Cryptanalytic Time-Memory Tradeoff

See:

Known Attacks

Denote the key size of a block cipher by $k$.

Then, there are two universal very simple attacks against any cipher:

1. Exhaustive search: $T = 2^k$ time, $S = 1$ memory

2. Table attack: $T = 1$ time, $S = 2^k$ memory
   (but $2^k$ precomputation, assumes knowing the plaintext in advance)

Both attacks require huge resources, even against DES.
Many problems in computer science have time-memory tradeoffs, in which the computation time can be reduced for the price of using larger memory (or the amount of memory can be decreased for the price of slowing the attack).

Can we find a cryptanalytic time-memory tradeoff?

Can we find other attacks with both $T, S < 2^k$?

Both attacks satisfy $TS = 2^k$.

Can we find other solutions to $TS = 2^k$?

Can we find an attack in which $T = S = 2^{k/2}$?

The attack that we will see finds some other tradeoff, that can work also for finding preimages of hash functions.
Time-Memory Tradeoff (cont.)

We assume that there is some one-way function $f(x)$ that we want to invert for some given output $y' = f(x')$, where $x'$ is unknown.
The Case of a Block Cipher

The attack assumes that the plaintext is known in advance, since it is required for the precomputation. Thus, this is basically a chosen plaintext attack, but it can also be applied as a known plaintext or even a ciphertext only attacks if there is some sufficiently frequent plaintext block (such as “Password”).
The Case of a Block Cipher (cont.)

Assume that the encryption function is

\[ C = E_K(P). \]

For a fixed plaintext block \( P_0 \), define

\[ f(K) = R(E_K(P_0)), \]

where \( R \) is some transformation from the ciphertext space to the key space (in DES: from 64 bits to 56 bits). \( R \) can be any transformation. Even if \( R \) only chooses \( k \) (56) bits of \( E_K(P_0) \) in some order, we have \( 64!/8! \approx 3 \cdot 10^{84} \) possibilities.

\( f(K) \) is an one-way function, otherwise the cipher \( E \) would be easy to break.

We will actually attack the one-way function \( f \).
A Simple Case: $f$ is a Permutation

If $f$ is a permutation (i.e., 1-1 and onto), then each value $x$ has a unique successor $f(x)$ and a unique predecessor $f^{-1}(x)$, and the values are divided into cycles.

Let $t = \lfloor \sqrt{n} \rfloor$.

In each cycle, select a starting point $x_0$, and let $p$ be the period of the cycle. Assume without loss of generality that $p|t$. Compute

$$x_t = f^t(x_0), x_{2t} = f^t(x_t), \ldots, x_p = x_0 = f^t(x_{p-t}).$$

Keep all the $x_{kt}$'s of all the cycles in a table, where for each $x_{kt}$ it is easy to access the respective $x_{(k-1)t}$. The generation of this table costs $O(n)$.

Then, given $y'$, compute $f$ iteratively on $y'$, till the result appears in the table as some $x_{kt}$. Jump to $x_{(k-1)t}$ and iteratively compute $f$ again, till we get $y'$. The previous value is $f^{-1}(y')$. The complexity is $O(\sqrt{n})$. 
The General Case: $f$ is Not a Permutation

In this case, there are no cycles, but a forest of many large connected subgraphs, with a small single cycle at the center of each.

Therefore, the technique used for permutations cannot be directly applied.
The Idea

Choose $m$ starting points (keys) $S_1, \ldots, S_m$, and apply $f$ iteratively $t$ times (we will choose $m$ and $t$ later):

\[
S_1 = X_{1,0} \stackrel{f}{\rightarrow} X_{1,1} \stackrel{f}{\rightarrow} X_{1,2} \stackrel{f}{\rightarrow} \ldots \stackrel{f}{\rightarrow} X_{1,t} = E_1 \\
S_2 = X_{2,0} \stackrel{f}{\rightarrow} X_{2,1} \stackrel{f}{\rightarrow} X_{2,2} \stackrel{f}{\rightarrow} \ldots \stackrel{f}{\rightarrow} X_{2,t} = E_2 \\
\vdots \\
S_m = X_{m,0} \stackrel{f}{\rightarrow} X_{m,1} \stackrel{f}{\rightarrow} X_{m,2} \stackrel{f}{\rightarrow} \ldots \stackrel{f}{\rightarrow} X_{m,t} = E_m
\]

where for any $i$ and $j$, $X_{i,j} = f(X_{i,j-1})$, and

\[E_i = f^t(S_i).\]

To save space, we will not keep the intermediate values of $X_{i,j}$: We will keep only $(S_i, E_i)$ sorted by $E_i$. 
Given a ciphertext $C_0$ of the plaintext $P_0$, encrypted under an unknown key $K$.

Apply $R$ and get

$$Y_1 = R(C_0) = f(K).$$

Check if $Y_1$ is one of the $E_i$’s: $Y_1 = E_i = X_{i,t}$. Then, $X_{i,t-1}$ is a candidate for $K$.

We have two problems:

1. We did not keep $X_{i,t-1}$.
   We can compute it again as $f^{t-1}(S_i)$.

2. There might be false alarms: some of these candidate keys are wrong.
   We can check this by encrypting a plaintext and verifying that it really encrypts like $K$. 
The Attack (cont.)

If $Y_1$ is not one of the $E_i$’s, or if a false alarm was found:

We compute $Y_2 = f(Y_1)$ and check if it is one of the $E_i$’s.

If it is, $X_{i,t-2}$ is a candidate for $K$.

And so on, for $Y_3$, $Y_4$, etc.
In general, we apply:

\[ Y_1 = R(C_0) \]

for \( j = 1 \) to \( t \) do

if \( Y_j \in \{ E \} \) (say \( Y_j = E \))

compute \( K' = X_{i,t-j} = f^{t-j}(S_i) \)

Check if \( K' \) is a false alarm

If \( K' \) is not a false alarm

Output \( K = K' \)

\( Y_{j+1} = f(Y_j) \)
Analysis

If all the $mt \ X_{i,j}$’s are different, and if $K$ is chosen at random, then the probability of success is

$$P(S) = \frac{mt}{N}$$

where $N = 2^k$.

For a particular probability, we can choose various values for $m$ and $t$ as long as their product remains unchanged.

This allows us to apply a tradeoff between time and memory: $t$ time, $m$ memory.

Can we choose $m = t = 2^{k/2}$? No.
Behavior of Random Functions

Denote the key space by \( \mathcal{K} = \{0, \ldots, 2^k - 1\} \).

\( f \) is a function \( f : \mathcal{K} \to \mathcal{K} \).
Are All Values in $\mathcal{K}$ in the Range of $f$?
Are All Values in $\mathcal{K}$ in the Range of $f$?

No. If $f$ is random, then we can view the outputs as of throwing $N$ balls into $N$ holes.

About

$$(1 - \frac{1}{N})^N \approx \frac{1}{e}$$

of the holes remain empty, i.e., $1/e$ of the values in $\mathcal{K}$ are not in the range of $f$.

Thus, unless these values are starting points (and most of them are not), we cannot find these keys by the table built from $f$.

Also, many values have more than one preimage.
How Long can we Walk Along $f$?

Start from some $K_0$. Define $K_1 = f(K_0)$, $K_2 = f(K_1)$, etc.

How many times can we apply $f$ before we get back to somewhere we were before?
If $f$ is a Permutation

If $f$ would be a permutation, then $f$ would be invertible, and thus each value would have exactly one preimage.

Going along the path from $K_0$, we have probability $1/N$ to collide with $K_0$ in each step.

We cannot collide with any other value in the path.

Thus, the expected period of the cycle of $K_0$ is about $O(N)$ (more accurately, $N/2$).

In this case, we can deterministically choose the starting points, by walking along the cycles, and choose a starting point every $\sqrt{N}$ applications of $f$. In this case the complexity is really $t = m = \sqrt{N}$, and every key can be found by the attack using one table.
The Period of a Random $f$

If $f$ is random, in each step we have $1/N$ probability to collide with $K_0$.

We also have $1/N$ probability to collide with any other preceding $K_i$ in the path.

How long should we walk along the path to find a collision (say with probability half)?

The birthday paradox predicts that $O(\sqrt{N}) = O(2^{k/2})$ random values are required to have a collision with probability half.
Discussion

If \( f \) would define one huge cycle, we could divide the cycle into \( T \) distinct path of length \( S = N/T \).

The best tradeoff would be \( T = S = \sqrt{N} \).

On the other hand, if \( f \) defined \( N \) cycles of length one, \( f \) would be the identify function, but our attack would not work at all.

In our case, after about \( \sqrt{N} \) applications of \( f \) we get a collision, i.e., a cycle, and continuing the path do not add any new value to the path. Thus, we should limit \( t \) to be \( t \ll \sqrt{N} \).

Moreover, there is no point to increase \( mt^2 \) over \( mt^2 = N \), since the gain in the probability of success is very small, due to overlaps.

If we choose \( m = t = N^{1/3} \) we get \( P(S) = \frac{mt}{N} = N^{-1/3} \). (Hellman claims that this choice is optimal).
This is small. How can we increase it?

We can apply the same algorithm $N^{1/3}$ times with $N^{1/3}$ different $R$'s, and then, the probability of success in one of the applications is

$$P(S) = 1 - \left(1 - \frac{1}{N^{1/3}}\right)^{N^{1/3}} \approx 1 - \frac{1}{e} \approx 0.63$$
**Complexity**

The full attack, i.e., applying the algorithm on \( \frac{N}{mt} \) tables, where \( mt^2 = N \), requires:

Time: \( T = t^2 \)

Memory: \( S = mt \).

Choosing \( m = t = N^{1/3} \) we get

\[
S = T = N^{2/3}.
\]

A precomputation of \( mt^2 = N \) steps is required. Then, the key of any ciphertext can be found in \( N^{2/3} \) time and space.
Rate of False Alarms

We still have to show that the rate of false alarms does not affect the complexity, i.e., it is small enough.

**Theorem:** The expected number of false alarms in each table, $E(F)$, is bounded by

$$E(F) \leq \frac{mt(t + 1)}{2N}.$$

**Conclusion:** If $mt^2 = N$ and $t \gg 1$ then

$$E(F) \leq 1/2,$$

i.e., the expected number of false alarms in a table is at most $t/2$.

The complexity of testing in a table is $t$, thus the additional complexity due to false alarms is at most 50%.
Rate of False Alarms (cont.)

Proof of the Theorem:

Let $F_{i,j}$ be the random variable of having a false alarm when processing $Y_j = E_i$.

Then,

$$E(F) = \sum_{i=1}^{m} \sum_{j=1}^{t} P(F_{i,j}).$$

Look at $F_{i,j}$: False alarm in $F_{i,j}$ can be achieved in $t - j + 1$ different (and independent) cases:

1. if $f(K)$ is already in the path from $S_i$ to $E_i$.
2. if $f(K)$ is not in the path from $S_i$ to $E_i$, but $f^2(K) = f(f(K))$ is.
3. if neither $f(K)$ nor $f^2(K)$ are not in the path from $S_i$ to $E_i$, but $f^3(K)$ is.
4. etc.
Rate of False Alarms (cont.)

Each of them has probability $1/N$ to occur, thus

$$P(F_{i,j}) = (t - j + 1)/N.$$ 

In total

$$E(F) \leq \sum_{i=1}^{m} \sum_{j=1}^{t} (t - j + 1)/N = \sum_{i=1}^{m} \sum_{j=1}^{t} j/N = \frac{mt(t + 1)}{2N} \approx 1/2.$$ 

QED