

On Applying Molecular Computation To The Data Encryption Standard

Leonard M. Adleman*, Paul W. K. Rothmund*, Sam Roweis†, Erik Winfree†

Laboratory for Molecular Science
University of Southern California
and

* Department of Computer Science
University of Southern California

† Computation and Neural Systems Option
California Institute of Technology

Recently, Boneh, Dunworth, and Lipton described the potential use of molecular computation in attacking the United States Data Encryption Standard (DES). Here, we provide a description of such an attack using the *sticker model* of molecular computation. Our analysis suggests that such an attack might be mounted on a table-top machine, using approximately a gram of DNA and might succeed even in the presence of a large number of errors.

1 Introduction

With their work on DES, Boneh, Dunworth, and Lipton [Bo2] provided the first example of a “practical” problem which might be susceptible to molecular computation. DES is one of the most widely used cryptographic systems. It produces a 64-bit ciphertext from a 64-bit plaintext under the control of a 56-bit key. While it has been argued that special purpose electronic hardware [Wi] or massively parallel supercomputers might be used to break DES in a reasonable amount of time, it appears that today’s most powerful sequential machines would be unable to accomplish the task. We continue the work of Boneh, *et al.* by considering the difficulty of breaking DES on the recently proposed *sticker model* of molecular computation [Ro]. While our results are encouraging, it must be stressed that the feasibility of such an attack will ultimately be decided in the laboratory.

In this paper we consider the so called plaintext-ciphertext attack. Here the cryptanalyst obtains a plaintext and its corresponding ciphertext and wishes to determine the key used to perform the encryption. The most naive approach to this problem is to try all 2^{56} keys, encrypting the plaintext under each key until a key that produces the ciphertext is found. Remarkably, a significantly more efficient attack is not known and consequently this brute force approach will be the one considered here.

We begin by describing the algorithm to implement a plaintext-ciphertext attack for breaking DES at a logical level. This allows us to identify the fundamental operations we need to implement on a *stickers machine*, and serves as a roadmap for what follows.

2 The Molecular Algorithm

Start with approximately 2^{56} identical ssDNA *memory strands* [Ro] each 11580 nucleotides long. We think of each memory strand as containing 579 contiguous blocks (referred to as regions in [Ro]) $B_0, B_1, B_2, \dots, B_{578}$ each 20 nucleotides long. As is appropriate in the sticker model there are 579 stickers S_0, S_1, \dots, S_{578} —one complementary to each block. (We refer to memory

strands with annealed stickers as *memory complexes*.) Hence, we consider that each strand represents a 579-bit memory, and we sometimes use B_i to refer to the bit which B_i represents. Block B_0 is never set and is used later in the implementation of the algorithm (Subsection 3.1). Blocks B_1 through B_{56} of the memory strands are used to store a key, the next 64 blocks, B_{57}, \dots, B_{120} , will eventually encode the corresponding ciphertext, and the remainder of blocks are used for intermediate results during the computation. The *stickers machine* which processes the memory strands to compute the ciphertexts does so under the control of a microprocessor. Because the plaintext is the same in all cases, the microprocessor may store it; we do not need to represent the plaintext on the memory strands. Now, given a plaintext-ciphertext pair, the algorithm is performed in three steps:

1. *Input step: Initialize* the memory strands to form memory complexes representing all 2^{56} keys.
2. *Encryption step: On each memory complex compute the ciphertext* corresponding to the encryption of the plaintext under that complex's key.
3. *Output step: Select* the memory complex whose ciphertext matches the given ciphertext, and *read* the corresponding key.

The bulk of the work is performed during the second step, where DES data encryption occurs, so we outline it below. Our interest is in demonstrating how DES can be implemented on a molecular computer, and for these purposes the exact details of DES are unnecessary. (For details, see [Na].) We will instead focus on the essential operations required in DES, and how these operations are combined to effect the full algorithm.

DES is a 16-round cipher. In each round a new 32 bit intermediate result is produced. These are designated R_1, \dots, R_{16} . We store R_{15} and R_{16} in locations B_{57} through B_{120} (adjacent to the key), while R_1, \dots, R_{14} are stored in locations B_{121} through B_{568} . Essentially, R_{15} and R_{16} , taken together, form the desired ciphertext. (We encode the ciphertext adjacent to the key for implementation reasons explained in Subsection 3.4.) The left 32-bits and right 32-bits of the plaintext are referred to as R_{-1} and R_0 , and are known to the controlling microprocessor.

Bits B_{569} through B_{578} are used as a *workspace* and are written and erased during the course of the computation. Hence, unlike the other bits which are used in a “write once” fashion, these bits may be cleared; for implementation reasons, we always clear the entire workspace at once.

Essentially, R_i is obtained from R_{i-1} and R_{i-2} by the following computation:

$$R_i = R_{i-2} \oplus S(E(R_{i-1}) \oplus K_i)$$

where \oplus denotes exclusive or (x-or), K_i denotes a round dependent selection of 48 bits from the key, E denotes the *expand* function which takes the 32 bits of R_{i-1} and repeats or permutes them to yield 48 bits, and S denotes the *S-function* which takes a 48-bit input and maps it to a 32-bit output. The function E , the function S and the selection K_i are *hard-coded*, like the plaintext, into the microprocessor.

In fact, the S-function can be separated into eight independent 6-bit to 4-bit functions known as *S-boxes*. Hence, each R_i may be computed in eight independent operations each of which produces a 4 bit *chunk* of the result. A given chunk is a function of 16 *input bits*: 6 bits of R_{i-1} , 6 bits of K_i and 4 bits of R_{i-2} . We describe the computation of a chunk below:

1. 6 bits of R_{i-1} and 6 bits of K_i are x-ored to produce a 6-bit result which is then stored in the workspace locations B_{569}, \dots, B_{574} .
2. One of the S-box functions is applied to bits B_{569}, \dots, B_{574} and the 4-bit result is stored in the workspace locations B_{575}, \dots, B_{578} .
3. Bits B_{575}, \dots, B_{578} are x-ored with 4 bits of R_{i-2} to produce the desired chunk of R_i which is then stored in the appropriate four blocks of the intermediate result bits B_{57}, \dots, B_{568} .
4. If the chunk being computed is not the last chunk of R_{16} , the entire workspace, bits B_{569}, \dots, B_{578} , is *cleared* in preparation for future use.

The positions on each memory complex of the 16 input bits required to compute a given chunk depend only on the chunk number (1, . . . , 8) and the round

number $(1, \dots, 16)$, though the 0/1 value of those bits will vary from memory complex to memory complex. The controlling microprocessor knows which positions contain these bits (they are hard-coded) and knows the x-or or S-box which it needs to apply.

We see, then, that encrypting a plaintext with DES comes down to a process of either (1) selecting 2 bits, producing their x-or, and writing the result in a new bit, or (2) selecting 6 bits, applying an S-box, and writing the resulting 4 bits.

3 Implementation

We now turn to implementing the algorithm on a stickers machine. Such a machine, as described in [Ro], may be thought of as a “parallel robotic workstation”. It consists of a *rack* of tubes (*data tubes*, *sticker tubes*, and *operator tubes*), some *robotics* (arms, pumps, heater/coolers, connectors, etc.) and a microprocessor that controls the robotics. Roweis *et al.* assume that the components of the robotics and a set of three data or operator tubes may be arranged to perform any of the following four operations: *separate*, *combine*, *set* and *clear*.

We assume that the robotics are capable of an extended set of operations:

1. *Parallel separate*. The robotics can *separate* the DNA from each of 32 data tubes into two more data tubes by using 32 separation operator tubes at once.
2. *Parallel combine*. The robotics can *combine* the DNA from 64 different data tubes into one data tube at once. We assume that the *blank operator tube* used for a *combine* in [Ro] is really just a connector which is part of the robotics.
3. *Parallel set*. The robotics can, using one sticker tube with stickers S_k , *set* the bit B_k on the complexes in 64 different data tubes at once. We assume the *sticker operator tube* used for *set* in [Ro] is just a filter that can be built into the robotics.

4. *Clear*. The robotics can *clear* the workspace bits on all complexes in one data tube. We assume that the stickers on the workspace are removed simultaneously. Hence the workspace blocks may be implemented using so called *weak regions* [Ro]. Again, we assume the *sticker operator tube* used for *clear* in [Ro] is just a filter that can be built into the robotics.

Hence, we perform the above four operations using just *data tubes* that may hold DNA memory complexes, *sticker tubes* that are (for the purpose of the computation) an inexhaustible source of a particular sticker S_k , and *separation operator tubes* that hold probes for a particular block B_k .

In the following subsections we describe, where applicable, the implementation of the molecular algorithm using these operations and, for the purposes of estimating the *time* and *space* required by a stickers machine we keep track of the following three *resource quantities*:

1. *Total steps*. We define the number of *steps* as the number of *parallel separations*, *parallel combines*, *parallel sets* or *clears* that any given complex experiences *after* it has been initialized. Hence, we count the operation of the robotics on a large number of tubes in parallel as a single step and ignore the (perhaps serial) process of moving data and operator tubes.
2. *Total rack tubes*. We define the number of *rack tubes* to be the total number of data tubes, sticker tubes, and separation operator tubes used during the computation. All of the tubes are reusable, so we only need copies of a tube if a particular kind of tube must be used more than once in a parallel operation. We note that we never need duplicates of sticker tubes—our robotics are incapable of using more than one sticker tube at once. We will however need duplicate separation operator tubes and many data tubes since we often wish to separate complexes in several different data tubes on the same bit B_i at once.
3. *Maximum number of active tubes per operation*. We define the number of *active tubes*, for any time during the computation, as the number of tubes which the robotics have removed from the rack and are currently processing. Note that the maximum number of active tubes defines the width of the parallelism used by our algorithm and hence it must match the parallelism built into our robotics.

We note that these quantities are applicable for only for the *computation of the ciphertexts* (step 2 of the molecular algorithm) and the *selection* of the given ciphertext (the first part of step 3 of the molecular algorithm). These parts of the molecular algorithm require only the operations given above and hence are performed using the stickers machine. The *initialization* of the memory complexes and final *reading* of the key, however, use some operations that are not included above (*i.e.* dividing the contents of a tube into two tubes, PCR amplification, ligation, etc.) These special operations are used at most once during the implementation of the molecular algorithm and we assume that they could be performed by the human operator of the stickers machine in a reasonably short amount of time (a few hours) and in a small space (a few tubes). Hence they are ignored in the final discussion of the resources required by a stickers machine (Subsection 3.5).

3.1 Initialization of the memory strands

First we must create the initial tube encoding keys. Our desire is to have each of the 2^{56} memory strands store a different key. This might be accomplished, for example, in the following way:

1. Divide the memory strands into two tubes, A and B .
2. Add an excess of S_1 through S_{56} to tube A and allow them to saturate the first 56 blocks on each strand.
3. Use the complement of B_0 as a probe to *separate* the memory complexes in tube A from the excess stickers.
4. Add tube B to tube A .
5. Heat and cool tube A to randomly reanneal the stickers.

The memory complexes produced by this process appear to be reasonably modeled with a Poisson distribution; it is expected that approximately 63% of keys will be represented and that, on average, there is one of each key. Hence, if no errors are committed during the computation, we have a reasonable chance of recovering the key for a ciphertext of interest. Of course, the chance

of succeeding can be increased by starting with more memory strands. To insure that approximately 95% of the keys are represented and that on average three copies of a key are present, we could simply use three times as much DNA. These issues are discussed in more detail in the Section 4.

3.2 Implementing the fundamental operations

As discussed in Section 2, the DES encryption algorithm is a composition of just two simple functions:

- x-ors which are 2-bit input to 1-bit output maps,
- S-boxes which are 6-bit input to 4-bit output maps.

The computation of the 2-bit to 1-bit x-or function is illustrated in Figure 1. We now describe the computation of the x-or function $B_k = B_i \oplus B_j$ in greater detail, give the overhead required, and generalize this to an n -bit to m -bit function:

- Parallel separate* the sample two times to yield four data tubes, one for each possible value $B_i B_j$. This is accomplished by first using one separation operator tube specific for B_i and then, in parallel, using *two* separation operator tubes specific for B_j . (In Figure 1 the bits being considered are shaded gray.) Roweis *et al.* model a single separation as an operation involving three active tubes at once: a source data tube, a separation operator tube, and one of two destination data tubes picked up and filled by the robotics in sequence. Hence, during each single separation, three tubes are active at once and three data tubes are used. During the second *parallel separation* above, then, six data tubes are used and six tubes are active.

For an n -bit to m -bit function this generalizes to:

Parallel separate the sample n times to yield 2^n data tubes, one for each value of the n -bit input. This requires 2^{i-1} separation operator tubes for the i th *parallel separation* (a total of $2^n - 1$). Hence for the n th parallel separation $3 \times 2^{n-1}$ data tubes are required and $3 \times 2^{n-1}$ tubes are active.

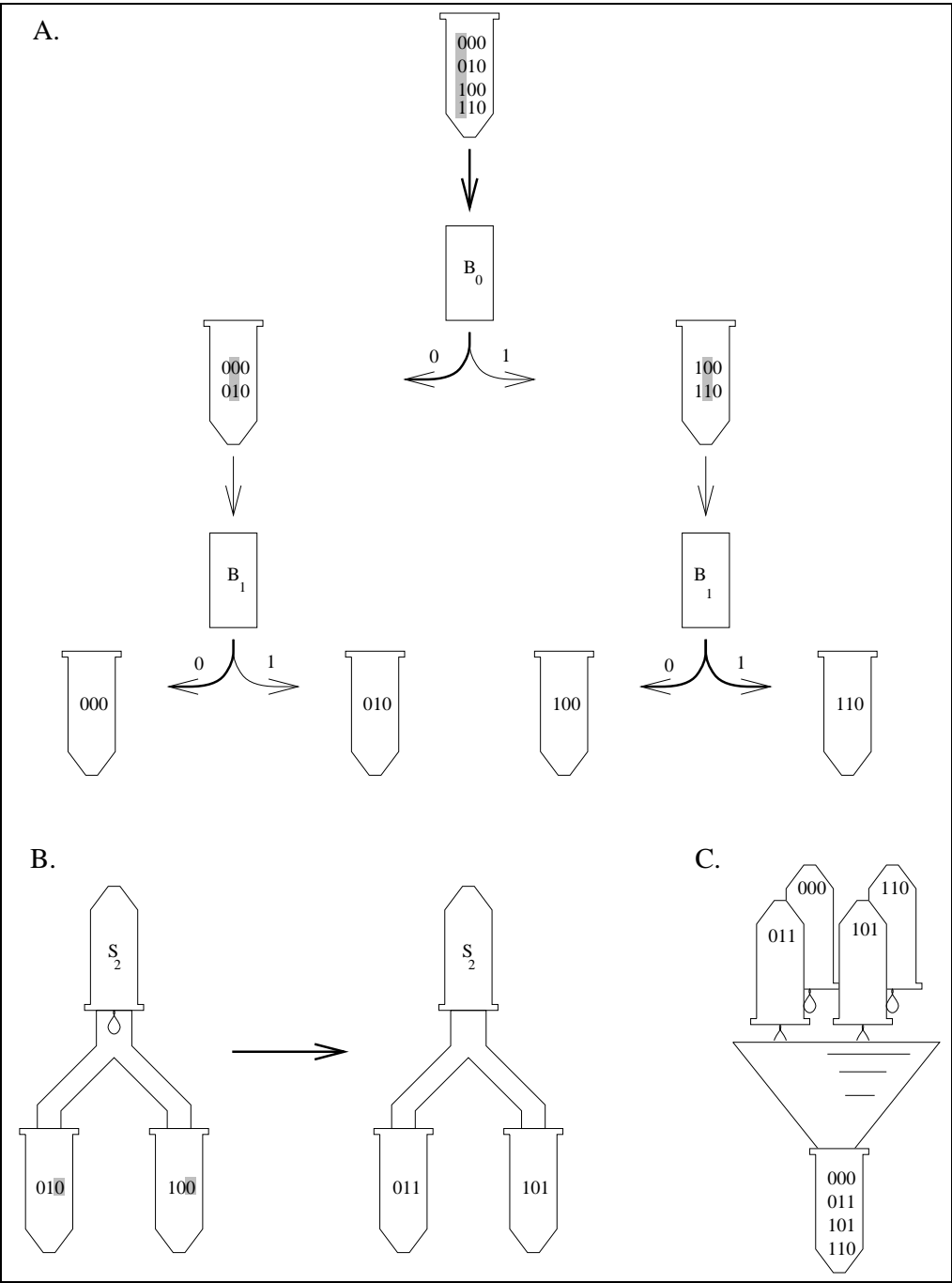


Figure 1: Computation of the 2-bit to 1-bit x-or function: $B_2 = B_0 \oplus B_1$.

- B. *Parallel set* B_k to 1 with an S_k sticker for all tubes for which this is applicable. For an x-or this is only applicable when $B_i B_j = 01$ or 10 but for a general 2-1 function this may require the addition of a sticker to any subset of the four data tubes in parallel. This requires one sticker tube and four data tubes, a total of five active tubes.

For an n -bit to m -bit function this generalizes to:

Parallel set a (possibly different) subset of the 2^n data tubes m times using a total of m sticker tubes. This requires $2^n + 1$ active tubes. Note that the subset of data tubes to which the *parallel set* is applied is determined solely by the algorithm stored in the microprocessor.

- C. *Parallel combine* the contents of all four data tubes into one data tube. This requires five data tubes and hence five active tubes.

For an n -bit to m -bit function this generalizes to:

Parallel combine the contents of all 2^n data tubes into one data tube. This requires $2^n + 1$ data tubes and $2^n + 1$ active tubes.

At the end of our x-or operation, all of our DNA has been returned to a single tube.

In general, an n -bit to m -bit function requires $n+m+1$ steps, 2^n-1 separation operator tubes (specific to various bits), m sticker tubes, a maximum of $3 \times 2^{n-1}$ data tubes, and a maximum of $3 \times 2^{n-1}$ active tubes.

At this point we can see that one of the resource quantities of interest, the maximum number of *active tubes* has already been specified. An 6-bit to 4-bit S-box is the biggest n -bit to m -bit function we implement in DES so we never use more than 96 active tubes. (The only other operation used in the algorithm, *clear*, uses but one active tube.)

The number of *rack tubes* is the sum of the data, sticker and separation tubes used but we may not simply sum up these numbers for an S-box and regard the result as its contribution to the computation. Data tubes are interchangeable so we know that the maximum number used in an S-box—96—will be the data tube contribution to the total *rack tube* count. However, sticker tubes and separator tubes have an identity, they are associated with a particular

block of the memory complexes. For the calculation of *rack tubes* (and total steps as well) we need to consider more details of the molecular algorithm.

Thus, in the next subsection we consider how the x-ors and S-boxes used to compute DES are composed, count the total number of steps, and discuss ways to plan our use of separation operator tubes and sticker tubes to minimize the total number of rack tubes.

3.3 Computing the ciphertexts

We recall that the basic unit of computation in our molecular algorithm is a 4 bit *chunk* that is computed using a composition of x-ors and S-boxes which takes 16 bits of input. For any given chunk the positions of these 16 input bits are the same for *every* memory complex. Thus the positions of the input bits are *hard-coded* into the microprocessor and we ignore their exact values in what follows.

From our consideration of n -bit to m -bit functions, we know that 4 steps are required to compute an x-or and 11 steps are required to compute an S-box. From this it follows that to compute a chunk requires $6 \times 4 + 11 + 4 \times 4 = 51$ steps. After a chunk is computed, if the workspace is to be used again, it must be *cleared*. To complete the computation requires computing $16 * 8 = 128$ chunks, and *clearing* the workspace 127 times; hence the total number of steps required is 6655.

During the computation of an x-or, one separation operator tube is required to *separate* on the first bit, and two separation operator tubes are required to *separate* (in parallel) on the second bit. To economize on separation operator tubes when performing x-ors we choose to *separate* first on a bit for which there are many instances of that “kind” of bit on the memory complex, and second on a bit for which there are only a few instances of that “kind” of bit. Hence, for each x-or involving a bit of R_{i-2} (of which there are 448 possibilities from R_1, \dots, R_{14}) and a bit from the workspace B_{575}, \dots, B_{578} (of which there are only 4), we *separate* on the R_{i-2} bit first and the workspace bit second. Likewise, for each x-or involving a bit of R_{i-1} (of which there are 480 possibilities from R_1, \dots, R_{15}) and K_i (of which there are 56 possibilities), we *separate* on the R_{i-1} bit first and the key bit second. It follows that for

each of the bits from R_1, \dots, R_{15} one separation operator tube specific to that bit is needed and, for each of the bits of K and each of the bits B_{575}, \dots, B_{578} two separation operator tubes specific to that bit are needed. Hence these bits require a total of $480 + 2 \times 56 + 2 \times 4 = 600$ separation operator tubes.

The implementation of the S-boxes demonstrates another way that we may economize on separation operator tubes: frequently used subcomputations do not require that we use up a new bit (and consequently another separation operator tube) every time they are run. Instead their input and output may be stored on a rewritable region of the memory complex—hence our placement of the input to the S-boxes in the first six workspace positions and the output in the last four workspace positions. Because of this, even though there are eight different S-boxes (one for each chunk) they all may use the same separation operator tubes and sticker tubes (though, under control of the microprocessor, the stickers are applied differently for each S-box). The S-boxes, then, only require an additional 63 separation operator tubes to *separate* the DNA into all possible 6-bit strings B_{569}, \dots, B_{574} .

Thus, to complete the ciphertext generation stage of the algorithm, we require 663 separation operator tubes. In all, $512 + 10 = 522$ sticker tubes are required to hold the stickers used to write the intermediate results and workspace bits. (Recall that our robotics only use one sticker tube at a time.) The greatest number of data tubes used during the computation is 96, during the final parallel *separation* step of an S-box computation. In total $663 + 512 + 96 = 1271$ rack tubes are required to compute the ciphertexts.

3.4 Selecting the given ciphertext and reading the correct key

Once the ciphertexts have been computed, the desired key can be *selected* by searching for the complex which has the given ciphertext encoded next to its key. This requires 64 *separation* steps. Upon isolating the desired complex, it is necessary to *read* its key. Reading could be attempted using single molecule detection and a binary tree decoding as described in [Ro]. However, it is not clear that such an approach can be satisfactorily carried out in lab. Below we describe two additional approaches, each of which entails some modification of the methods described in the previous sections:

- In the first approach, the memory strands are 5' biotinylated. Once the ciphertexts have been computed, as described in the previous sections, the memory complexes are transformed *en masse* into single-stranded form. One way to accomplish this is as follows:
 1. For each sticker S_0, \dots, S_{120} create a new *0-sticker*, S'_i , which shares the 3' and 5' 8-mers with S_i but which differs from it in the middle 4-mer.
 2. Add an excess of the S'_i to the final solution under conditions which favor annealing despite the mismatches.
 3. Add ligase. Each memory complex now has a heteroduplex region in its key-ciphertext section where the non-memory strand consists of a sequence of (regular) stickers and 0-stickers. Note that our original decision to place the ciphertext next to the key on the memory strands is no longer a mystery. Our placement minimizes the number of ligations which must be successful for a strand which encodes both the ciphertext and key to form.
 4. *Separate* out the biotinylated memory strands using a streptavidin-coated solid support and retain the new strands as the *library* of solutions.

This process essentially converts Sticker memory complexes into *Lipton style* memory strands [Li] for which each block has one of two unique sequences, one for 0 and another for 1. A ciphertext of interest may be *selected* (by applying the usual 64 *separation* steps), PCR amplified (using S'_0 and the complement of the last ciphertext bit as primers), and *read* (with standard DNA sequencing).

Notice that, once created, the library of solutions can be replicated by PCR and hence multiple copies can be made. Each such copy is essentially a codebook consisting of (key, ciphertext) pairs. This codebook has approximately $2^{56} \times (56 \text{ key bits} + 64 \text{ ciphertext bits}) = 2^{63}$ bits of information (the equivalent of approximately one billion 1 gigabyte CDs) but occupies a dry volume of approximately 1/7 of a teaspoon. As Boneh and Lipton have noted [Bo1] such a codebook could be widely distributed and used to speed up subsequent attacks on DES.

- In the second approach to *reading*, instead of converting to Lipton style memory strands after the computation of ciphertexts, a Lipton-Sticker hybrid model for encoding the initial memory complexes is used.
 1. Make single-stranded DNAs representing all 2^{56} keys using the encoding of Lipton. Additionally, guarantee that each key strand begins and ends with the same short PCR primer.
 2. Ligate, to each of the Lipton key strands, identical 522 bit Sticker memory strands.
 3. Proceed to compute DES as usual.
 4. Perform 64 *separations* on the result to obtain the complex carrying the desired ciphertext.
 5. Perform PCR using the primers which bound the Lipton style key sequence. The key sequence will be amplified exponentially and may be sequenced.

We note that all of the techniques for *selecting* a desired ciphertext discussed here and in [Ro] would require 64 separation operations so we approximate that this process requires an additional 64 steps. Further, we note that to effect all 64 separations an additional 32 separation operator tubes specific to the bits of R_{16} (in addition to the 32 separation operator tubes already counted for R_{15}) are required.

3.5 Discussion

In summary, to find the key for a DES-encoded plaintext-ciphertext pair, we first create memory complexes representing all keys, then we compute the ciphertext corresponding to each key (6655 steps), and finally we select and read the key of interest (64 steps). This requires a total of 6719 steps, each of which is one of the operations described above.

The actual running time for the algorithm depends on how fast the operations can be performed. If we assume, as we might if a graduate student had to perform each operation, that each operation requires 1 day, then the computation will require 18 years. If each operation requires 1 hour (Boneh *et*

al. assume 2.4 hours, [Bo2]) then the computation will require approximately 9 months. If each operation can be completed in 1 minute, perhaps using a robotic stickers machine, then the computation will take 5 days. Finally, if the effective duration of a step can be reduced to 1 second, perhaps by running the algorithm in a continuous flow *parallel refinery* [Ro], then the effort will require 2 hours.

The size of the rack is dictated by the amount of DNA used. When the 2^{56} memory complexes have half of their sticker positions occupied, as we expect will be the case at the end of the computation, they weigh approximately .7 g and, in solution at 5 g/liter, would occupy approximately 140 ml. Hence, the volume of the 1303 *rack tubes* (1271 for ciphertext computation, an additional 32 for ciphertext selection) need be no more than 140 ml each. It follows that the rack tubes occupy, at most, 182 L and can, for example, be arrayed in a rack 1 m (approximately 39 inches) long and wide and 18 cm (approximately 7 inches) deep. Since the robotics must be able to operate on 96 *active tubes* in parallel we approximate the volume required by the robotics, give or take some arms and pumps, as 13 L—1/14th the volume of the rack. The microprocessor is likely to be quite small. Thus, it is reasonable to assume the entire machine would fit on a desktop.

It is worth pointing out that, for much of the computation, the robotics are not processing at full capacity (96 tubes) and many of the tubes are sitting idle. Hence it may be possible to increase the parallelism significantly by pipelining the computation.

4 Analysis of Errors

We say that an error has occurred whenever a memory complex is transformed in an unintended way or ends up in an unintended place. Hence, there are many kinds of errors that can occur during the operation of our DNA computer: strands can break, stickers can fall off one memory complex and reanneal to another, complexes can be “lost” on the walls of a tube, complexes can end up in the wrong tube following a *separation*, etc.

For each operation, we define its error rate to be the fraction of molecules that commit an error during that operation. Some operations are more prone

to errors than others. To simplify our analysis, we define E to be the error rate of the worst operation and assume that all of the operations have error rate E . Note that $1 - E$ corresponds to what chemists call yield. Hence an error rate of 10^{-4} corresponds to a step yield of 99.99%.

Given an input ciphertext-plaintext pair, it is possible that several different keys map the ciphertext into the plaintext (though for DES, it seems unlikely that the number of such keys would be large). All such keys will be called *winning keys*. Under ideal conditions, after the codebook is created and the separation on the input ciphertext performed, we are left with a *final tube* with the following properties:

1. For each winning key there is at least one complex encoding it.
2. All complexes that are there encode winning keys.

In reality this can fail for either of two reasons. First, complexes encoding winning keys may be missing, either because they were not created during initialization or because they encountered an error during the computation. Second, there may be *distractors*: complexes which do not encode winning keys, but due to errors end up in the final tube anyway. In subsection 4.1 we analyze the probability that a winning key has a complex encoding it in the final tube. In subsection 4.2 we calculate the expected number of distractors in the final tube.

We make the following assumptions:

1. After the initialization step, each complex encodes a 56-bit key chosen at random (*i.e.* chosen from the space of all 56-bit keys with equal probability).
2. DES with the input plaintext maps each of the keys to a random ciphertext.
3. A complex which encounters an error during the computation produces a random ciphertext (*i.e.* unrelated to the ciphertext normally associated with that complex's key).

4.1 Probability that a winning key has a complex encoding it in the final tube

In the computation above, we began with 2^{56} memory strands. It will be convenient to carry out the analysis in greater generality. We now assume that we begin with $2^{56}X$ memory strands, where X is a positive rational. Informally, X is the factor by which we multiply our original amount of DNA.

Let K_w be a winning key. Following initialization, the number of memory complexes which encode K_w is given by a binomially distributed random variable $(n, p) = (2^{56}X, \frac{1}{2^{56}})$.

The probability that a memory complex makes it correctly through all 6655 steps of the computation and 64 steps of the selection process (in total 6719 steps) is given by:

$$S = (1 - E)^{6719}$$

Hence, after the computation, the number of memory complexes in the final tube which encode K_w is given by a binomially distributed random variable $(n, p) = (2^{56}X, S\frac{1}{2^{56}})$. Because $2^{56}X$ is very large and $S\frac{1}{2^{56}}$ is very small, this distribution may be approximated by a Poisson distribution with Poisson parameter $\lambda = np = SX$. From this it follows that the expected number of complexes encoding K_w in the final tube is SX and that the probability that a complex encoding K_w is in the final tube is:

$$(1 - e^{-SX})$$

In particular when $X = 1/S$, the expected number of complexes encoding K_w in the final tube is one and the probability that a complex encoding K_w is in the final tube is 63%. We will refer to 63% as a *reasonable chance*.

4.2 Number of distractors in the final tube

For a memory complex M , let $H(M)$ denote the Hamming distance of the ciphertext encoded on M from the input ciphertext. For M to enter the final tube, $H(M)$ errors must occur during the final 64 separation steps. By our assumptions, after the computation of the codebook, each memory complex (whether it has encountered an error or not) encodes a ciphertext which is a random 64 bit string. It follows that the Hamming distances associated with memory complexes will be a binomially distributed random variable $(n, p) = (64, 0.5)$. Hence, the probability that $H(M) = L$ is:

$$\binom{64}{L} \left(\frac{1}{2}\right)^{64-L} \left(\frac{1}{2}\right)^L = \binom{64}{L} \frac{1}{2^{64}}$$

The probability of complex M with Hamming distance $H(M) = L$ making it through the 64 step selection process is given by the probability that it correctly negotiates $64 - L$ separations for which it matches the input ciphertext times the probability that it commits an error at the L separations for which it mismatches the correct ciphertext:

$$(1 - E)^{64-L} E^L$$

There are $2^{56} X$ complexes in the codebook so the expected number of distractors is given by:

$$\sum_{L=0}^{64} \frac{2^{56} X}{2^{64}} \binom{64}{L} (1 - E)^{64-L} E^L = \frac{X}{256}$$

Perhaps surprisingly, the expected number of distractor molecules is independent of the error rate. Note, in particular, that for $X=1$ the expected number of distractors is less than one.

4.3 Feasibility

Combining the results from 3.2 and 3.3 gives the following table which shows for various error rates, the amount of DNA that must be pushed through the DES computation to insure that there is a *reasonable chance* (63%) of getting a winning key in the final tube. The table also records the expected number of distractors which will be present in the final tube.

achievable error rate E	0	10^{-4}	10^{-3}	10^{-2}
X	1	2	830	2.1×10^{29}
grams of DNA required	.7	1.4	580	1.5×10^{29}
distractors	.004	.008	3.2	8.3×10^{26}

This indicates that for an error rate of 10^{-4} , a little more than 1 gram of DNA is needed and the final tube will usually have no distractors. If an error rate of 10^{-3} is achievable, then less than a kilogram of DNA is needed and only a small number of distractors need be dealt with before finding the correct answer. However, if an error rate of 10^{-2} is the best that is attainable, then huge amounts of DNA are needed (approximately 23 Earth masses) to have a *reasonable chance* that a winning key ends in the final tube, but even then it will have to be distinguished from a colossal number of distractors - clearly an unacceptable situation. In such cases, where only very high error rates are possible, techniques like those described in [Ro] (for example, a *refinery algorithm*) may be used to reduce the amount of DNA required. In section 5.8 of [Ro] Roweis *etal.* give a brief analysis of the application of a refinery algorithm to DES.

5 Conclusions

We wish to emphasize that our description of an attack on DES is, at this point, entirely theoretical and whether it can be carried out in the lab remains to be seen. Huge challenges remain. For example, as yet, we have been unable to perform *separations* (or any of the sticker operations) in the lab with error

rates approaching 10^{-4} . Nonetheless, the analysis presented in this paper demonstrates (at least in principle) two things:

- “Real problems” can be solved with small machines which do not require huge amounts of DNA and use little or no enzymes.
- Error rates similar to those normally demanded of electronic computers are not required.

If the attack on DES described here can be carried out in the lab, then some other cryptosystems might also be vulnerable to this approach. Indeed, the small size of the machine we describe suggests that systems like the 64-bit key FEAL cryptographic system of Shimizu-Miyaguchi [Sh] might be susceptible to such an attack.

Finally, there are several messages for cryptography in these findings. First, it seems appropriate to reconsider one of the “axioms” of cryptography: Improvements in computational power always favor the cryptographer over the cryptanalyst. This is almost certainly untrue. The analysis presented here suggests the possibility of computers with super-parallelism that can help the cryptanalyst immensely, yet provide no help for the cryptographer. Even if DNA computers prove infeasible, it is possible that new machines capable of super-parallelism may make cryptosystems like DES insecure.

The DNA computer is an example of a super-parallel machine with very slow processors (complexes of DNA). The potential vulnerability of DES arises for two reasons. First, the key space is insufficiently large. Second, the DES algorithm is “too short.” The fact that only 6655 steps are needed to do an encryption allows the slow DNA processors to finish their encryptions in (at least in theory) a reasonable amount of time. Hence the much valued throughput speed of DES and similar systems, may carry with it a potential vulnerability to super-parallel machines with slow processors.

Acknowledgments

Leonard M. Adleman and Paul W.K. Rothmund are supported in part by the National Science Foundation under grant CCR-9403662 and the Sloan Foundation. Sam Roweis is supported in part by the Center for Neuromorphic

Systems Engineering as a part of the National Science Foundation Engineering Research Center Program under grant EEC-9402726 and by the Natural Sciences and Engineering Research Council of Canada. Erik Winfree is supported in part by National Institute for Mental Health (NIMH) Training Grant # 5 T32 MH 19138-06; also by General Motors' Technology Research Partnerships program.

References

- [Bo1] Boneh, D., Lipton, R.: “Batching DNA Computations”, Princeton CS Tech-Report CS-TR-489-95.
- [Bo2] Boneh, D., Dunworth, C., Lipton, R.: “Breaking DES Using a Molecular Computer”, Princeton CS Tech-Report CS-TR-489-95.
- [Li] Lipton, R.: “Using DNA to solve NP-Complete Problems”, *Science* 268:542–545, April 1995. January 1977.
- [Na] National Bureau of Standards: “Data Encryption Standard”, U.S. Department of Commerce, FIPS, pub. 46, January 1977.
- [Ro] Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothmund, P.W.K., Adleman, L.M.: “A Sticker Based Model for DNA Computation”, this volume.
- [Sh] Shimizu, A., Miyaguchi, S.: “Fast Data Encipherment Algorithm Feal”, *Lecture Notes in Computer Science* 304:267–278, 1988.
- [Wi] Wiener, M.: “Efficient DES Key Search”, TR-244, School of Computer Science, Carleton University, May 1994.