Using Content-Addressable Search Engines To Encrypt and Break DES

Peter C. Wayner
Computer Science Department
Cornell University
Ithaca NY 14853

Abstract

A very simple parallel architecture using a modified version of content-addressable memory can be used to cheaply and efficiently encipher and decipher data with DES-like systems. The paper will describe how to implement DES on these modified content-addressable memories at speeds approaching some of the best specialized hardware. The chips can also be used to build a large scale engine for exhaustively searching the entire keyspace of DES.

The original DES system was designed to be easily implemented in hardware [NBS77] and the current silicon manifestations of the cipher use modern processor design techniques to encipher and decipher information at about 1 to 20 megabits per second. Implementations of DES in software for standard CPUs, however, are markedly slower than specialized chips because many of the operations involved in DES are bit-level manipulations. As a result, many of the DES-like systems such as Merkle's Khufu [Mer90] were designed as replacements that could be easily implemented on conventional hardware.

There is one class of general architecture, however, that implements bit-level operations. The machines like the CM-1, CM-2 and CM-200 from Thinking Machines Corporation and the Maspar machine all have thousands of one-bit processors. The designers intended that large number of processors would make up for the deficencies of the individual nodes.

Another example of this small architecture is now emerging from the labs of memory designers who are trying to build sophisticated content addressable memory. The individual processors of these machines are even weaker than the ones of the CM-1, but they can be packed very densely on a chip. The tiny processors have only a fraction of the memory of a CM-1 (42 bits versus thousands) and only a one dimensional interconnection network (vs 12), but this is sufficient to implement DES successfully. Moreover, the packing density (1024 processors per chip) ensures that a significant amount of parallelism is very cheap.

Implementing the cipher on generalized parallel architectures like this have one main advantage– cost. Many computer designers often find that the tradeoff between the speed of a specialized DES chip is often not worth the price. Generalized, content-addressable machines, however, have many other applications and this makes them a good compromise for the system designer. This paper will describe how to implement the algorithm on this architecture and produce results that are on par with the best specialized hardware.

1 Content-Addressable Memory Machines

Standard memory maps an address to a value. Unfortunately, there are many applications when an algorithm needs to know which memory location holds a particular value. The only recourse is to search all the memory to find the value in question. Content-addressable memory is a hardware solution to this problem that will invert the search and provide the address holding a value in only one operation. The technique has been well-researched over the years and the book by Kohonen [Koh87] notes many approaches and summarizes some of the more salient research. Several companies including AMD are making basic content-addressable memory modules.

Recently teams at Syracuse University (some publications include [Old86, OWN87, OSB87]), MIT and Cornell ([Bri90, WS89, Zip90]) have developed more sophicated and powerful implementations in silicon. They allow the programmer to chain the result of several searches together in a simple fashion so that larger data structures and more complicated searches can be performed in hardware. Some of this hardware was originally intended to speed up logic programming, but many people have found surprising and interesting applications for the simple hardware. Oldfield and his team at Syracuse, for instance, are currently working on compressing data.

A company, Coherent Research Incorporated of Syracuse, New York, is building sophisticated content-addressable memory chips called the Coherent Processor for widespread use. This paper will use their chip as an example because it is commercially available, but there is no reason why the algorithms cannot be modified slightly for use on similar chips.

At the basic level, the Coherent Processor is a large, single dimensional array of very simple parallel processors. Each processor has 42 bits of memory ($W_i[0] \ldots W_i[41]$, the i denotes the processor number) and three one-bit registers (R_1, R_2 and R_3). It also has a processing unit that can execute instructions on the registers, transfer data between the registers and the memory, communicate with the two neighboring processors or match a value on the internal bus. The instructions are simple boolean operations that read three bits of memory and store the result in a third. The match instructions can be used to simultaneously compare one 42-bit value against the entire array of processors. If there is a match, then the appropriate value is placed in a register.

The following table shows the basic Coherent Processor instructions and the number of clock cycles used to complete them.

1. *MATCH* Simultaneously compare the 42 general bits at each processor with the values on a bus and store the result of this match in R_1. This is used to look up

items quickly. The match routine can include wild-card matches for individual bits so it is possible to match for strings of bits like "0000******11*****" (a "*" matches both a "0" and a "1"). If you want to move the value of bit $W_i[2]$ into R_3, then you would "match" a pattern with 1 in bit $W_i[2]$ and wild-card matches specified for the rest and store the result in R_3. If the value of bit $W_i[2]$ was 1 in a particular word, then the match would be successful and a 1 would be stored in R_3. If a zero was in bit $W_i[2]$, then the match would be unsuccessful and a zero would be stored. The values of the other columns would not be affected. `Cost: 4 cycles.`

2. *CALC* Calculate a three-bit function of the three registers and store the result in a third register. `Cost: 2 cycles.`

3. *READ* Take the result of a selected word and place it on the bus. This operation usually follows a MATCH operation. `Cost: 3 cycles.`

4. *WRITE* Move the result from the bus into the selected word(s). `Cost: 2 cycles.`

5. *SHIFT* The first registers of each word are interconnected. They can shift the bit in their register to adjacent words in one step. `Cost: 2 cycles.`

6. *WRITECOLUMN* Moves a bit from a register into one of the 42 bits of memory. `Cost: 2 cycles.`

These commands can be strung together to manipulate data in simple and straightforward methods.

2 Implementing Plain DES

There are three main operations involved in encrypting a block of 64 bits with the basic mode of the Data Encryption Standard known as the Electronic Code Book (ECB). They are 1) permuting the bits, 2) passing a 32-bit block through an s-box and 3) permuting the key structure. Each of these steps is easy to program on the Coherent Processor , in a large part because the architecture is so limited. Several features of the instruction set, however, make implementing the algorithm very easy.

Let the plaintext blocks of data be denoted, $\{B_1, \ldots, B_n\}$ and the individual bits of block B_i be $\{B_i[0] \ldots B_i[63]\}$. The key is K and the individual bits are $K[0] \ldots K[55]$.

There are sixteen rounds of encryption and a key scheduling algorithm chooses a 32 bit subset of key bits to be used on each round. Let $K^{(l)}[0] \ldots K^{(l)}[31]$ be the 32 bits used in round l. Each block of 64 bits is broken into two, 32-bit halves (called B_L and B_R) and in each round the value of one of the halves is mixed with a subset of the key bits, passed through the s-box and then mixed with the other 32-bit half. More precisely, in each round:

$$B_R \leftarrow B_R \oplus f(B_L \oplus K^{(l)}).$$

Then B_L and B_R are exchanged.

The data to be encrypted is broken into 64-bit blocks and each block is stored in 32-bit halves in two adjacent 42 bit words in the array, W_i and W_{i+1}.

2.1 Permuting the Bits

At the beginning and the end of the encryption process, the 64 bits in the block are passed through a bit-wise permutation. This step is often considered the slowest part of many software implementations for general purpose machines and many people believe that it was included to slow down software implementations and force general CPUs to move bits one by one. The Coherent Processor must also move each bit one at a time, but at least this is the best that it can do. In practice, the large number of parallel processors makes up for the weakness.

Let the permutation be written, $W_i[p_0] \rightarrow W_j[p_1] \rightarrow \ldots \rightarrow W_i[p_{63}] \rightarrow W_i[p_0]$. The process can be accomplished by stringing together a chain of bit moving commands. When the bits to be exchanged are on different words, then the CAM must also execute a bit-passing command to swap the bit to the adjacent word. The work can be summarized in pseudo-code:

```
Move $W_i[p_0]$ into a bit .
for k:=1 to 63 do
    Move $W_i[p_k]$ into a bit.
    Move $W_i[p_{k-1}]$ into its destination.
    If $W_i[p_k]$ is on the wrong word,
        then pass it to the correct one.
Move $W_i[p_{63}]$ into $W_i[p_0]$ .
```

There are only 32 bits that need to be shifted between words. It is possible to do this quickly. The next section which computes the values of the s-boxes is much more time intensive.

Cost: 129 MATCH or WRITECOLUMN instructions, 32 SHIFT instructions. 580 cycles.

2.2 Computing the S-boxes

The s-box are responsible for providing the non-linear mixing of the bits that is necessary to provide adequate security. At the highest level, the s-box is a function that maps 32 bits to 32 other bits. The s-boxes used in DES are, though, much simpler and they can be described as eight functions that take 6 out of the 32 bits and return four. Some bits are used more than others. These eight s-boxes can be further simplified into 32 functions that map six bits to one bit and this is the best level of abstraction to use when programming the Coherent Processor .

Meyer and Matyas [MM82] describe the design of the S-boxes in terms of *minterms*, which are roughly the same as clauses of boolean variables. An equation describing output of one bit of an s-box might look something like this:

$$B_i[1] \cdot \neg B_i[2] \cdot B_i[3] \cdot B_i[4] + B_i[1] \cdot \neg B_i[5] \cdot \neg B_i[6] + B_i[2] \cdot B_i[5]. \tag{1}$$

("$\cdot$"=boolean and, "+"=boolean or, "$\neg$"=boolean not.) There are three minterms in the example and it is generally believed that the number of minterms in a minimal expression is one measure the complexity of the s-box. The recent papers by Biham and Shamir [BS91] and others , show that there are additional ones that are more important. Meyer and Matyas note that there are 52 and 53 minterms in the description of each of the 8 s-boxes.

These minterm descriptions of the s-boxes can be directly converted into operations for the Coherent Processor . Each clause of variables to be ANDed together can be computed with a MATCH equation with appropriate set of ones for the variables in the clause, zeros for the negated variables in the clause and wildcards for the unrepresented variables. The expression from equation 1 can be encoded:

$$\begin{array}{l} MATCH``1011***\ldots***'' \rightarrow R_1 \\ CALC R_1 \rightarrow R_2 \\ MATCH``1***00**\ldots***'' \rightarrow R_1 \\ CALC R_1 \cdot R_2 \rightarrow R_2 \\ MATCH``*1**1**\ldots***'' \rightarrow R_1 \\ CALC R_1 \cdot R_2 \rightarrow R_1 \end{array} \tag{2}$$

This takes 6 cycles per minterm. At 53 minterms per s-box and 8 s-boxes per encryption round, this takes 2544 cycles per encryption round to calculate the values of the bits. It takes one SHIFT, one MATCH, one CALC and one COLUMNWRITE to XOR each of the 32 bits into the adjacent word. That is an additional 384 cycles for 2928 per encryption round. There are 16 rounds in DES, the permutations take 580 cycles and the overall encryption process takes 47,528 cycles.

2.3 Handling the Key

When the result of one of the 32 functions is computed it must be xor-ed with the key and then passed to the adjacent word to be xor-ed with the appropriate bit. The same key encrypts all the blocks at the same time and it can be included by XORing the key vector, $K^{(l)}$, into the match words. For instance, assume that "11001100 10101110 01001100 11100101" is the 32 bits of key being used in a round and the minterms from equation 1 define the s-box equations. Then the operations in example 2 become:

$$\begin{array}{l} MATCH``0111***\ldots***'' \rightarrow R_1 \\ CALC R_1 \rightarrow R_2 \\ MATCH``0***11**\ldots***'' \rightarrow R_1 \\ CALC R_1 \cdot R_2 \rightarrow R_2 \end{array}$$

$$\begin{aligned} &MATCH``*0**0**\ldots***'' \rightarrow R_1 \\ &CALC R_1 \cdot R_2 \rightarrow R_1 \end{aligned} \tag{3}$$

The same key is used to encrypt or decrypt each block of data in the simple version of DES. There are 56 key bits, but only 32 of them are used during each of the 16 different rounds. The bits being used are maintained by the program running on the general machine that is driving the Coherent Processor . It selects the subset of 32 bits that are used in each encryption and modifies the s-box functions accordingly.

This method presupposes that the sixteen 32-bit subsets of the keys are precomputed and "compiled" into the code. This process is non-trivial and certain to cost some time. When the amount of data encrypted or decrypted per key change is large, then this "compilation" time is minimal. If the key is changed frequently,then there may be some significant impact on the encryption times. A better understanding of the effects of this will need to wait until the software is completely implemented on a working system.

2.4 The Total Cost

The current version of the Coherent Processor will run at speeds up to 50 MHtz. If an encryption takes about 47,428 cycles, then each pair of words in the processor array can encrypt about 1,000 64-bit blocks per second. Writing a word into the array and reading it out takes 5 cycles in total. One chip of the current model has 1024 words or processors, so it can read in, encrypt and write out blocks of 32K in 52,548 cycles. This is equivalent to 31.2 megabits per second– something that is in line with the middle range of current DES chips. The Cryptech CRY12C102 data sheet reports that it runs at 22.5 megabits per second and the Pijnenburg PCC100 gets 20 megabits per second. Moreover, the Coherent Processor is designed to be easily expanded by linking together multiple copies of the chip and n chips will n times faster for small numbers of n. When there are hundreads or thousands of chips, the cost of writing and reading the information from the Coherent Processor becomes the limiting factor. Coherent Research reports that the new chip will cost about $100 per copy in small quantities and substantially less in large ones.

3 Exhaustive Attack on DES

When DES was introduced in 1977, some computer scientists protested that 56 bits were not sufficient because it would be possible to do an exhaustive search of the key space in a short amount of time using a massively parallel computer. [DH77] In their book, Meyer and Matyas [MM82] discount that possiblity and predict that it would just not be physically possible to build the machine until the 1990's because there were too many physical limitations. Heat and power usage are two major barriers.

How easy would it be to build one today? Standard off-the-shelf encryption chips are plentiful and relatively cheap, but they require a second processor feeding them the keys and the test cases. Anyone who wants to build such a machine must undertake a project of building such a large array of distributed computers. This would require a large amount of custom design work. A truly dedicated attacker could even fabricate custom DES testing chips which have a built in circuit for incrementing the key by one bit and testing the result against another register. Only governments could afford a budget this large. Moreover, the slightest change in the algorithm would render this machine worthless.

Garon and Outerbridge calculated the approximate costs of designing such a machine and found that it would cost about \$129,000 for a machine that would break DES within 1 year if the machine was built in 1990. [GO91] They assume that it is possible to build a node that encrypts 2 million key tests for \$25 in order to complete such a machine. They do not describe the details of how to design the board or manufacture it is sufficient quanties.

The Content Addressable Memory array chips, however, are designed to be built into large parallel arrays of chips. It is already possible to buy a board for a PC which has 64 chips of a previous model of the Coherent Processor . Large arrays should not be hard to create. Moreover, the algorithm is implemented in software, so the machine can also be used to attack many other subtle and not-so-subtle variations of DES.

What is the best way to do an exhaustive search with the current architecture of the Coherent Processor? The version described for simple encryption and decryption is able to work very quickly because it can encode the key in the stream of instructions fed to the Coherent Processor. This approach must be abandoned because an exhaustive search of the key space requires that each processing node must use a different key.

One alternative is to store the key bits in the 10 extra tag bits stored at each node. Two nodes are used to hold the two 32-bit half-blocks of each case, so there are up to 20 extra key bits which can be stored at each node. Let there be 2^n processors in the machine. That means there are 2^{n-1} potential keys that can be tested with each round because two nodes are used for each encryption. Assume that $n \leq 21$ and the problem does not overflow the physical space of the real machine. (Later versions of the architecture could have more free bits available.) At each pair of nodes, store a unique set of $n-1$ key bits. These bits will be used by this pair of nodes alone. The other $56-(n-1)$ bits are shared by all the instances and they are encoded in the instruction stream as before.

At the beginning of each round of encryption, the local key bits must be XOR-ed into the appropriate half-block of bits before that half-block is passed through the S-boxes. These four or five instructions will XOR in the key bit K_i in to position B_j:

$$\begin{array}{l} MATCHK_i \rightarrow R_1 \\ SHIFT \\ MATCHB_j \rightarrow R_2 \\ CALCR_1XORR_2 \rightarrow R_2 \\ WRITECOLUMNR_2 \rightarrow B_j \end{array} \tag{4}$$

The SHIFT instruction is only necessary if the key bit is on the opposite node from the destination bit. This process is repeated at the end of the S-box calculation to remove the bit from the data. Only 32 of the 56 key bits are used at each round, but it is possible that up to $n-1$ of these bits will come from the bits stored locally. The operations in equation 4 take 16 cycles. They must be repeated $2n-2$ times for each round. The result takes $512n-512$ extra cycles for each encryption. If a machine was built with a full complement of 2^{21} processors, then it would take 57,126 cycles to test 2^{20} potential keys. This step must be repeated 2^{36} times and the machine is capable of doing about 875 of these tests per second or about 76 million per day. Exhausting the entire space would require 904 days. If the well-known trick of exploiting symmetry in the keys is used to reduce the key space to 2^{55} keys, then one machine will test all in 452 days.

How much would such a machine cost? There are 2^{10} processors on a chip that will cost between \$30 and \$100. 2^{11} chips are necessary and this would cost between about \$60,000 and \$200,000. Control hardware would add additional \$10,000 to \$20,000. 45 machines would cost about \$3 million dollars and exhaustively search the space in 10 days or \$30 million to search the space in 1 day with 450 machines. I'm assuming that volume discounts would apply at this scale. This is a price that should apply at the end of 1992 when the chips become widely available.

The standard assumptions about time and transistor density should apply to this model as well. It is entirely conceivable that we will see large changes in density and price of these machines in the near future and this should prove to be a major threat to the security of DES and by logical extension, the UNIX password system. This large machine made up of CAM is reprogrammable so it is possible to retool it to mount an exhaustive search on DES-like system where either the s-boxes or the scheduling algorithms were constrained.

4 DES with Modified Chaining

The last several section described how to encrypt a large block of data in parallel using a simple DES with no feedback. A more robust version of DES feeds the result of encrypting each block into the key selection of the next block. Let $E_i = f(K, B_i)$ represents the cipher text blocks. A feedback cipher sets $E_i = f(K, B_i \oplus E_{i-1})$. "$\oplus$" represents boolean XOR. E_0 is set to a pre-arranged constant. This process is called Cipher Block Chaining (CBC).

The modification adds a great deal of strength to the plain DES because it reduces the redundancies that can developed if there is an 64 bit block that occurs often in the plaintext. The feedback mode ensures that a different value will permute each block and obscure the redundancy. It should be obvious that this system cannot be used when all the blocks are computed in parallel. Here is a modified version of chaining that can be implemented in parallel.

One solution is to exchange and XOR bits with neighbors at the end of certain rounds of encryption. In round 1, the left half of each block is used to compute the value XORed into the right half. After this, the left blocks are exchanged with the neighboring blocks and XOR'ed into the right halves of the neighboring block. This can be done with pseudo-code like this. W_i is the left half and W_{i+1} is the right half.

```
for k:=0 to 31 do
    MATCH $W_i[k] \rightarrow R_1$
    CALC COPY $R_1 \rightarrow R_2$
    SHIFT
    SHIFT
    CALC XOR $R_1 R_2 \rightarrow R_1$
    WRITECOLUMN $R_1 \rightarrow W_{i+1}[k]$
```

This command shifts one bit to the next pair of words over and xor's it with the value of a neighboring block. It takes 16 cycles per bit to do it. This can be repeated as often as desired at the cost of slowing down the entire encryption. Doing this at the end of each round of encryption costs 8,192 cycles and this slows the encryption rate to 27.0 megabits per second. In this case, a change in block B_i will propigate through blocks B_i to B_{i+16} and effect their encrypted values. Arbitrarily complex shifting can be included as long as care is taken to ensure that the results can be reversed. If this step is done often in the process, it can effectively turns the encryption into one large block at a small decrease in speed.

5 Conclusion

This paper has shown that a simple architecture intended for information storage and retrieval can encrypt and decrypt messages faster than many of the best specialized chips. Naturally, the results show that the process can be extended to other DES-like systems. The only problem is expressing the s-boxes so they can be implemented with minterms.

Chips like the Coherent Processor also makes it very easy to create a large-scale processor for exhaustive cryptanalysis of the key space because the chips were designed to be grouped together in a large array. The hypothetical machine described here is much different from the other machines described in the literature because it is substantially closer to being created. The next several generations of this architecture should pose a large threat to wide-spread use of the old DES.

There are several changes to the Coherent Processor that would make it better able for encrypting DES. Currently, the key is "compiled" into the program for the CAM and this may be a non-trivial event. If future versions of the architecture have more that 42 bits per word, then it could be practical to store the key locally and add the key in bit by bit as it is done in the brute force attack. Also, the current version of the Coherent Processor will only compute 3 bit functions. 4 or 5 bit functions may be quite practical and they would speed the results of the process.

This draft of the paper represents a preliminary look at the problem of implementing DES on the special CAM architecture. Better numbers will be available when working hardware is available.

6 Acknowledgements

The author would like to thank Chuck Stormon at Coherent Research for taking the time to teach me how to program the Coherent Processor and making many valuable comments about the structure of this paper.

References

[Bri90] Sharon Marie Britton. *8k-trit Database Accelerator with Error Detection.* PhD thesis, Massachusetts Institute of Technology, February 1990.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of snefru, khafre, redoc-ii, loki, and lucifer. In *Crypto 91*, Santa Barbara, California, 1991.

[DH77] Whitfield Diffie and Martin Hellman. Exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, 10(6):74–84, 1977.

[GO91] Gilles Garon and Richard Outerbridge. Des watch: And examination of the sufficiency of the data encryption standard for financial institution's information security in the 1990's. *Cryptologia*, 15(3):177–193, July 1991.

[Koh87] Teuvo Kohonen. *Content-Addressable Memories.* Springer-Verlag, Berlin, New York City, 1987.

[Mer90] Ralph Merkle. Fast software encryption function. In A.J. Menezes and S.A. Van Stone, editors, *Crypto 90*, Berlin, New York City, 1990. Springer Verlag.

[MM82] Carl H. Meyer and Stephen M. Matyas. *Cryptography: New Dimension in Computer Security.* John Wiley and Sons, New York, 1982.

[NBS77] NBS. Data encryption standard (des). Technical report, National Bureau of Standards (US), Federal Information Processing Standards, Publication 46, National Technical Information Services, Springfield, Virginia, April 1977.

[Old86] J.V. Oldfield. Logic programs and an experimental architecture for their execution. *Procedings of the I.E.E.E. Part E*, 133:163–167, 1986.

[OSB87] J.V. Oldfield, Charles D. Stormon, and M.R. Brule. The application of vlsi content-addressable memories to the acceleration of logic programming systems. In *CompEuro 87, VLSI and Computers*, pages 27–30, Hamburg, Germany, May 1987.

[OWN87] J.V. Oldfield, R.D. Williams, and N.E.Wiseman. Content-addressable memories for storing and processing recursively-divided images and trees. *Electronics Letters*, 23(6):262–263, 1987.

[WS89] John Wade and Charles Sodini. A ternary content-addressable search engine. *IEEE Journal of Solid-State Circuits*, 24(4):1003–1013, August 1989.

[Zip90] Richard Zippel. Programming the data structure accelerator. In *Proceedings of Jerusalem Conference on Information, Technology*, Jerusalem, Israel, October 1990.