Cryptography

by D. Coppersmith

This paper is concerned with two aspects of cryptography in which the author has been working. One is the Data Encryption Standard (DES), developed at IBM and now in wide use for commercial cryptographic applications. This is a "private key" system; the communicants share a secret key, and the eavesdropper will succeed if he can guess this key among its quadrillions of possibilities. The other is the Diffie-Hellman key exchange protocol, a typical "public key" cryptographic system. Its security is based on the difficulty of taking "discrete logarithms" (reversing the process of exponentiation in a finite field). We describe the system and some analytic attacks against it.

Introduction

I have been asked to give "timeless and immortal" comments on the subject of "cryptography." Instead I will give a somewhat technical presentation of the niche within cryptography where I have done some of my work. I will concentrate on two subjects: the Data Encryption Standard (DES) and the discrete logarithm problem (Diffie-Hellman key exchange). The opinions in this paper are purely personal and are not intended to represent IBM policy.

Cryptography is the art of secret writing, or devising ways of transmitting messages so that others cannot read them. Cryptanalysis is the art of breaking such cryptographic schemes, thus reading the heretofore secret messages. I believe that some solid experience in the latter is a prerequisite to the successful practice of the former art; one cannot devise strong systems without some idea of the manner of attacks to which such systems are likely to be subject. This paper will include aspects of each field.

DES

The Data Encryption Standard, or DES, was developed at IBM during the period 1973–74. I consider this to be a crowning achievement of the Yorktown Mathematical Sciences Department, although our department certainly did not act alone. Among the people involved in this project at IBM in Yorktown were Horst Feistel (the designer of the original "Lucifer" scheme), Alan Konheim, Bryant Tuckerman, Edna Grossman, and myself (all in the Mathematical Sciences Department), as well as Bill Notz and Lynn Smith, who did much of the implementation. IBM Kingston was well represented, and the Crypto Competency Center there now "owns" this product. We were assisted by several outside consultants, and benefited greatly from the expertise of the National Security Agency.

The basic structure of DES was inherited from the earlier "Lucifer" machine [1]. There is a secret 56-bit key, under whose influence a 64-bit cleartext (input) is transformed into a 64-bit ciphertext (output). The input message is broken into two halves, "left" and "right." During the first of sixteen "rounds," the 32-bit right half, along with 48 of the 56 key bits, is fed into a nonlinear function F. The 32-bit output of this function, added to the left half message, becomes the new right half message. In the meantime, the old right half message is funneled forward to become the new left half message. Thus ends one round. The process is repeated sixteen times, using a different selection of 48 key bits each time. The final left half and right half messages become the ciphertext.

With this design, decipherment is easily performed, as long as one possesses the same key. One simply climbs back up the ladder, reversing the effects of one round at a time. One sees the new left half message and the new right half, as well as the key. The new left is the same as the old right; that, and the appropriate 48 key bits, are fed into F. The 32-bit output of F is subtracted from the new right half to obtain the old left half. Thus can one reverse one round, and by repeated application, one reverses the entire encryption.

DES is described more fully in the books by Meyer and Matyas [2] and Konheim [3].

REPRINTED FROM IBM JOURNAL OF RESEARCH AND DEVELOPMENT, VOL. 31, NO. 2, 1987; ©1987, 2000

0018-8646 / 00 / $5.00 © 2000 IBM

I was a student at the time of the development, working summers, but I am proud of my small part in this project. This cryptographic engine, small enough to fit on a chip in 1974, has withstood all analytic attacks since then. Many papers have been published, focusing on apparent anomalies in the design and attempting to create analytic attacks against the system. But to my knowledge nobody has ever come up with a shortcut that would make cryptanalysis easier than key exhaustion.

The quickest known way is to try all 2^{56} possible keys to see which keys encipher a given plaintext into a known ciphertext. Now $2^{56} = 7.2 \times 10^{16}$ is 72 quadrillion. That's a lot of operations. With advancing technology and parallel architectures, there is a perception that 2^{56} encryptions is a feasible task. I do not agree with that perception yet, but the viewpoint is being loudly touted and widely believed. For those skeptics, we also have Plan B: triple encryption, which is in use in several installations. Choose three independent keys, encipher your plaintext under the first key, re-encipher the ciphertext under the second key, and once again under the third key. (Some prefer the sequence "encrypt, decrypt, encrypt," which allows compatibility with single-key systems.) This costs three times as much, but the security gain is tremendous: The apparent work factor is now $2^{112} = (2^{56})^2 = 5.2 \times 10^{33}$. I believe there is a word for that number—"decillions"—but I don't believe there is a machine for it, or ever will be. [The apparent work factor is $(2^{56})^2$ rather than $(2^{56})^3$ because of a "meet in the middle" attack.]

Key passing

DES gives us a way of communicating secret information across a public channel: I encrypt the data under a secret key (or three secret keys), and send you the ciphertext; you possess the same secret key(s) and can decipher, recovering the plaintext. But how did you and I agree on the same secret keys? If we had a way of passing secret keys, why did we not use the same path for passing our secret messages and do away with DES altogether?

One possible answer to this question is given by Diffie and Hellman [4]. They propose a "public key" scheme for secret passing, based on exponentiation in a finite field. I will describe the later variant due to Shamir, Rivest, and Adleman [5], which is a little more flexible.

- *Logarithms in GF(p)*

Suppose I wish to tell you a secret, but you and I have never previously shared any secret which we could use as a cryptographic key. We agree on a large prime p. This can be done in public. I wish to send you the secret y. I represent y as an element of the finite field $GF(p)$ (the integers mod p). I also select two integers, c and c^{-1}, such that $cc^{-1} \equiv 1 \bmod p-1$, and keep these integers secret. You select two secret integers d and d^{-1} such that $dd^{-1} \equiv 1 \bmod p-1$, and keep these secret as well. By repeated squaring, I compute the field element y^c, that is $y^c \bmod p$, and send that field element to you. You raise this to the d power in the field, and send me the resulting element y^{cd}. I raise this to the c^{-1} power, obtaining y^d, which I send to you. You raise that to the d^{-1} power and obtain the secret y.

The eavesdropper knows the field $GF(p)$, and three elements: y^c, y^{cd}, y^d. Suppose that he knows how to extract the integer d from the field elements y^c and $y^{cd} = (y^c)^d$; i.e., he can tell what power y^c must be raised to in order to obtain y^{cd}. This is the finite field analog of taking logarithms, and is today called the "discrete logarithm" problem. An eavesdropper with this power could break the system: From y^c and y^{cd} he could recover d, and this, along with y^d, would give him y. (There may be other ways of breaking the system; it has not been proven that it is as hard as discrete logarithms.) This gives a cryptanalytic motivation to the study of discrete logarithms.

> Question No. 1: "What's the point? I know how to take logarithms of real numbers; can it be that much harder in finite fields?"
>
> Answer No. 1: "Yes: it's an entirely different problem. In the reals, we can use power series expansions to find successive approximations to the real logarithm, and if we get twenty-digit accuracy, that's enough for all practical purposes. But in the finite field case, there are no 'approximations'; a miss is as good as a mile. Further, there is no way even to tell whether you are getting 'close,' because there is no such thing as 'close' here."

To fix notation, suppose that b is a *generator* of $GF(p)$, meaning that powers of b take on each nonzero residue mod p. Let $y = \log_b x \in Z/(p-1)$ be an integer y such that $b^y = x \in GF(p)$.

An early method for solving this problem, attributed to Roland Silver [6], is the "giant steps, baby steps" technique. It runs in time $\sqrt{p}$, or more precisely, $\sqrt{q}$, where q is the largest prime dividing $p-1$. The technique is similar to a Vernier caliper: you make two lists of size $\sqrt{p}$ each, and where the two lists have a common element, you compute your logarithm from that intersection. Set $k = [\sqrt{p}] + 1$, and make two lists: xb^i, $i = 0, 1, \cdots, k-1$, and b^{jk}, $j = 0, 1, \cdots, k-1$. Look for a common element, and compute $xb^i = b^{jk}$, so that $x = b^{(jk-i)}$, and $\log_b x = jk - i$.

More efficient techniques are given by Western and Miller [7] and Adleman [8]. Their algorithms are based on *smoothness*, the same concept that pervades many of the modern integer factorization algorithms. One selects a "smoothness bound"

$$B = e^{\beta\sqrt{\log p \log\log p}}$$

for a small constant β. Say that an integer x is "smooth"

(with respect to B) if it is expressible as the product of small primes $q_i < B$:

$$x = \prod_{q_i < B} q_i^{\alpha_i}.$$

The intermediate goal is to find $\log_b q_i$ for each of the small primes $q_i < B$. To this end, we will develop a system of linear equations relating these logarithms:

$$\sum \alpha_i \log_b q_i = \sum \beta_i \log_b q_i \text{ in } Z/(p-1).$$

Exponentiating such a linear equation, we would find

$$\prod q_i^{\alpha_i} = \prod q_i^{\beta_i} \bmod p.$$

Thus our goal is to relate smooth numbers multiplicatively mod p.

Adleman's method of doing this involves selecting random integers z and computing $b^z \bmod p$, thus creating a random integer between 1 and $p-1$. If this integer happens to be *smooth*, then its expression as the product of small primes gives us one of the equations we need:

$$b^z \equiv \prod q_i^{\alpha_i} \bmod p,$$

$$z = \sum \alpha_i \log_b q_i \text{ in } Z/(p-1).$$

If the integer is not smooth, we get nothing. Thus the amount of trial and error depends on the proportion of smooth integers in this range.

Having gathered many such equations, we solve the system of linear equations, to obtain the logarithms of all the small primes. [The equations are in a finite ring $Z/(p-1)$, but this poses no problems.] Now we are in a position to find individual logarithms. To find $x = \log_b c$, we first randomize: Select a random integer z, compute $y \equiv cb^z \bmod p$, and ask whether y is smooth. If it is, its expression as the product of small primes gives us our answer:

$$y \equiv cb^z \equiv \prod q_i^{\gamma_i} \bmod p,$$

$$\log_b c + z = \sum \gamma_i \log_b q_i$$

The running time is dominated by the search for smooth numbers, and this governs our choice of B. If B is too large, we will have to gather too many equations (one for each small prime) before we can solve the system. If B is too small, smooth numbers will be too scarce and we will have too much trial and error. The selection of B as $e^{\beta\sqrt{\log p \log\log p}}$ gives an optimal running time of $e^{\gamma\sqrt{\log p \log\log p}}$. Variants on this algorithm have improved the constant γ [9].

• *Logarithms in fields of characteristic two*

The key exchange protocol has since been extended to finite fields of the form $GF(2^n)$. The protocol itself is unchanged, except that arithmetic is carried on in the field $GF(2^n)$. The circuitry for carrying out this arithmetic seems to be easier than for fields $GF(p)$ of comparable size. However, my 1983 work [10] shows that the cryptanalysis is also easier in $GF(2^n)$, so that the trade-off between security and ease of implementation is not clear-cut. I will describe this work next.

$GF(2^n)$ can be represented as the set of polynomials $A(x)$ with coefficients in $GF(2)$ (the field of integers mod 2), reduced modulo a fixed irreducible polynomial $P(x)$ of degree n. We assume that $P(x)$ is in fact primitive, meaning that powers of x in the field, $x^i \bmod P(x)$, take on all the nonzero values $A(x) \bmod P(x)$. The logarithm is defined as before: $y = \log_x A(x)$ if $x^y \equiv A(x) \bmod P(x)$. The logarithm y lives in the ring $Z/(2^n - 1)$.

It is handy to bear in mind a rough correspondence between notions in $GF(2^n)$ and the corresponding notions in $GF(p)$.

$GF(p)$	$GF(2^n)$
integer	polynomial
prime	irreducible polynomial
small prime	irreducible polynomial of small degree

Using such a correspondence, Hellman and Reyneri [11] adapted the Adleman algorithm to find logarithms in $GF(2^n)$. Choose a smoothness bound $b = c\sqrt{n \log n}$. Say that $A(x)$ is smooth if it is the product of irreducible polynomials $q_i(x)$ of degree at most b. Select a random z, compute $A(x) \equiv x^z \bmod P(x)$, and ask whether $A(x)$ is smooth. If so, its explicit representation as the product of small irreducible polynomials gives a linear equation relating the logarithms of these small irreducible polynomials:

$$A(x) \equiv x^z \equiv \prod q_i(x)^{\alpha_i} \bmod P(x),$$

$$z = \sum \alpha_i \log_x q_i(x).$$

Collect enough such equations, solve the system, and obtain the logarithms of all the small irreducible polynomials. Finally, to find the logarithm of a given element, multiply by a random power of x, reduce mod $P(x)$, and hope that the result is smooth. If so, the expression in terms of small irreducible polynomials gives the desired logarithm. The running time of their algorithm is $e^{\gamma\sqrt{n \log n}}$; recalling that n is comparable to $\log p$ for fields of comparable size, we see that this running time is comparable to that of Adleman for the case of $GF(p)$.

One can do better than this, however. To motivate my work, consider: The smaller a polynomial (in terms of degree), the more likely it is to be smooth. Each multiplicative relation [mod $P(x)$] between smooth polynomials gives us one of the desired equations relating the logarithms of the small irreducible polynomials. Therefore, we want to find a source of multiplicative relations [mod $P(x)$] among rather small polynomials.

VOL. 31, NO. 2, 1987, REPRINT

We take advantage of the following peculiarity of fields of characteristic two: *Squaring is a linear operation.* That is, $(A + B)^2 = A^2 + B^2$, since the usual $2AB$ term drops out ($2 = 0$ here).

Consider the example of $GF(2^{127})$, where we actually applied the techniques I will describe. Let the field be defined by $P(x) = x^{127} + x + 1$. (It turns out that we are free to choose the defining polynomial at our convenience, as long as it is irreducible and of the right degree.)

Select two arbitrary polynomials $A(x)$, $B(x)$ of degree at most 10. Construct $C(x) = x^{32}A(x) + B(x)$. Set $D(x) \equiv C(x)^4 \bmod P(x)$. We find

$$D(x) \equiv C(x)^4 = x^{128}A(x)^4 + B(x)^4$$
$$\equiv (x^2 + x)A(x)^4 + B(x)^4 \bmod P(x),$$

where the second equality follows from the linearity of squaring, and the third follows from the fact that here $x^{128} \equiv x^2 + x \bmod P(x)$. Notice that both $C(x)$ and $D(x)$ are of relatively small degree: at most 42. Thus there is a good chance that both $C(x)$ and $D(x)$ are smooth; if so, the equation $D(x) \equiv C(x)^4 \bmod P(x)$, together with the explicit representations of $C(x)$ and $D(x)$ as the products of small irreducible polynomials, yields a linear equation relating the logarithms of the small irreducible polynomials:

$$\prod q_i(x)^{\beta_i} = D(x) \equiv C(x)^4 = (\prod q_i(x)^{\alpha_i})^4 \bmod P(x),$$
$$\sum \beta_i \log_x q_i(x) = \sum 4\alpha_i \log_x q_i(x).$$

Because of this source of multiplicative relations among smooth polynomials [mod $P(x)$], we are faced with less trial and error. This in fact gives us a running time of

$$e^{cn^{1/3}(\log n)^{2/3}},$$

compared with the previous time of

$$e^{c'n^{1/2}(\log n)^{1/2}}.$$

> Skeptical remark No. 2: "So, you've decreased 1/2 to 1/3. Who cares?"
>
> Answer No. 2: "Ah, but it's in the second exponential! A little change makes a big difference there."
>
> Skeptical remark No. 3: "These are all asymptotic estimates. Given the speaker's work on matrix multiplication [12] (where the asymptotically fast results only work for matrices larger than the size of the known universe), how seriously should we take these estimates?"
>
> Answer No. 3: "In the field $GF(2^{127})$, a moderately sized field, we did in a few minutes what the competing schemes were taking days to do. Our advantage will increase for larger fields."

We compare $GF(2^n)$ with $GF(p)$. My work shows that the cryptanalysis in $GF(2^n)$ is roughly exponential in the cube root of the number of bits, compared to roughly exponential in the square root of the number of bits for $GF(p)$. Thus, for a comparable level of security, the $GF(2^n)$ case must use a much larger key size. But since the arithmetic is so much easier to implement in circuitry for $GF(2^n)$ than for $GF(p)$, it remains a viable alternative. I had hoped to have killed the scheme with this work, but that does not seem to be the case.

- *Elliptic curves*

The key-passing scheme has been described in two flavors of finite field: $GF(p)$ and $GF(2^n)$. But in fact it never makes use of *addition* in the finite field, only *multiplication.* The scheme will generalize to any finite Abelian group.

At Crypto '85, Victor Miller [13] described an adaptation of the Diffie-Hellman scheme to a finite elliptic curve. This has the appearance of being much harder to break, although not as many people have been trying to break it, so its apparent strength remains to be fully tested.

Select a large prime ℓ, and integers A, B. An elliptic curve E_ℓ is given as the set of solutions of a given cubic equation, together with an extra point at infinity:

$$E_\ell = \{(x, y) \mid x, y \in Z/\ell, y^2 \equiv x^3 + Ax + B \bmod \ell\} \cup \{\infty\}.$$

The group operation is defined by a geometric construction. If P and Q are elements of E_ℓ, draw the line joining P and Q in (x, y)-space. The line will intersect the curve in one other point. Reflect this point about the x-axis, and the reflection will be called the sum $(P + Q)$. There are also rules to handle the special cases when the line has only two points of intersection (one being a "tangent point") or when the line is vertical (necessitating the extra point at infinity). It can be shown that the group operation so defined is associative and commutative, so that we really have a finite Abelian group. Its size $|E_\ell|$ is between $\ell + 1 - 2\sqrt{\ell}$ and $\ell + 1 + 2\sqrt{\ell}$, and achieves any integer in that range.

We know how to add two points P and Q on the curve E_ℓ. By repeated duplication, we can take an integer multiple of a point: If $P \in E_\ell$, $n \in Z$, we can form $nP = P + P + \cdots + P$. (Repeated duplication means $12P = 6P + 6P$.) But the reverse problem (corresponding to the "logarithm") appears to be difficult: Given P, $Q \in E_\ell$, find an integer n such that $Q = nP$. In fact it appears to be much more difficult than the corresponding problem in $GF(p)$.

The key-passing scheme is by now familiar. We agree on ℓ, A, B, and thus E_ℓ. We find its size $\ell' = |E_\ell|$. (This in itself seems to be fairly difficult, but not insurmountable.) I wish to pass you the secret y, which I represent as a point $P \in E_\ell$. I select auxiliary secret integers c, c^{-1} with $cc^{-1} \equiv 1 \bmod \ell'$. You select d, d^{-1}. I pass you cP, you pass me cdP, I pass you dP, and you compute P. The eavesdropper can crack the scheme if he can find "logarithms" as described above: Given P, $Q \in E_\ell$, find $n \in Z$ such that $Q = nP$.

This logarithm problem seems to be difficult. The smoothness techniques do not seem to work any more. Before, we were taking an element of $GF(p)$ and lifting it to Z, that is, taking an integer between 1 and $p-1$ and asking whether that integer is the product of small primes in Z. Trying an analogous procedure here, we find that the rational points on the *rational* elliptic curve

$$E_Q = \{(x, y) \mid x, y \in Q, y^2 \equiv x^3 + Ax + B\} \cup \{\infty\}$$

are too large to be of much help: To lift a general point from E_ℓ to E_Q we would take too long just writing down the numerators and denominators of the coefficients.

The only technique which is known to work in this case is the "giant steps, baby steps" method of Roland Silver, which takes time $\sqrt{q}$, where q is the largest prime dividing $|E_\ell|$. (His method will work for any finite Abelian group, in fact.) By judicious choice of ℓ, A, B, we can force q to be as large as ℓ.

Put another way, for the same apparent level of security, one can allow the prime ℓ here to be much smaller than the prime p in the original Diffie-Hellman scheme. Rough estimates suggest that we can deal with integers of 128 bits rather than 512. This means (first) that the modular multiplications involved in group operations are much easier (being on smaller integers), and (second) that there are fewer group operations to perform, since the number of group operations per exponentiation is roughly $\log_2 \ell = 128$ rather than $\log_2 p = 512$. The group operations themselves are more complicated: They seem to require nine multiplications rather than one. But the net result is that, for the same *apparent* level of security, the elliptic curve method can be faster by a factor of 7:1. Thus this method is promising, and bears a closer examination.

I summarize the best known running time of the discrete logarithm problem for the three cases examined:

$GF(p)$ $\quad e^{c_1(\log p)^{1/2}(\log\log p)^{1/2}}$,

$GF(2^n)$ $\quad e^{c_2(n)^{1/3}(\log n)^{2/3}}$,

E_ℓ $\quad e^{c_3(\log \ell)^1} = \sqrt{\ell}\,(c_3 = 1/2)$.

All are exponential in some fractional power of number of bits, and this fraction in the second exponent is the important measure: 1/2 for $GF(p)$, 1/3 for $GF(2^n)$, and 1 for the elliptic curves. Thus elliptic curves are apparently much harder than $GF(p)$, which is in turn harder than $GF(2^n)$.

Summary

I have attempted to give some flavor of the techniques involved in the discrete logarithm problem, which is one instance of cryptanalysis. I have also attempted to communicate some of the excitement I feel in working with cryptographic problems, and some of the pride I feel for having been associated with DES, especially the latter. DES has been much maligned in the popular press, and I welcome this opportunity to defend it publicly. I view it as one of the fine contributions of a fine department, now happily celebrating its twenty-fifth anniversary.

References

1. H. Feistel, "Cryptography and Computer Privacy," *Sci. Amer.* **228,** 15–23 (May 1973).
2. C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, John Wiley, New York, 1982.
3. A. G. Konheim, *Cryptography: A Primer*, John Wiley, New York, 1981.
4. W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Info. Theory* **IT-22,** 644–654 (1976).
5. A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental Poker," *MIT/LCS/TM-125*, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, February 1979.
6. S. C. Pohlig and M. Hellman, "An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance," *IEEE Trans. Info. Theory* **IT-24,** 106–110 (1978).
7. A. E. Western and J. C. P. Miller, *Tables of Indices and Primitive Roots*, Royal Society Mathematical Tables, Vol. 9, Cambridge University Press, England, 1968.
8. L. M. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," *Proceedings, 20th IEEE Found. Comp. Sci. Symp.*, 1979, pp. 55–60.
9. D. Coppersmith, A. M. Odlyzko, and R. Schroeppel, "Discrete Logarithms in $GF(p)$," *Algorithmica* **1,** 1–15 (1986).
10. D. Coppersmith, "Fast Evaluation of Logarithms in Fields of Characteristic Two," *IEEE Trans. Info. Theory* **IT-30,** 587–594 (1984).
11. M. E. Hellman and J. M. Reyneri, "Fast Computation of Discrete Logarithms in $GF(q)$," *Advances in Cryptology: Proceedings of CRYPTO '82*, D. Chaum, R. Rivest, and A. Sherman, Eds., Plenum Publishing Co., New York, 1983, pp. 3–13.
12. D. Coppersmith and S. Winograd, "On the Asymptotic Complexity of Matrix Multiplication," *SIAM J. Comput.* **11,** No. 3, 472–492 (1982).
13. V. S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology: Proceedings of Crypto '85*, Hugh C. Williams, Ed., *Springer-Verlag Lecture Notes in Computer Science*, Vol. 218, 1986, pp. 417–426.

Received August 28, 1986; accepted for publication November 5, 1986

Don Coppersmith *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Coppersmith received the B.S. in mathematics from the Massachusetts Institute of Technology, Cambridge, in 1972 and the M.S. and Ph.D. in mathematics from Harvard University, Cambridge, Massachusetts, in 1975 and 1977. He was the winner of the Putnam Mathematical Contest in 1968, 1969, 1970, and 1971. He is currently manager of the Theory of Computation group at the Thomas J. Watson Research Center, where he has been a Research Staff Member since 1977. His research interests include cryptography, computational complexity, combinatorics, and coding theory.