CS787: Advanced Algorithms

Topic: Cryptanalysis **Presenter(s):** Tycho Andersen, Todd Frederick, and Michael McKinley

17.4.1 Introduction

17.4.1.1 Cryptography terminology

Three of the most important terms in cryptography are the plaintext, the ciphertext, and the key. The **plaintext** is the data the sender wishes to encode, the **ciphertext** is the encoded data, and the **key** is a second piece of data that the sender uses to encrypt the message and the receiver uses to decrypt.

Ciphers can be classified into two groups, based on the nature of the key. The family we'll be talking about is **symmetric-key** - the same key is used to encrypt and decrypt the data. There is a second family, called **asymmetric-key** cryptography. In assymetric key cryptography, the sender uses one key to encrypt and second to decrypt. Discussion and analysis of assymetric-key cryptography is beyond the scope of this project.

Ciphers can also be classified by the manner in which they operate on the data. Two common types are block ciphers and stream ciphers. Block ciphers operate on fixed-sized segments of data. If the data a user wishes to encrypt is longer than the length of the block, the data is broken into multiple blocks. For example, if the block size is 16 bits and the data is 48 bits, the data will be split among 4 blocks. Each of the 4 blocks will be encrypted indepedently; that is, the second block will not be encrypted any differently than the first block. We will be discussing block ciphers. Some examples of block ciphers include DES, Rijndael, and Blowfish.

Stream ciphers are different; operate on the data one byte at a time. From the key, the cipher generates a keystream. To encrypt the text, the i^{th} byte of the keystream is XOR'd with the i^{th} byte of the plaintext. An example of a stream cipher is RC4 - a broken cipher that was used in WEP.

17.4.1.2 Block ciphers

Most block ciphers are comprised of a series of three operations: **substitution**, **permutation**, and **key-mixing**. These ciphers are known as substitution-permuation networks (SPN) and Feistel ciphers. Generally, the cipher is broken into a series of n **rounds**. Each round repeats a series of operations on the data.

An important concept in block ciphers is **sub-keys**. In each round, a cipher will apply the key in some manner to the data. However, for added security, it would be best if the key applied was different in each round. To do so, most block ciphers use sub-keys. At the start of the algorithm, n keys (where n is the number of rounds in the cipher) are generated. This is known as **key scheduling**. For a block size of b, the simplest way to create sub-keys is to have a key-size of $b \times n$.

The key is divided evenly into sub-keys of size b.

17.4.1.2.1 Key mixing

In key mixing, a sub-key is applied to the data. The most common method is to perform a bitwise XOR on the sub-key and the data.

17.4.1.2.2 Substitution

In a substitution operation, the block of data is broken into one or more pieces. Each piece is passed through a substitution box, often shortened to **s-box**. In the s-box, the piece is replaced with another value based on a static table. The s-box introduces non-linearity into the algorithm, making it harder to crack. A poorly-designed s-box can pose a significant weakness in the security of a cipher [4].

We will be talking about a simple algorithm where every s-box is the same. In most real-world algorithms, each s-box is different. That is, in one s-box, $0xF$ might be mapped to $0x9$ and another might map $0xF$ to $0x6$. Furthermore, an s-box might have a different-length output than input.

17.4.1.2.3 Permutation

In a permutation operation, the bits are shuffled in a fixed manner. See Table **??** for an example.

17.4.2 The algorithm

We will be looking at a cipher described in [1]. The cipher has a 16-bit block size, a 64-bit key, and is 4 rounds. The sub-keys are generated by dividing the key evenly.

Input value	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output value	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Table 1: S-box for the cipher

Input bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Table 2: Permutation scheme for the cipher

An overview of the algorithm is presented in Figure 17.4.2.1

17.4.3 Linear Cryptanalysis

17.4.3.1 Overview

If we could find a linear expression relating the input plaintext of a cipher to its output ciphertext, the cipher would be trivially breakable. This is why SPN ciphers use S-Boxes, to introduce non-linearity. However, for poorly designed ciphers, we may be able to find linear expressions be-

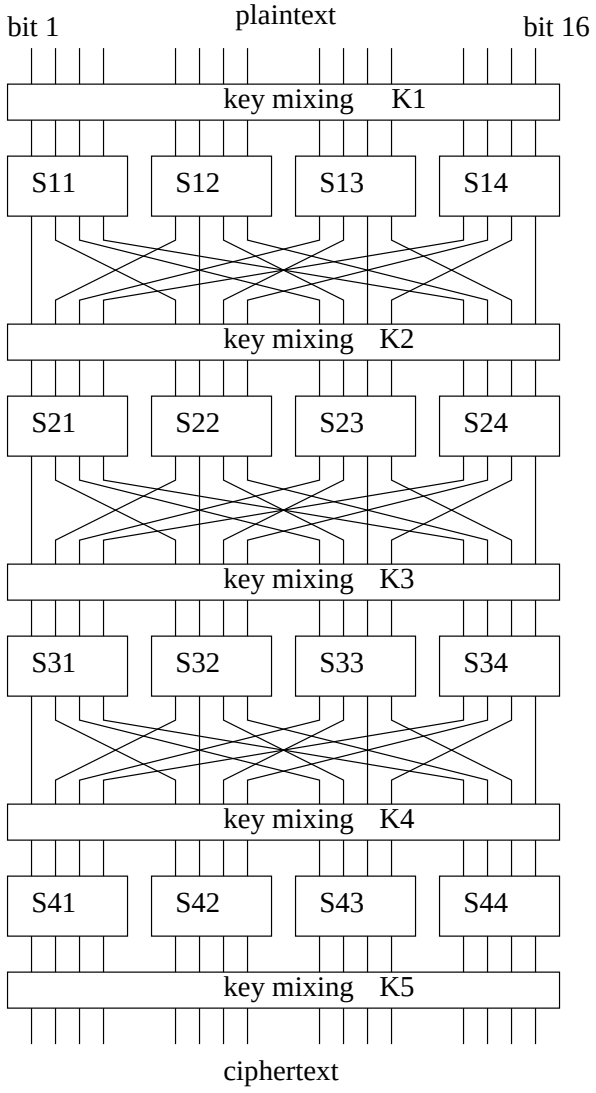


Figure 17.4.2.1: Cipher overview

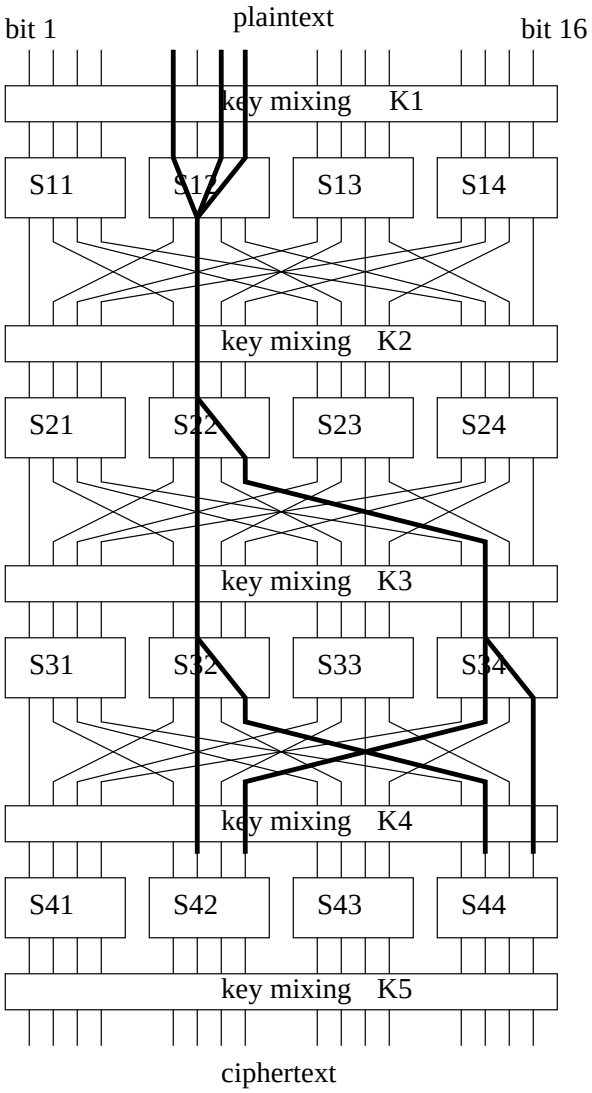
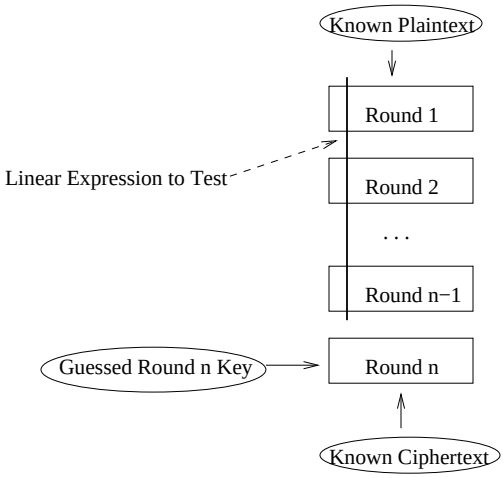


Figure 17.4.2.2: Cipher path

tween certain states of the cipher that are either highly likely to hold or highly likely to not hold. Specifically, these expressions involve bitwise XOR sums of particular bits of the input and output.

Once we have found such an expression between the plaintext and the input to the last round of the cipher, we can obtain a part of the key using the following method: For every possible final round key, we decrypt each known ciphertext for just the last round using this key guess. If we have guessed the correct key, our linear expression will likely hold between the plaintext and this partially-decrypted intermediate state. For each key guess, we count the number of plaintext/ciphertext pairs for which the expression holds. The premise of linear cryptanalysis is that the key for which the linear relationship holds for the greatest number of plaintext/ciphertext pairs is likely the correct key.



The following diagram illustrates the overall strategy of linear cryptanalysis. We now fill in the details of the process with a specific example as provided by [1].

17.4.3.2 S-Box Expressions

The first step in finding a linear expression for the cipher is to find linear expressions for the S-Boxes because these are the only non-linear parts of the cipher. Recall that an S-Box takes input bits X_i and produces output bits Y_i. We will find expressions of the following form, where i and j are sets of indicies:

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_n} \oplus Y_{j_1} \oplus Y_{j_2} \oplus \cdots \oplus Y_{j_m} = 0$$

For a given S-Box, we want to find expressions of this form such that the probability that the expression holds differs significantly from 1/2. We do this by considering every possible expression and counting the input/output pairs of the S-Box for which the expression holds.

When enumerating these expressions for an S-Box, it is useful to describe the relationship by an input mask and an output mask. Each mask determines which bits in the input or output will be part of the expression. For example, suppose we have input mask 5 and output mask A, and we number bits from left to right. Then the expression is:

$$X_2 \oplus X_4 \oplus Y_1 \oplus Y_3 = 0$$

For all 4-bit integers, we apply the S-Box and check if the expression holds between the input and output. We do this for all possible expressions of this form for each S-Box used by the cipher.

For each S-Box, we summarize the results of this process in a table. The axes of the table describe the parameters of the expression, the input and output mask. For a given expression, the value in the table is the number of input/output pairs for which the expression holds.

Since there are 16 possible inputs to a 4-bit S-Box, the probability that an expression holds is

its value in the table divided by 16. If the S-Box were completely non-linear, the probability of each expression holding would be 1/2, and all the values in the table would be 8. However, for poorly designed ciphers this is not always the case. In the next part of the algorithm we consider expressions from several S-Boxes that either hold with high probability or low probability.

17.4.3.3 Cipher Approximation Construction

Since the S-Boxes are the only non-linear components of an SPN cipher, we can combine the approximate linear expressions we found in the previous step into an approximation of the entire cipher. Recall that our overall strategy is to find an approximate linear expression relating some plaintext bits to some input bits of the final cipher round, after which we test possible final round keys in the last step. Consider the diagram representing the example SPN cipher. We must use enough S-Box approximations so that we can construct a path from a selection of plaintext bits to a selection of final-round input bits. The following S-Box approximations will lets us construct such a path in the example cipher:

$$\begin{aligned} S_{12} : X_1 \oplus X_3 \oplus X_4 \oplus Y_2 = 0 \\ S_{22} : X_2 \oplus Y_2 \oplus Y_4 = 0 \\ S_{32} : X_2 \oplus Y_2 \oplus Y_4 = 0 \\ S_{34} : X_2 \oplus Y_2 \oplus Y_4 = 0 \end{aligned}$$

Let $U_{i,j}$ denote the bit j input to the stage i S-Boxes, and let $V_{i,j}$ denote the bit j output from the stage i S-Boxes. The following diagram shows how we can build a path through the cipher using these S-Box approximations that relates plaintext bits P_5, P_7, P_8 to final-round input bits $U_{4,6}, U_{4,8}, U_{4,14}, U_{4,16}$.

Figure 17.4.2.2 describes how we combine the S-Box approximate linear expressions to form an approximate linear expression for the cipher. For example, consider the expression for S-Box S_{12}. Then the expression for S_{12} becomes

$$U_{1,5} \oplus U_{1,7} \oplus U_{1,8} \oplus V_{1,6} = 0$$

Let $K_{i,j}$ denote bit j of the round i key. Then we can substitute expressions for the key mixing phase of round 1 as follows:

$$(P_5 \oplus K_{1,5}) \oplus (P_7 \oplus K_{1,7}) \oplus (P_8 \oplus K_{1,8}) \oplus R_{1,6} = 0$$

We continue this process for rounds 2 and 3, as well as the key-mixing phase of round 4, by using output expressions from one round as inputs to the next round according to the path through the cipher. We can effectively remove all $K_{i,j}$ from our approximate expression because the key is fixed for all the plaintext/ciphertext pairs we will consider in the next step. The overall approximate expression is the following:

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = 0$$

17.4.3.4 Piling-Up Lemma

In order to estimate the number of plaintext/ciphertext pairs we need in the final step to find the key, we must know the probability with which our overall approximate linear expression holds. For this we use a result from [2] called the Piling-Up Lemma.

Lemma 17.4.3.1 *Let X_i for $i = [1, n]$ be random binary variables, where p_i is the probability that $X_i = 0$. Then the probability that $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$ is*

$$1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2)$$

We constructed our cipher approximation from S-Box approximations that held with the following probabilities:

$$S_{12} : 12/16$$
$$S_{22} : 4/16$$
$$S_{32} : 4/16$$
$$S_{34} : 4/16$$

So, by the Piling-Up Lemma, the cipher approximation holds with probability 15/32.

17.4.3.5 Key Testing

The final step of linear cryptanalysis is to guess keys to the final round, and for each key, count the number of plain/ciphertext pairs for which the linear approximation holds. The key guess for which the fraction of pairs that hold differs the most from 1/2 is likely the correct key.

The linear expression in this example involves bits 6,8,14 and 16 of the input to the final round S-Boxes. These bits are inputs to the second and fourth final round S-Boxes specifically, and the outputs of these S-Boxes are only combined with bits 5 through 8 and 13 through 16 of the final round key. So, our key guesses will only involve these bits of the final key.

Let K_a denote the 4-bit block of bits 5 to 8 of the final round key and K_b denote bits 13 to 16. For each possible value of K_a and K_b, we do the following: Take each known ciphertext C_i and decrypt the appropriate bits with K_a and K_b. Apply the appropriate fourth-round S-Boxes in reverse to obtain $Q_{6,4}, Q_{8,4}, Q_{14,4}$, and $Q_{16,4}$ that correspond to C_i. Now take P_5, P_7, and P_8 of the plaintext P_i that corresponds to C_i. Check whether the cipher approximation holds for this C_i/P_i pair, and increment a counter if it does.

In [1], 10000 plain/ciphertext pairs were generated using $K_a = 2$ and $K_b = 4$ (hex). As expected, the count of plaintext/ciphertext pairs for which the approximation held using the correct K_a and K_b as a key guess differed the most from 5000 out of all keys guessed.

17.4.4 Linear Cryptanalysis on DES

Using the techniques above, [3] presents the following two equations, which (on random plaintexts and their ciphertexts) hold with a 15-round best probability of $\frac{1}{2} - 1.19 \times 2^{-31}$. P_H, P_L and C_H, C_L denote the high and low 32 bits of the 64-bit plain and ciphertexts, respectively. K_n denotes the n-th round 48-bit subkey in DES. F_n denotes the n-th round F function from the 2nd to the 15th round of 16-round DES. N denotes the given number of plaintext/ciphertext pairs.

$$\begin{aligned} & P_H[7,18,24] \oplus F_1(P_L,K_1)[7,18,24] \oplus C_H[15] \oplus C_L[7,18,24,29] \oplus F_{16}(C_L,K_{16})[15] \\ = \quad & K_3[22] \oplus K_4[44] \oplus K_5[22] \oplus K_7[22] \oplus K_8[44] \oplus K_9[22] \oplus K_{11}[22] \oplus K_{12}[44] \oplus \\ & K_{13}[22] \oplus K_{15}[22] \end{aligned} \tag{1}$$

$$\begin{aligned} & C_H[7,18,24] \oplus F_{16}(C_L,K_{16})[7,18,24] \oplus P_H[15] \oplus P_L[7,18,24,29] \oplus F_1(P_L,K_1)[15] \\ = \quad & K_{14}[22] \oplus K_{13}[44] \oplus K_{12}[22] \oplus K_{10}[22] \oplus K_9[44] \oplus K_8[22] \oplus K_6[22] \oplus K_5[44] \oplus \\ & K_4[22] \oplus K_2[22] \end{aligned} \tag{2}$$

17.4.4.1 Algorithm for breaking 16-round DES

Data Counting Phase

1. Prepare 2^{13} counters TA_i and initialize them to 0 (i corresponds to the value of the 13 effective text bits of equation 1).
2. For each given plaintext P and corresponding ciphertext C, compute the value of i of step 1, and increment TA_i.

Key Counting Phase

3. Prepare 2^{12} counters KA_j and initialize them to 0 (j corresponds to each value of the 12 effective key bits in equation 1).
4. For each j in step 3, let KA_j be the sum of the TA_i's such that the left side of equation 1, whose values can be uniquely determined by i and j, is equal to 0.
5. Sort KA by the magnitude of $|KA_i - \frac{N}{2}|$, and call this sorted list KAS. For each KAS_i, if $KS_i - \frac{N}{2} \leq 0$, guess the right side of equation 1 is zero, else guess one.

Each of the above 5 steps can be repeated with equation 2, yielding TB, KB, and KBS. Since both of these equations contain 13 effective key bits, the results give us 26 key bits. The sets KAS and KBS are the list of probable key bits, sorted in order of likelihood.

Exhaustive Search Phase

From here, we would like to determine the remaining $56 - 26 = 30$ key bits. We do this by the following algorithm:

6. Create the list Q, which is a list of the candidates in order of decreasing likelihood.
7. For each Q_i, brute force search the remaining 30 key bits until the correct key is found.

17.4.4.2 Algorithm Analysis

Next, we present an analysis of the runtime and space requirements of the above algorithm for breaking 16-round DES.

Space requirements. The total amount of memory required by this algorithm (in bytes) is $(2^{12} + 2^{13}) \times 4$, since the value of each counter required by steps 1 and 3 can fit in 4 bytes. There is also some small constant amount of space required for the computation of DES in the final brute for the final two phases of the algorithm.

Computational complexity. Step 2 of the algorithm requires $O(N)$ time, since it must compute the value of TA from step one for each plaintext/ciphertext pair. Since this is a very large number (something around 2^{45}+ p/c pairs are required), this step dwarfs all the other steps in terms of computational complexity. One caveat is the last step: the lower the number of p/c pairs, the lower the success rate of linear cryptanalysis, and so we will probably have to iterate further through the list Q (each iteration requires 2^{30} executions of DES). An expression for the success rate based on N is discussed in the next section. A final thing to note is that step 2 of the algorithm can be parallelized, which could speed things up quite a bit. Also, in the experiments presented in [3], most of the time was spent generated the 2^{43} p/c pairs (40 days), compared to 10 days to execute step 7 and actually find the key. [3] contains a much deeper experimental analysis of the number of given p/c pairs and the expected computational complexity and success rate.

Comparison with brute force. We can compare this runtime of $O(N) \approx 2^{44}$ (assuming a success rate of around 96%, see next section for more discussion on this) to that of brute force, which has has a runtime of 2^{56}. This means that linear cryptanalysis gives us a $2^{56-44} = 2^{12} = 4096$ times speedup over brute force!

Another interesting question is: how do we create the list Q in step 6? Clearly the first element is $Q_0 = (KAS_0, KBS_0)$, since both keys have the highest probability of being correct. [3] conducts some experiments on a smaller version of DES with only 8 rounds. The proposed result is that the list Q should be constructed from KAS, KBS in order of increasing products of the indexes, i.e. in order of increasing $(i + 1) \times (j + 1)$.

17.4.4.3 Algorithm Success Rate

Discussing the success rate of linear cryptanalysis is a bit of a misnomer. In fact, linear cryptanalysis always succeeds at finding the solution. The proof for this is not hard to see: step 7 of the algorithm iterates over the entire list Q, which contains all 2^{26} possible options for the 26 key bits (they are

ordered by likelihood, but they're all there). Each iteration of step 7 of the algorithm takes one possible setting of the 26 key bits and brute forces the remaining 30, meaning that every possible combination of key bits is tried, so the algorithm above will always succeed. Thus, when we talk about the success rate of linear cryptanalysis vs DES, what we really mean is "the probability that linear cryptanalysis will be faster than a brute force search of the keyspace". That is, the probability that a particular key is in the first k keys in the list Q (k is usually a small integer, say 100).

In [2], Matsui presents a rather complicated formula which approximates the probability of success as a function of the number of given p/c pairs. The experiments performed in [2] show that the formula is a good approximation. Evaluated at several places, the formula yields the table of success rates seen in figure 17.4.4.3. With $p = \frac{1}{2} - 1.19 \times 2^{-21}$ as it is for the linear approximations [3] proposes for DES, the quantity $|p - \frac{1}{2}|^{-2} \approx 2^{42}$. Thus, if one wants to have a probability of success of more than 96.7%, around 6×2^{42} p/c pairs are needed.

N	Success Rate
$2 \times \|(p - \frac{1}{2})\|^{-2}$	48.6%
$4 \times \|(p - \frac{1}{2})\|^{-2}$	78.5%
$6 \times \|(p - \frac{1}{2})\|^{-2}$	96.7%
$8 \times \|(p - \frac{1}{2})\|^{-2}$	99.9%

Figure 17.4.4.3: Success rates vs Number of plaintexts

17.4.4.4 Modern Ciphers

Since the substitution box is the only non-linear part of most ciphers, the substitution boxes in modern ciphers are chosen very carefully to protect against linear cryptanalysis. [5] has some discussion on good substitution box choices for protection against linear and differential cryptanalysis. [4] has a nice discussion of more work on choosing good s-boxes.

References

[1] H. M. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[2] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology (CRYPTO)*, 1993.

[3] Mitsuru Matsui. The first experimental analysis of the data encryption standard. In *Advances in Cryptology (CRYPTO)*, 1994.

[4] Terry Ritter. S-box design, a literature survey. `http://www.ciphersbyritter.com/RES/SBOXDESN.HTM`.

[5] A. M. Youssef and S.E. Tavares. Resistance of balanced s-boxes to linear and differential cryptanalysis. *Information Processing Letters*, (56):249–252.