A Cryptanalytic Time–Memory Trade-Off

MARTIN E. HELLMAN, FELLOW, IEEE

Abstract—**A probabilistic method is presented which cryptanalyzes any N key cryptosystem in $N^{2/3}$ operations with $N^{2/3}$ words of memory (average values) after a precomputation which requires N operations. If the precomputation can be performed in a reasonable time period (e.g., several years), the additional computation required to recover each key compares very favorably with the N operations required by an exhaustive search and the N words of memory required by table lookup. When applied to the Data Encryption Standard (DES) used in block mode, it indicates that solutions should cost between $1 and $100 each. The method works in a chosen plaintext attack and, if cipher block chaining is not used, can also be used in a ciphertext-only attack.**

I. Introduction

MANY SEARCHING tasks, such as the knapsack [1] and discrete logarithm problems [2], allow time–memory trade-offs. That is, if there are N possible solutions to search over, the time–memory trade-off allows the solution to be found in T operations (time) with M words of memory, provided the time–memory product TM equals N. (Often the product is of the form $cN \log_2 N$, but for simplicity we neglect logarithmic and constant factors.)

Cryptanalysis is a searching problem that allows the two extremes of exhaustive search ($T=N$, $M=1$) and table lookup ($T=1$, $M=N$), but until this paper no general time–memory trade-offs had been published. Letting m and t be parameters whose significance will be explained later and neglecting precomputation, this technique requires approximately $M=mt$ words of memory and $T=t^2$ operations provided $mt^2=N$. Letting $m=t=N^{1/3}$ results in $M=T=N^{2/3}$, which is much more cost effective than exhaustive search and table lookup. If complexity is measured by $M+T$ this technique reduces the effective key length by one-third when judged against exhaustive search. Breaking the 56-bit Data Encryption Standard (DES) with this method is less complex than doing an exhaustive search on a 38-bit key system.

Complexity and cost are not synonymous because memory costs more than time. But Section III shows that the cost per solution of breaking the DES drops from approximately $5000 for exhaustive search to approximately $10 using the time–memory trade-off.

This time–memory trade-off is not as good as those known for the knapsack and discrete logarithm problems, where $M=T=N^{1/2}$ can be obtained and where the precomputation is no more complex than the search itself. This indicates that improvements may well be possible.

Exhaustive search can be accomplished under a known plaintext attack, while table lookup requires a chosen plaintext attack [3]. In an exhaustive search, the ciphertext can be deciphered under each key and the result compared with the known plaintext. If they are equal, the key tried is probably correct. Occasional false alarms are rejected by additional tests.

In table lookup, the cryptanalyst first enciphers some fixed plaintext P_0 under each of the N possible keys to produce N ciphertexts. These are sorted and stored in a table with their associated keys.

When a user chooses a new key K, he is forced (in a chosen plaintext attack) to provide the cryptanalyst with the encipherment of P_0

$$C_0 = S_K(P_0), \tag{1}$$

where $S_K(*)$ denotes the enciphering operation under key K. Because the table is sorted by ciphertext, the cryptanalyst can find C_0 and its associated key in at most $\log_2 N$ operations using a binary search. Either by neglecting logarithmic factors or through hash coding [4], this will be counted as one operation.

The N operations required to compute the table are not counted because they constitute a precomputation which can be performed at the cryptanalyst's leisure. In the real world, we must ensure that the precomputation is not excessive, and this will be done in Section III for the time–memory trade-off applied to the DES.

Those unfamiliar with cryptography often question the validity of using a known or chosen plaintext attack in assessing the strength of a system. They think of a ciphertext-only attack in which the cryptanalyst possesses only ciphertext and some statistical knowledge of the plaintext. Aside from the fact that one should be conservative in assessing security levels, a successful known or chosen plaintext attack can often be modified to work under the ciphertext-only assumptions. Reference [5] explains one such instance, where the parity bit in the American National Standard Code for Information Interchange (ASCII) allows a known plaintext attack to be turned into a ciphertext-only attack.

Similarly, the chosen plaintext assumption required for table lookup can often be relaxed. If the DES is used in block mode, the cryptanalyst can choose P_0 to be a frequent plaintext block, such as the ASCII representation of eight blanks. He then inspects the ciphertext for re-

Manuscript received October 24, 1978; revised October 16, 1979. This work was supported in part by the National Science Foundation under Grants ENG 10173 and ECS 16161. This paper was presented at the IEEE International Symposium on Information Theory, Grignano, Italy, June 25–29, 1979.

The author is with the Department of Electrical Engineering, Stanford University, Durand 135, Stanford, CA 94305.

0018-9448/80/0700-0401$00.75

peated blocks and uses his cryptanalytic approach on each repeated block. If there are ten repeated blocks, this only increases his effort by a factor of ten. The correct solution is easily determined through additional tests.

The cryptanalytic technique described in this paper requires the same kind of chosen plaintext attack as does a table lookup. We use the chosen plaintext assumption to simplify explanations, but it should be remembered that the time–memory trade-off can also be used with a ciphertext-only attack by looking for repeated ciphertext blocks and assuming that they correspond to the chosen (frequent) plaintext.

II. Iterative Approach

The time–memory trade-off is best understood by considering a specific cryptosystem, such as the DES. It operates on a 64-bit plaintext block P to produce a 64-bit ciphertext block C under the action of a 56-bit key:

$$C = S_K(P). \tag{2}$$

Letting P_0 be a fixed plaintext block, define

$$f(K) = R[S_K(P_0)], \tag{3}$$

where R is some simple reduction from 64 to 56 bits, such as dropping the last 8 bits of the ciphertext. Fig. 1 depicts the construction of f.

Computing $f(K)$ is almost as simple as enciphering, but computing K from $f(K)$ is equivalent to cryptanalysis. If the cryptosystem is secure, f is therefore a one-way function [3]. The time–memory trade-off described in this paper applies to inverting any one-way function, not just those derived from cryptosystems.

As part of the precomputation, the cryptanalyst chooses m starting points, $\mathrm{SP}_1, \mathrm{SP}_2, \cdots, \mathrm{SP}_m$, each an independent random variable drawn uniformly from the key space $\{1,2,\cdots,N\}$. For $1 \leqslant i \leqslant m$ he lets

$$X_{i0} = \mathrm{SP}_i \tag{4}$$

and computes

$$X_{ij} = f(X_{i,j-1}), \qquad 1 \leqslant j \leqslant t \tag{5}$$

as depicted in Fig. 2. The parameters m and t are chosen by the cryptanalyst to trade-off time against memory, as discussed below.

The last element or endpoint in the ith chain (or row) is denoted by EP_i. Clearly

$$\mathrm{EP}_i = f^t(\mathrm{SP}_i). \tag{6}$$

To reduce memory requirements, the cryptanalyst discards all intermediate points as they are produced and sorts the $\{\mathrm{SP}_i, \mathrm{EP}_i\}_{i=1}^m$ on the endpoints. The sorted table is stored as the result of this precomputation.

Now suppose someone chooses a key K and the cryptanalyst intercepts or is given

$$C_0 = S_K(P_0). \tag{7}$$

He can apply the reduction operation R to obtain

$$Y_1 = R(C_0) = f(K). \tag{8}$$

He can check if Y_1 is an endpoint in one "operation" because the $\{(\mathrm{SP}_i, \mathrm{EP}_i)\}$ are sorted on the endpoints.

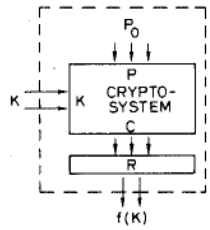

Fig. 1. Construction of the function f.

$$\begin{aligned}
&\mathrm{SP}_1 = X_{10} \xrightarrow{f} X_{11} \xrightarrow{f} X_{12} \xrightarrow{f} \cdots \xrightarrow{f} X_{1t} = \mathrm{EP}_1\\
&\mathrm{SP}_2 = X_{20} \xrightarrow{f} X_{21} \xrightarrow{f} X_{22} \xrightarrow{f} \cdots \xrightarrow{f} X_{2t} = \mathrm{EP}_2\\
&\qquad\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\
&\mathrm{SP}_m = X_{m0} \xrightarrow{f} X_{m1} \xrightarrow{f} X_{m2} \xrightarrow{f} \cdots \xrightarrow{f} X_{mt} = \mathrm{EP}_m
\end{aligned}$$

Fig. 2. Matrix of images under f.

If Y_1 is not an endpoint the key is not in the next to the last column in Fig. 2. (If it were, Y_1, which is its image under f, would be an endpoint.)

If $Y_1 = \mathrm{EP}_i$, either $K = X_{i,t-1}$ (i.e., K is in the next to last column of Fig. 2), or EP_i has more than one inverse image. We refer to this latter event as a false alarm. If $Y_1 = \mathrm{EP}_i$, the cryptanalyst therefore computes $X_{i,t-1}$ and checks if it is the key, for example by seeing if it deciphers C_0 into P_0. Because all intermediate columns in Fig. 2 were discarded to save memory, the cryptanalyst must start at SP_i and recompute $X_{i,1}, X_{i,2}, \cdots$, etc. until he reaches $X_{i,t-1}$.

If Y_1 is not an endpoint or a false alarm occurred, the cryptanalyst computes

$$Y_2 = f(Y_1) \tag{9}$$

and checks if it is an endpoint. If it is not, the key is not in the $t-2$nd column of Fig. 2, while if $Y_2 = \mathrm{EP}_i$, the cryptanalyst checks if $X_{i,t-2}$ is the key. In a similar manner, the cryptanalyst computes $Y_3 = f(Y_2), \cdots, Y_t = f(Y_{t-1})$ to check if the key is in the $t-3$rd, $\cdots$, or 0th column of Fig. 2.

If all mt elements in the 0th through $t-1$st columns of Fig. 2 are different and if K is chosen uniformly from all possible values, the probability of success $P(S)$ would be mt/N. Only m words of memory and t operations are required, so the time–memory product has come into play. An exhaustive search with t operations has only $P(S) = t/N$, while a table lookup with m words of memory has only $P(S) = m/N$.

If the matrix in Fig. 2 has some overlap, but a fixed fraction of distinct elements, the probability of success is only lowered by the same fixed fraction. A mild amount of overlap therefore can be tolerated in the matrix without affecting the basic gain inherent in the time–memory trade-off. This analysis also neglects other constant and logarithmic factors (e.g., it counts an encipherment, reduction operation, and check for Y_1 equal to an endpoint as one operation).

Theorem: If $f(*)$ is modeled as a random function mapping the set $\{1,2,\cdots,N\}$ into itself, and if the key K

is chosen uniformly from this same set, then the probability of success is bounded by

$$P(S) \geq (1/N) \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N-it)/N]^{j+1}. \quad (10)$$

Remark 1: Equation (10) indicates that for a fixed value of N there is not much to be gained by increasing m or t beyond the point at which $mt^2 = N$. Because $[(N-it)/N]^{j+1} \doteq \exp(-ijt/N)$, the last term is closely approximated by $\exp(-mt^2/N)$ and when $mt^2 \gg N$ most terms will be small. (Most will have values of i and j which are a significant fraction of m and t, respectively.) If $mt^2 \ll N$, each term in (10) is close to one and (10) reduces to

$$P(S) \geq mt/N, \quad (11)$$

which is also an upper bound so there is negligible overlap. Increasing either m or t then produces a significant effect. If $mt^2 = N$, with both m and t large, then (10) can be numerically evaluated and equals $0.80mt/N$ to two significant figures. (Approximating the sum by an integral and scaling m and t shows that the fractional efficiency, 0.80, is independent of m and t so long as the product mt^2 is unaltered.) Operating at $mt^2 = N$ therefore increases the expected cryptanalytic effort by at most the small constant factor $1/0.80 = 1.25$. We will often neglect this slight increase in cryptanalytic effort. Numerical evaluation of (10) can be avoided at the expense of some looseness. Approximating each term by $\exp(-ijt/N)$, lower bounding these by $\exp(-it^2/N)$ and summing predicts an efficiency of $1-\exp(-1) = 0.63$ when $mt^2 = N$ and m and t are both large. A slightly more complex bound suggested by one of the reviewers predicts an efficiency of $3/4 = 0.75$ when $mt^2 = N$ and is a true lower bound.

Remark 2: A secure cryptosystem is a good pseudorandom number generator so modeling $f(*)$ as a random function makes intuitive sense. As will be seen from the proof, $f(*)$ need only be random so far as its cycle structure (i.e., the lengths of its cycles and associated "tails") is concerned. This is a much weaker condition.

Also, to a large extent, the random function assumption increases the expected effort and is therefore conservative. If $f(*)$ tended to have longer than average cycles, less overlap would occur. In the limit, if $f(*)$ had one cycle of length N then the starting and endpoints could be spaced $N^{1/2}$ apart and completely cover the key space with $M = T = N^{1/2}$, a significant improvement over the $N^{2/3}$ complexity under the random function assumption.

If $f(*)$ had the other extreme of degeneracy, $f(K) = K$ for all K, then cryptanalysis would be even more trivial. There are cycle structures which ruin the time–memory trade-off, but it is hard to see how one could obtain them with a secure cryptosystem.

As a check on the validity of (10), we ran a small test on the DES reduced to a 10-bit key ($N = 1024$) with $m = t = 10$. The lower bound predicts that $P(S) \geq 7.7$ percent, and with 20 different R functions we obtained a range of 6.8 percent to 9.1 percent, in excellent agreement with the bound. If there were no overlap at all, $P(S)$ would have been 9.8 percent.

Remark 3: Equation (10) indicates that $P(S)$ will be small for typical values of m and t. For example, if $m = t = N^{1/3}$ then $P(S) \doteq 1/(N^{1/3})$. This is overcome by generating $O(N^{1/3})$ different tables with different choices for R. If the first table does not produce a success, the second table is tried, etc. New choices of R are valuable because their cycle structures are independent of past tables, so a point repeated in two tables does not imply a repeated row.

Even if R is restricted to be a mapping which just chooses an ordered subset of 56 bits out of the ciphertext's 64 bits there are $(64!)/(8!) = 3 \times 10^{84}$ choices for R. If the cryptosystem is a good pseudorandom number generator even such minor changes in R will make the cycle structures of the associated $f(*)$ functions independent. This was done in the small DES simulation and $P(S) = 81.3$ percent overall coverage was obtained with 20 tables. If the coverage of each table was independent of the others, then 80.7 percent coverage was predicted from the individual $P(S)$. There was a slight positive bias because the 200 starting points for the 20 tables were taken to be the first 200 integers. This modification from random selection of the starting points reduces the expected search effort but is more difficult to analyze.

Proof of Theorem: The proof is closely related to the birthday problem [6, p. 33]. Letting A denote the subset of keys covered by the first t columns of Fig. 2 (i.e., not including the endpoints) we have

$$P(S) = E|A|/N, \quad (12)$$

where $|A|$ denotes the number of elements in A. Letting $I\{X\}$ denote the indicator function of the event X,

$$P(S) = E \sum_{i=1}^{m} \sum_{j=0}^{t-1} I\{X_{ij} \text{ is new}\}/N$$

$$= \sum_{i=1}^{m} \sum_{j=0}^{t-1} \Pr(X_{ij} \text{ is new})/N \quad (13)$$

where a point being "new" means it has not occurred in a previous row or thus far in its row. Using

$$\begin{aligned} &\Pr(X_{ij} \text{ is new}) \\ &\quad \geq \Pr(X_{i0}, X_{i1}, \cdots, X_{ij} \text{ are all new}) \\ &\quad = \Pr(X_{i0} \text{ is new}) \Pr(X_{i1} \text{ is new} | X_{i0} \text{ is new}) \cdots \\ &\qquad \Pr(X_{ij} \text{ is new} | X_{i0}, X_{i1}, \cdots, X_{i,j-1} \text{ are new}) \\ &\quad = \frac{N-|A_{i0}|}{N} \frac{N-|A_{i0}|-1}{N} \cdots \frac{N-|A_{i0}|-j}{N}, \end{aligned} \quad (14)$$

when A_{ij} denotes the set of elements covered thus far. Clearly each factor in (14) is larger than $(N-it)/N$ since there are at most t different elements in each row. Therefore

$$\Pr(X_{ij} \text{ is new}) \geq [(N-it)/N]^{j+1} \quad (15)$$

and

$$P(S) \geq (1/N) \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N-it)/N]^{j+1}.$$

completing the proof.

To obtain the $N^{2/3}$ complexity claimed earlier, set $m = t = N^{1/3}$ so that $P(S)$ is approximately $N^{-1/3}$ for a single table. Generate $N^{1/3}$ (or several times that number) of tables with different reduction mappings R. With high probability one of the tables will produce the correct answer, although occasionally a key will be chosen that is not included in the tables, and the method will fail to produce a solution.

Overall there are $M = N^{2/3}$ words of memory ($N^{1/3}$ tables, each with $m = N^{1/3}$ words), and the overall number of operations is also $T = N^{2/3}$ ($N^{1/3}$ operations per table). The different tables can be tried sequentially, with $N^{1/3}$ parallel processors, or anywhere in between.

We must still show that the false alarm rate is not so high as to dominate the computation.

Theorem: The expected number of false alarms per table tried, $E(F)$, is bounded by

$$E(F) \leq mt(t+1)/2N. \tag{16}$$

Remark: When a false alarm occurs, at most t operations are required to rule it out, which is comparable to the normal computation required for computing $Y_1, Y_2, \cdots, Y_t$. If $mt^2 = N$ and $\gg 1$, then the expected computation due to false alarms increases the expected computation by at most 50 percent.

Proof: Letting F_{ij} denote the occurrence of a false alarm due to $Y_j = EP_i$,

$$E(F) \leq \sum_{i=1}^{m} \sum_{j=1}^{t} \Pr(F_{ij}). \tag{17}$$

F_{ij} can occur in j different ways: due to $f(K)$ merging immediately with the ith row of the matrix, that is if $f(K) = f^{t-j+1}(\mathrm{SP}_i)$, or merging after one iteration, that is if $f(K)$ is not in the ith row of the matrix, but $f^2(K)$ equals $f^{t-j+2}(\mathrm{SP}_i)$; etc. Each of these j different ways of causing F_{ij} to occur has probability at most $1/N$ because, up to the merging, K, $f(K)$, etc. are independent random variables uniformly distributed over $\{1, 2, \cdots, N\}$. Therefore

$$\begin{aligned} E(F) &\leq \sum_{i=1}^{m} \sum_{j=1}^{t} j/N \\ &= mt(t+1)/2N, \end{aligned} \tag{18}$$

completing the proof.

III. Hardware Implementation

The preceding ideas establish the time–memory trade-off from a theoretical viewpoint but, because of the higher cost of memory, it is necessary to look at specific hardware implementations to determine to what extent, if any, the technique produces a cost savings over exhaustive search. This section therefore estimates the cost of a machine which breaks the DES using the time–memory trade-off. The machine uses off-the-shelf hardware costing approximately \$4 million and produces 100 solutions per day, with an average wait of one day between the time the problem is entered and the solution produced. The machine can also be used to effect the precomputation in approximately one year.

If the machine is used, fully loaded, for five years after the precomputation then the equivalent cost per solution is \$25. Other manufacturing and operating costs may increase this to \$100 per solution, but use of less expensive components and larger values of m may reduce the cost to as little as \$1 per solution. The geometric midpoint, \$10 per solution, is taken as an "order of magnitude" estimate.

When compared to the estimated \$5000 per solution cost [5] of exhaustive search, it is seen that the new technique is significantly cheaper. Further, the time–memory trade-off's cost could be reduced if custom large-scale integrated (LSI) circuitry is allowed as in [5]. It should be remembered, however, that the time–memory trade-off does not work in a known plaintext attack if block chaining or cipher feedback is used, whereas exhaustive search continues to be usable. Also the higher throughput of the time–memory machine (100 problems per day versus 2 problems per day in [5]) requires a larger number of problems to keep the machine fully loaded and realize its full cost advantage.

The DES has $N = 2^{56} = 7 \times 10^{16}$ keys. By rounding this to $N = 10^{17}$, we can neglect overlap in the matrices because (10) shows that approximately 80 percent of the points are distinct when $mt^2 = N$. Optimizing over m and t is not a simple matter because there is no simple objective function. The values $m = 10^5$ and $t = 10^6$ were selected after some trial and error as resulting in a reasonable machine cost, cost per solution, time to solution, and throughput. Using these values results in

$$P(S) = mt/N = 10^{-6}, \tag{19}$$

so approximately 10^6 tables are needed. Overall $M = 10^{11}$ words of memory are required, each 112-bits long (56 bits each for SP_i and EP_i), for a total memory requirement of 10^{13} bits. A \$20 magnetic tape can store on the order of 10^9 bits so 10 000 tapes are needed at a cost \$0.2M. To read these in one day requires 100 tape drives. (The data transfer rate is then approximately 100×10^6 bits per second, or 10^{13} bits per day. Assigning 100 tapes to each drive also means a tape is changed every 15 minutes per drive.) At a cost of \$20 000 per drive this adds \$2.0M to the system cost.

Each tape drive has a semiconductor memory consisting of 625 16-kbit random access memory (RAM) chips, or 10^7 bits per memory. This can store the $m = 10^5$ words needed for one table, at an approximate cost of \$4 000 per memory (\$6 per chip) or \$0.4M total cost.

Because there are $t = 10^6$ points in a row of Fig. 2 and $P(S) = 10^{-6}$, on the average it is necessary to compute 10^{12} values of $f(*)$ before achieving success. There are 100

tape drive/memory units working in parallel, and there are approximately 10^5 seconds in a day, so a DES unit must be able to implement the $f(*)$ function in 10^{-5}s = 10 μs to achieve a one-day solution time. Fairchild has announced a DES chip set which will implement the $f(*)$ function in approximately 5 μs (load a key and a plaintext, encrypt, and output the ciphertext). Initially it is selling for approximately $100 per unit but should reach $20 in quantity within a few years. Using the $100 figure to be conservative, each fast memory can have 100 DES units associated with it, each one working on a different problem. (Parallelism cannot speed up computation of successive iterations of $f(*)$.) This approximately equalizes the DES and other costs ($10 000 per drive for DES units) and does not overload the memory since there are more memory chips, and they can be accessed much faster than the DES units need data. There are some multiplexing and queueing problems, but these can be resolved easily because of the probabilistic nature of the search. If a small fraction of the memory accesses are delayed by queueing, it might be most cost effective to merely go on to the next iteration and forfeit that chance of success.

The total parts cost is $3.6M. If depreciated over five years this is approximately $2500 per day, or $25 per solution since the machine works on 100 problems in parallel. While other manufacturing and operating costs might increase this to $100 per solution, the use of a larger value of m should decrease cost per solution (but increase machine cost). This is because the larger table uses more memory chips, allowing a commensurate increase in the number of DES units and the number of problems being solved in parallel. Of course, the cryptanalyst must have enough problems to keep the machine fully loaded if he is to realize this cost savings.

It also should be possible to use less expensive components. For example, the $2.0 M for 100 tape drives might be replaced by $0.1 M for 100 video recorders used as inexpensive tape drives. The probabilistic nature of the computation allows us to tolerate occasional errors in the data, and the sequential nature of the accessed data eliminates the need for extremely rapid forward and reverse speeds.

The DES chip cost was conservative. If they can be obtained for $20 per unit, then 500 units can be interfaced to each tape drive/memory at no increase in projected cost, and 500 solutions would be produced each day. Memory costs are also falling rapidly. Taken together, these improvements indicate that the cost per solution might be as low as $1 in the near future.

The precomputation is equivalent to an exhaustive search of the keyspace because there are approximately t tables, each requiring mt encipherments, for a total of $mt^2 = N$ encipherments. A single Fairchild DES unit operating at 5 μs per encipherment would require 11 000 years for the precomputation, but the above described machine with 10 000 units could complete it in 1.1 years.

Because tape cost is a small part of the overall system cost, the added cost to store several times the average number of tables needed for a solution is unimportant, and the expected *computation* per solution is not increased if the tables are used cyclically. A new problem can be entered at any stage without affecting the average number of tables that it must try before achieving success.

For similar reasons, it is possible to produce different tapes for different "targets": using eight blanks as the chosen plaintext would be a good general choice, while "XYZ Corp" or "Login :" might work better for other targets. A special machine would be needed for doing ongoing precomputations, but it would consist primarily of 10 000 DES units and one tape drive so its parts cost would be approximately $1.0M and not have a large effect on system cost.

IV. Conclusion

The time–memory trade-off was described for use with a block cipher, but the same approach works with a synchronous stream cipher [7]. The first k bits of keystream are taken as the $f(K)$ function, where k is the number of bits of key. This can be done under a known plaintext attack.

The method works on all systems in a chosen plaintext attack [7] but does not work with a known plaintext attack on a cipher feedback system [7] if the initial load of the shift register is random and varies between conversations. Proposed Federal standards suggest this precaution.

Even a block cipher can foil the time–memory trade-off in a known plaintext attack through cipher block chaining [7], [8] or other techniques which introduce memory into the encipherment. Then, even when eight blanks occur in the plaintext, their encipherment depends on the preceding text. Even if the first block of text is fairly standard (e.g., "Login: "), this technique can be foiled by the transmission of a random "indicator" which is used to affect the encipherment (e.g., it is taken as the 0th plaintext block). Again, proposed standards include provision for cipher block chaining with a random indicator.

While this time–memory trade-off cryptanalytic technique can be easily foiled, it does work on the DES in basic block mode. More importantly, it indicates that even when cipher block chaining or other techniques are added, a larger key size is needed to have a reasonable assurance of security. While table lookup and exhaustive search are currently infeasible on systems with 64-bit or larger key sizes, an $N^{1/2}$ time–memory trade-off would push the minimum usable key size up to 128 bits. The $N^{2/3}$ technique described here, coupled with the large number of $N^{1/2}$ time–memory tradeoffs known for other searching problems, indicates that valuable data should not be entrusted to a device with smaller key size.

References

[1] R. C. Merkle and M. E. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 525–530, Sept. 1978.

[2] S. C. Pohlig and M. E. Hellman, "An improved algorithm for

computing logarithms over GF(p) and its cryptographic significance," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 106–110, Jan. 1978.
[3] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, Nov. 1976.
[4] D. E. Knuth, *The Art of Computer Programming, Vol. III: Sorting and Searching*. Reading, MA: Addison–Wesley, 1973.
[5] W. Diffie and M. E. Hellman, "Exhaustive cryptanalysis of the NBS data encryption standard," *Comput.*, vol. 10, pp. 74–84, June 1977.
[6] W. Feller, *An Introduction to Probability Theory and Its Applications*, 3rd ed. New York: Wiley, 1968.
[7] W. Diffie and M. E. Hellman, "Privacy and authentication: An introduction to cryptography," *Proc. IEEE*, vol. 67, pp. 397–427, Mar. 1979.
[8] H. Feistel, "Cryptography and computer privacy," *Sci. Amer.*, vol. 228, pp. 15–23, May 1973.

Group Codes for the Gaussian Broadcast Channel with Two Receivers

CHARLES P. DOWNEY AND JOHN K. KARLOF, MEMBER, IEEE

***Abstract*—The two-receiver Gaussian broadcast channel is a model of a communication system where a single codeword is transmitted over two distinct Gaussian channels that have different signal-to-noise ratios. The receiver with the better signal-to-noise ratio decodes all of the information carried by the codeword, while the other receiver decodes only some of the information. The concept of group codes for the Gaussian broadcast channel is developed and applied to permutation codes for the Gaussian broadcast channel. A group code for the Gaussian broadcast channel is a group code for the single-user Gaussian channel in which a subgroup of the generating matrix group is used to partition the codewords into disjoint subsets called clouds. Distinct clouds represent distinct messages for the noisier channel while the individual codewords represent distinct messages for the other channel. The clouds share some of the distance properties of group codes. Necessary and sufficient conditions are given for finding good group codes for the Gaussian broadcast channel (in terms of minimum distance).**

I. Introduction

THE TWO-RECEIVER Gaussian broadcast channel is a model of a communication system where a single codeword is transmitted over two distinct Gaussian channels and is received by two receivers. The receivers have no contact with each other, and the channels have different signal-to-noise ratios. The receiver with the better signal-to-noise ratio decodes all of the information carried by the codeword, while the other receiver decodes only some of the information.

In 1972, Cover [3] formally introduced the concept of broadcast channel coding theory and discussed the problem of finding the set of simultaneously achievable transmission rates for the two channels. Using random coding arguments, Bergmans [1] established the capacity region of the two-receiver Gaussian broadcast channel. Recently, Heegard, dePedro, and Wolf [4] used permutation codes to achieve rates and error probabilities better than time sharing. The purpose of this paper is to develop the concept of a group code for the two-receiver Gaussian broadcast channel.

A group code for the Gaussian broadcast channel is a group code for the single-user Gaussian channel in which a subgroup of the generating matrix group is used to partition the codewords into disjoint subsets called clouds. Distinct clouds represent distinct messages for the noisier channel, while the individual codewords represent distinct messages for the other channel. We prove that the clouds share some of the distance properties of group codes. For example, by defining the distance between two clouds to be the minimum distance between any two codewords (one from each cloud), we show that the set of distances between any one cloud and the rest is independent of the cloud considered. We also examine necessary and sufficient conditions for finding good group codes for the Gaussian broadcast channel (in terms of minimum distance).

In the next section we define the Gaussian broadcast channel and summarize some of the previous work in the area. Group codes for the single-user Gaussian channel are discussed in Section III, and in Section IV we introduce group codes for the Gaussian broadcast channel. In Section V we discuss permutation codes for the Gaussian broadcast channel and present some examples. The Appendix contains a brief summary of the group representation theory used in this paper.

II. Two-Receiver Gaussian Broadcast Channel

In Fig. 1, source 1 contains a library of equally likely messages $I=\{a_1,a_2,\cdots,a_{n_1}\}$, one of which needs to be sent every T seconds to both receivers. Source 2 contains a library of equally likely messages $J=\{b_1,b_2,\cdots,b_{n_2}\}$, one of which needs to be sent every T seconds to only receiver 1. Channel 1 has the higher signal-to-noise ratio. The encoder has a set $X=\{x_1,\cdots,x_M\}$ of $M=n_1n_2$ codewords and a one-to-one encoding function $I\times J\rightarrow X$.

Manuscript received April 25, 1979.
The authors are with the Mathematics/Computer Science Department, University of Nebraska at Omaha, Omaha, NE 68182.

0018-9448/80/0700-0406$00.75