Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout 11
October 8, 1998

Problem Set 2 Solutions

Problem 2-1. Cryptanalysis of a Block Cipher

Chris Peikert
Brandon DuRette
Anthony Young-Garner
Yoav Yerushalmi

This problem was not trivial. Many students had some very good solutions, but then got stuck trying to prove that their solutions worked, even in a "worst case" scenario. Others came up with very good techniques that worked in the average case, but failed to explain the assumptions they made. The following is a general solution that works in the average case, and after that, I will provide a probabilistic algorithm that works for all cases with a certain probability.

This block cipher can easily be broken using a chosen plaintext attack. The key observation is that there are large classes of keys $(\pi, (S_0, S_1, \ldots, S_7), \mu)$ that yield equivalent mappings from plaintext to ciphertext. In particular, the input can be divided into eight sets of eight bits each, and the output can be divided into eight sets of eight bits each, where each set of input bits affects the bits in exactly one of the output sets. As an example, in one case we may have input bits 0, 5, 6, 10, 20, 25, 36, and 63 solely determining the values of output bits 10, 12, 21, 31, 37, 40, 46, and 57. In addition, the order of the S-boxes is irrelevant. Given a particular key, there exists a key with equivalent S-boxes which can undo the input permutation within a set of input bits, and also within a set of output bits. Therefore π and μ serve only to define the sets of input and output bits (but not their order into and out of the S-boxes). Performing a differential attack can allow us to determine the sets of bits in both the input and output. With this information, we can determine equivalent S-boxes by cycling through each of the 256 input values for each S-box. This can be done in parallel since the pairings of input bit sets and output bit sets are independent. Once we have equivalent S-boxes, we can simulate the encryption without the key.

The first-pass algorithm is relatively fast, but makes assumptions that are not necessarily true.

- The designers of the algorithm wanted it to be generally secure, and so aimed for maximal diffusion/confusion. Therefore, in the average case, changing a bit results in 4 bits changing in the output.
- If we start with a random input, and alter one bit, we will discover between 1 and 8 bits changed in the output, but expect it to be four. If we then start from a random output, and then see what changes when we flip a bit we know about, we'll find out approximately four bits in the input that change.
- Therefore, the bits that go together into some S-box and the corresponding output bits from the same S-box can be decided by using less than 64 messages.

The algorithm works thus:

1. pick a random input R, and encrypt to get $E(R)$.
2. change a bit that we know nothing about in R, yielding R', then get $E(R')$
3. $E(R')$ and $E(R)$ differ in at least 1 and at most 8 places. Group those changed bits. Look in at the other output groups. If any share a bit with this group, unite the two into one larger group, and unite the associated inputs into one larger group.
4. repeat until all 8 input-output groups are known about.

We can also do the same starting from ciphertexts, and in fact the two can be combined to get the sets faster.

Repeating the above until we learn all 8 bits for each box deems that on the first pass, we get around 4 unknown bits, then 2, then 1, and the last unknowns can be determined by changing them and seeing the results. Recall that when any particular output bit changes and belongs to a set that we know about, all those outputs are related to the input, and the input is related to all the other inputs that affect those outputs.

Therefore, it takes around 8 messages per S-box to learn which inputs-outputs go together, so around 64 messages to learn the mappings, plus another 256 to find out the S-box values, which yields a result of around 320.

Unfortunately, the above technique fails for 'bad' S-boxes. For example if the S-boxes were the identity function, we could never learn which bits go together. While this is bad, if we can determine that some bits are independent from others within an S-box, we can actually build the S-box table more quickly.

However, there is also one more situation that could be bad, and that we can't 'easily' detect. Say the S-boxes are identity functions, except on two specific inputs. For example, say that S is an identity, but 01000000 becomes 00000010, and 00000010 becomes 01000000.

At this point, the easiest way to solve the problem is to provide a probabilsitic randomized algorithm. Here is the basic idea:

- Every input bit is associated with at most 8 output bits, and at least 1 output bit. An input bit is 'associated' with an output bit if there is some way to set all other 63 input bits such that toying with the input bit we are looking at will affect the output bit. In a well designed random S-box, every input bit is associated with every output bit. In an identity, every input is associated with only one output.
- There are at LEAST 64 associations (otherwise it will not be a bijection), and at MOST $(64 * 8) = 256$ associations (otherwise some bits are used as inputs to more than one S-box).
- If we pick a random input, and change one bit, we will discover at least one association.
- For any association of an input bit, there's a $1/128$ chance of discovering it by setting the input to some random value and then flipping the input bit.

So, given the above facts, we will try to discover enough associations to build up the groups, and then do another 256 lookups to build the table. We use the following algorithm:

1. Pick a random input R, and find its encryption $E(R)$.
2. For all input bits, change R in just one bit, and look at the output. This should reveal at least 64 associations (one per bit changed).
3. Repeat the above k times.

Now, we note that on a single pass of the algorithm ($k = 1$), we have a $1/128$ chance of noticing a particular association. Therefore, we analyze what is the probability of NOT finding any particular association after k rounds.

$$\begin{aligned}
\text{Pr[finding a particular association]} &\geq 1/128 \\
\text{Pr[NOT finding it on a pass]} &\leq 1 - 1/128 \\
\text{Pr[NOT finding it after k passes]} &\leq [1 - 1/128]^k \\
\text{Pr[not finding some association after k rounds]} &\leq (8 * 64) * [1 - 1/128]^k
\end{aligned}$$

Each round takes 65 messages (although we can reuse them later for when we build the tables). Now we just need to decide how many rounds to do this for. For $k = 512$, we have the probability of missing an association ≤ 0.018. We can usually build the groups even if we are missing some associations. After trying around 2^{16}, we have less than around a percent chance of being wrong. This is MUCH fewer than 2^{64}.

Problem 2-2. Cascade Ciphers

Roger Dingledine
Nick Mathewson
Brian Sniffen

(a)

We will now show that when $K1$ and $K2$ are chosen independently, $E_3((K1, K2), M) = E_2(K2, E_1(K1, M))$ is at least as secure against chosen plaintext and ciphertext attacks as E_1 and E_2. In particular, we will show that having a successful attack on E_3 allows us to trivially mount an attack against both E_1 and E_2.

Suppose that we have an attack on E_3 which, given enough chosen plaintexts and ciphertexts, allows us to determine $K1$ and $K2$.

If we want to mount an attack against E_1 or E_2, we proceed by wrapping our inputs and outputs to the unknown cipher with the other cipher, so that we have E_3. We then break E_3, and deduce the key to the unknown cipher.

In other words, if we wish to get a K_1 being used with E_1, given an insider that will encrypt/decrypt for us, we let $E(M) = E_2(K0, E_1(K1, M))$ and let $D(C) = D_1(K1, D_2(K0, C))$, for some chosen key $K0$ and the messages required by the attack on E_3. But then $E(M) = E_3((K1, K0), M)$. We simply pass to our "friend" the M's our attack requires, and have him encrypt for us. Then we encrypt using E_2 and our K_0 and feed it to our E_3 breaker. Since we can break E_3, we can deduce the identities of $K1$ (which we wanted) and $K0$.

Additionally, if we wish to break $E_2(K2, M)$, we let $E(M) = E_2(K2, E_1(K0, M))$ and let $D(C) = D_1(K0, D_2(K2, C))$, for some chosen key $K0$. So, to attack E_2 and get $K2$ using an "insider" who will encrypt/decrypt for us, we figure out what messages the attack on $E3$ needs, and encrypt them using a dummy $K0$. Then we give the encryptions to our friend and have him encrypt them. We feed his result to our $E3$ breaker, and then $E(M) = E_3((K0, K2), M)$, and since we can break E_3, we can deduce the identities of $K2$ (which we wanted) and $K0$.

So by enciphering and deciphering our chosen plaintexts and ciphertexts above, we may reduce an attack on E_1 or E_2 to an attack on E_3. Therefore, E3 must be at least as strong as the stronger or E_1 and E_2.

(b)

Here is a trivial example of a system where keys are not chosen independently, and so E_3 may be more vulnerable than E_1 or E_2.

Suppose E_2 leaks key bits by appending the key to the message, and $K1 = K2$. Clearly, it is possible to deduce $K2$, the key to E_2 by inspecting the ciphertext. But since $K1 = K2$, is is therefore possible to deduce the key to E_1, no matter how secure E_1 may be.

(c)

Now we will show that when two stream ciphers E_1 and E_2 are used in tandem, so that the combined pad $P_3 = P_1 \oplus P_2$, and $K1$ is independant of $K2$, E_3 is at least as secure as E_1 and E_2. We will do this by showing that any attack which allows us to deduce the keys to E_3 from known plaintext/ciphertext pairs can be extended to an attack on E_1 and E_2.

Such an attack on E_3 is equivalent to having a function which takes in some amount of P_3 and returns $K1$ and $K2$. Suppose we have such an attack, and want to break E_1 alone. We first get enough plaintext/ciphertext pairs from E_1 to deduce a sufficient amount of P_1. We then choose some arbitrary K_2, and let $P_3 = P_1 \oplus P_2$, where P_2 is the pad generated by $E_2(K_2)$. But this is enough to mount our attack on E_3, which will deduce $K1$ (which we wanted), and $K2$.

The attack on E_2 is homologous.

In order to show that the above relation does not hold when $K1$ and $K2$ are not independant, we provide this conterexample: Suppose that E_1 is equal to E_2, delayed by a byte, and that $K1 = K2$. In other words, if $P_2 = \{A, B, C \ldots\}$, then $P_1 = \{0, A, B, C \ldots\}$.

Tt is trivial to deduce P_1 and P_2 from P_3: take the first byte of P_1, $P_1[0] = 0$, then iteratively let $P_2[n] = P_1[n] \oplus P_3[n]$, and let $P_1[n] = P_2[n-1]$.

This would hold even if E_2 were a one-time pad, and so we see that if $K1$ and $K2$ are not independant, we cannot deduce the security of E_3.

Problem 2-3. Finite Fields

William Chuang
Mark Silis
Serena Chan

(a): Addition and multiplication tables for $GF(2^3)$, using $x^3 + x + 1$

Since we're working with coefficients modulo 2, addition (and subtraction) is just the same as binary arithmetic. So our addition table for $GF(2^3)$ is below, with the boxed sums indicating where an additive inverse has occurred. We note that in binary arithmetic, a number is its own additive inverse:

These table elements are read with the higher orders to the left, so (111) is $x^2 + x + 1$.

addition	(000)	(001)	(010)	(100)	(011)	(101)	(110)	(111)
(000)	$\boxed{(000)}$	(001)	(010)	(100)	(011)	(101)	(110)	(111)
(001)	(001)	$\boxed{(000)}$	(011)	(101)	(010)	(100)	(111)	(110)
(010)	(010)	(011)	$\boxed{(000)}$	(110)	(001)	(111)	(100)	(101)
(100)	(100)	(101)	(110)	$\boxed{(000)}$	(111)	(001)	(010)	(011)
(011)	(011)	(010)	(001)	(111)	$\boxed{(000)}$	(110)	(101)	(100)
(101)	(101)	(100)	(111)	(001)	(110)	$\boxed{(000)}$	(011)	(010)
(110)	(110)	(111)	(100)	(010)	(101)	(011)	$\boxed{(000)}$	(001)
(111)	(111)	(110)	(101)	(011)	(100)	(010)	(001)	$\boxed{(000)}$

element	additive inverse
(000)	(000)
(001)	(001)
(010)	(010)
(100)	(100)
(011)	(011)
(101)	(101)
(110)	(110)
(111)	(111)

Since we're using $x^3 + x + 1$ as the irreducible polynomial, we know that the expression modulo itself is 0:

$$\begin{aligned}
x^3 + x + 1 &\equiv 0 && \pmod{x^3 + x + 1} \\
x^3 &\equiv -x - 1 \\
x^3 &\equiv x + 1 && \text{(coefficients modulo 2)} \\
&\vdots \\
x^4 &\equiv x^2 + x \\
x^5 &\equiv x^3 + x^2 \longrightarrow x^2 + x + 1
\end{aligned}$$

Once we have this, we can work out the multiplication long-hand and substitute back in, operating with coefficients modulo 2, to get the table below. The boxed products are where an element and its multiplicative inverse have produced 1.

multiplied	(000)	(001)	(010)	(100)	(101)	(110)	(011)	(111)
(000)	(000)	(000)	(000)	(000)	(000)	(000)	(000)	(000)
(001)	(000)	$\boxed{(001)}$	(010)	(100)	(101)	(110)	(011)	(111)
(010)	(000)	(010)	(100)	(011)	$\boxed{(001)}$	(111)	(110)	(101)
(100)	(000)	(100)	(011)	(110)	(010)	(101)	(111)	$\boxed{(001)}$
(101)	(000)	(101)	$\boxed{(001)}$	(010)	(111)	(011)	(100)	(110)
(110)	(000)	(110)	(111)	(101)	(011)	(010)	$\boxed{(001)}$	(100)
(011)	(000)	(011)	(110)	(111)	(100)	$\boxed{(001)}$	(101)	(010)
(111)	(000)	(111)	(101)	$\boxed{(001)}$	(110)	(100)	(010)	(011)

This table could be half the size, but we'll list them all out to make it easier to scan.

element	multiplicative inverse
(000)	none
(001)	(001)
(010)	(101)
(100)	(111)
(011)	(110)
(101)	(010)
(110)	(011)
(111)	(100)

(b): Multiplicative order of each nonzero element

The easiest way to do this is to look in the above multiplication table and look at the value of the element times 001 – if it is 001 then the order is 1. If it isn't, multiply the result by the original element (increment the order also) and check to see if that is 1, and so on.

It turns out that every element other than 000 and 001 is a generator:

element	multiplicative order
(000)	none
(001)	1
(010)	7
(100)	7
(011)	7
(101)	7
(110)	7
(111)	7

(c): Argue that $(a+b)^p = a^p + b^p$ in $GF(p^k)$

The binomial expansion of $(a+b)^p$ is:

$$(a+b)^p \quad = \quad a^p + \binom{p}{1} a^{p-1} b + \ldots + \binom{p}{p-1} a b^{p-1} + b^p$$

If we look at all the interior terms (i.e., not a^p and b^p) and consider that $\binom{a}{b} = \frac{a!}{(a-b)!b!}$, we notice that every term has $a!$ in its numerator. Since this is a combinatorial, the denominator is guaranteed to not have a larger term than a.

So essentially we see that in our case, $\binom{p}{n}$ with any n other than 0 or p will be divisible by p. That means that it is equivalent to 0 (mod p), and thus all the interior terms drop out leaving:

$$\begin{aligned}(a+b)^p &= a^p + \binom{p}{1}a^{p-1}b + \ldots + \binom{p}{p-1}ab^{p-1} + b^p && \text{(m od p)} \\ &= a^p + b^p && \text{(mod p)}\end{aligned}$$