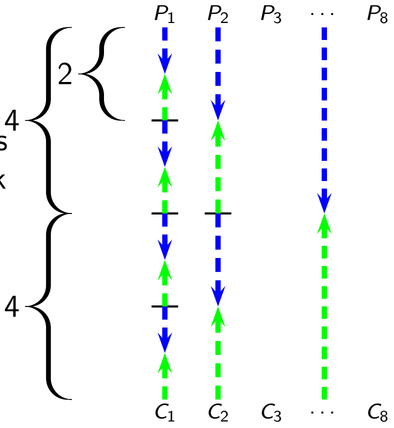
- For each X_1^2 guess, we did two MitM attacks of 2^n time and memory.
- Then, we had another MitM of 2^n time and memory.
- So in total — time complexity is 2^{2n}, and memory complexity is 2^n.

Extending the Basic Attack

- Obviously, enjoying the 2^n gain when attacking r-encryption with $r \geq 4$.
- Just guess the $r-4$ last keys, and apply the 4-encryption attack.
- Of course, the question is whether we can do better...
- Namely, can we gain more given that we already gained something?

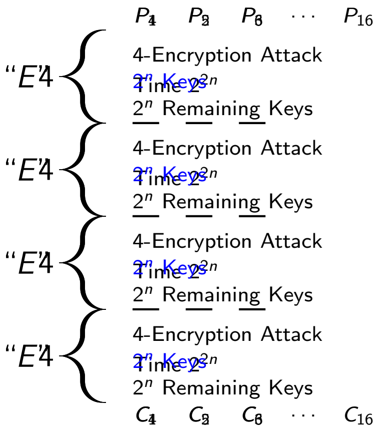
The LogLayer Algorithm

- A straightforward extension is the LogLayer algorithm.
- When attacking r-encryption, we guess $r/2-1$ internal states just after round $r/2$, and attack each half independently.
- With 2^n memory, the running time is $2^{n(r-\log(r))}$.
- The "gain" sequence is: 2,4,8,16,32,. . . .



The Square Algorithm

- A different improvement that relies on symmetry.
- Consider 16-Encryption:
- Now, we need to attack "4-Encryption" again.
- The complexity is $2^{n(r-\sqrt{r}+1)}$.
- The "gain" sequence is: 2,4,9,12,16,25,36,. . . .

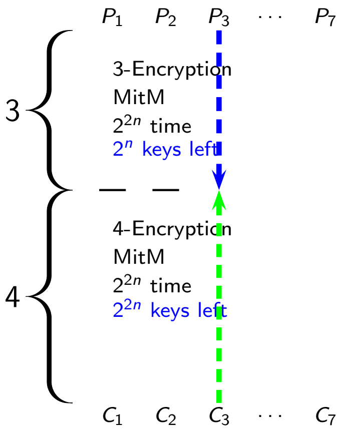


Why Asymmetry is Important in Symmetric-Key Attacks

- The shared characteristic of both LogLayer and Square is the fact that they are “symmetric” in nature.
- They do not distinguish between the “forward” direction stored in the table, and the “backward” direction which is checked in the table.
- In reality, they are different. The “backward” direction can be generated “on-the-fly”.

The Best Algorithm (we could find)

- A different improvement relies on symmetry.
- Consider 7-Encryption:
- We access the table with the 2^{2n} suggested keys.
- The idea is to balance the complexity of the attack (on the second half) with the number of "solutions".
- The "gain" sequence is: 2,4,7,11,16,22,29,. . . .



Attacking r-Encryption

1. Guess as many keys as needed to reduce the scheme to a "magic number" (from the gain list).
2. *Dissect* the remaining encryptions:
    1. For the ith magic number, guess $i-1$ internal states after round i.
    2. Attack the first i rounds, obtain 2^n keys, and construct a table.
    3. Attack the remaining rounds, and access the table to find full key candidates.

We call this technique "**Dissection**".

Dissection using Parallel Collision Search

- Just like in the PCS algorithm for double-encryption, to use the PCS we need to divide the full encryption function into two.
- This is done be defining

$$F^{upper} : (K_1, \ldots, K_{r/2}) \mapsto (X_1^{r/2}, \ldots, X_{r/2}^{r/2}) \text{ and}$$
$$F^{lower} : (K_{r/2+1}, \ldots, K_r) \mapsto (X_1^{r/2}, \ldots, X_{r/2}^{r/2}).$$

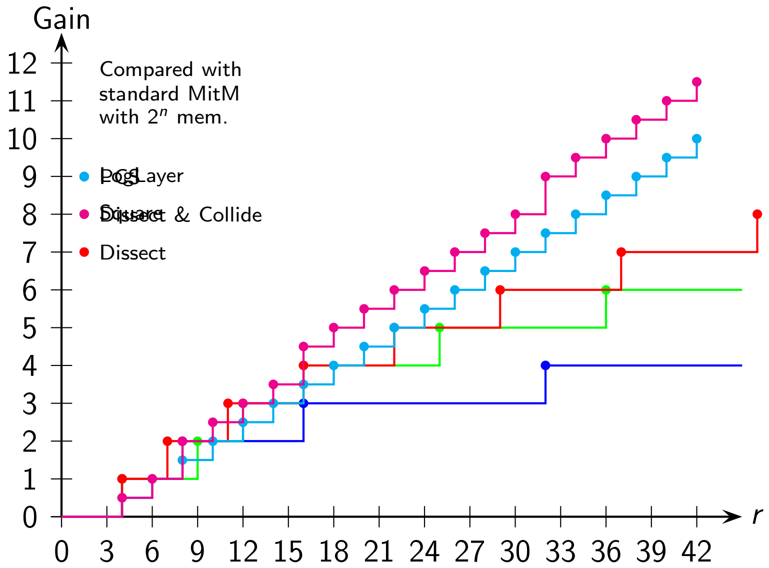
- Given Floyd's algorithm (or Nivasch's or Brent's or ...), find collisions between the two functions.
- Actually, we can use Hellman's TMTO attacks to find 2^n collisions simultaneously in time $2^{(r/4+1/2)n}$.
- After $2^{(r/2)n}$ such collisions, we expect the right one to show up.

Dissection using Parallel Collision Search (cont.)

- The key idea is to compute the functions F^{upper} and F^{lower} using dissection and the extra available memory.
- Namely, we "agree" on the output of the functions, thus, restricting them to a smaller space.
- For 8-Encryption:

$F^{upper} \quad : \quad (K_1, K_2, K_3, K_4) \quad \mapsto \quad X_1^4, X_2^4, X_3^4, X_4^4$
$\tilde{F}^{upper} : X_1^2 \mapsto X_4^4$
Uses $P_1, \ldots P_4$ Uses $P_1, \ldots P_4$ and
X_1^4, X_2^4, X_3^4
$F^{upper} \quad : \quad (K_5, K_6, K_7, K_8) \quad \mapsto \quad X_1^4, X_2^4, X_3^4, X_4^4$
$\tilde{F}^{upper} : X_1^6 \mapsto X_4^4$
Uses $C_1, \ldots C_4$ Uses $C_1, \ldots C_4$ and
X_1^4, X_2^4, X_3^4
Takes $O(1)$ to evaluate Takes $O(2^n)$ to evaluate

The Gains of the Algorithms



Questions?

Thank you for your attention!