Euromath Bulletin
Vol. 2, No. 1, 1996

Securing the Net — the Fruits of Incompetence

Arjen K. Lenstra
Room MCC−1C317B, Bellcore, 445 South Street
Morristown, NJ 07960−6438
U. S. A.
lenstra@bellcore.com

Abstract

This note reviews the most popular mathematical primitives that are used in current attempts to build secure networks.

1 Introduction

Corporations worldwide suddenly regard the Internet, once the almost exclusive playground of the global community of computer nerds, as an immense business opportunity. Until recently phrases such as "consult our homepage at `http://www.digicrime.com`" made sense to only a few; now they belong to our everyday vocabulary. This is all part of the explosive growth of what has been called the *Global Information Infrastructure*. It is a development that cannot be stopped and probably should be welcomed.

Nevertheless, corporations are beginning to see that venturing out on the Internet may expose them to enormous risks. The purpose of "putting your homepage on the web" is to increase visibility and to draw attention. Unfortunately the audience includes not only potential customers but also virtually *all* hackers worldwide. At least some of them will, intentionally or not, cause trouble.

Solutions to the resulting security problems are not hard to find on the net, since many software vendors now advertise "secure" versions of their products. This makes using the net really risky, because users might mistakenly believe they are well protected. The widely publishized and rather frequent news stories about network break–ins and imperfections in security software should dispel such illusions. It seems that our competence to secure the net cannot keep up with our desire to use it.

Despite the confusing array of security solutions, there are only a few mathematical primitives on which they are based. Even in faulty security products, the soundness of the underlying mathematics is hardly ever in question; it is the way it is used that causes the vulnerabilities. In this note I discuss the mathematical primitives—not the many slippery ways in which they are employed. I concentrate on the primitives themselves and the assumption of their soundness and will show that this is one of the most important reasons that *computational number theory* has become so fashionable, even at industrial labs. This popularity, based entirely on our incompetence at efficiently solving a few basic number theoretic problems, is hardly something of which to be proud. Purists who object to the vulgarization of number theory should find comfort in the prospect that as soon as efficient solutions have been found, number theory will again be the immaculately impractical Queen of Mathematics—its status before "security applications" came along.

This note is written for an undergraduate mathematics audience that is not familiar with the mathematical notions involved in many popular security products. In Section 2 I sketch a possible security application on the Internet. Of course, much more is involved in practice than I am able to mention here, but I show some of the basic concepts and set the stage for the mathematical primitives that are presented in Sections 3 and 4. No attempts are made to formalize notions such as "infeasible", "hard" or "efficient". For further background refer to [2] (Section 2) and [1] (Sections 3 and 4).

2 Background

The following naive scenario, though grossly oversimplified, shows some of the key issues of communication security. Suppose that two parties who have never met want to exchange confidential information over some untrusted but reliable network. "Untrusted" here means that all messages are accessible to eavesdroppers; "reliable" means that no bits are dropped or changed. The Internet is a reasonable example of such a network.

If the two parties share a random string s of secret bits that is as long as the message m, then the problem can easily be solved: send the *bitwise exclusive or* $s \oplus m$ of s and m. Since s is random, $s \oplus m$ leaks no information. Furthermore, it is easy to derive m from $s \oplus m$ because $m = s \oplus (s \oplus m)$. This would solve the problem, except that we cannot assume that any two parties have a secret string of random bits in common. A further disadvantage is that, using this simple approach, each s can be used only once (since $s \oplus m_1$ and $s \oplus m_2$ reveal information about $m_1 \oplus m_2 = (s \oplus m_1) \oplus (s \oplus m_2)$).

The latter problem can be overcome by using, for instance, DES—the U.S. government's Data Encryption Standard. DES is an example of a *block cipher*. It can be used to construct a function f such that it is sufficiently hard to derive m from $f(s,m)$ (for any number of messages m of any length), such that $m = f(s, f(s,m))$, and such that f can very efficiently be computed. Here s is a 56−bit string, or a 168−bit string if higher security is needed (triple−DES); this string is referred to as the *key*. Thus, if both parties had a common key s that was unknown to any other party, any message m could be encrypted as $f(s,m)$, sent to the other party using an untrusted channel, and decrypted as $m = f(s, f(s,m))$. This can be repeated back and forth, for any reasonable number of messages.

A description of DES is beyond the scope of this note; it does a lot of seemingly arbitrary bit−fiddling that aims to, among other things, *confuse and diffuse* the bits of the key s and the message m. There are many other ciphers that can be used to construct functions that have properties similar to f. For our purposes the problem that remains to be considered is how the two parties perform the *key exchange* for a relatively short key (of, say, 56 bits), in such a way that the key that is exchanged remains hidden to an eavesdropper.

Note that the simple approach where the (trusted) provider of the communication services assigns a unique random key to each pair of possible parties is not feasible: each party would need an enormous data base of keys, which would not only be hard to keep updated (for new subscribers) but would also have to be safeguarded very carefully. An elegant solution to the problem of key exchange is given in Section 4. It only requires a small amount of public information that is accessible to the entire network. While using it, however, all parties involved need to sign all their messages.

This requires *digital signatures* to convince each of the communicating parties that the messages they receive come from the party they intend to communicate with. This can be realized using *public key cryptography*, as explained in the next sections. In public key cryptography all parties have a *secret key* and a corresponding *public key*. In signature applications the secret key is used by its owner to generate a signature; the corresponding public key can be used by anyone to check the validity of the signature.

Thus, all secret keys are kept hidden by their owners, but all parties have access to each other's public keys, just as telephone numbers are (mostly) public information. Alternatively, parties that wish to communicate can exchange their public keys first; this in turn leads to the problem of authenticating public keys and related issues, which can also all be solved using public key cryptography (and which may, in certain circumstances, require a trusted third party). In practice many other problems have to be addressed as well. The purpose of our simpleminded example is only to introduce the basic principles as a background for the mathematical primitives that are presented below.

A cryptographic technique that is often used for digital signatures in conjunction with public key cryptography is *hashing*. A hash function h, when applied to a message m of arbitrary length, results in a fixed length hash $h(m)$ of m; for MD5 ('Message Digest') the resulting length is 128 bits, for SHS (the U.S. government's 'Secure Hash Standard') it is160 bits. For a good hash function it should be infeasible to compute an m such that $h(m)$ is equal to any prescribed value. Also, it should be infeasible to find different m_1 and m_2 such that $h(m_1)=h(m_2)$. Like DES, the currently popular hash functions are based on very efficient seemingly random bit manipulations, and not on clear-cut mathematical ideas as most public key cryptosystems are (even though those ideas might turn out to be incorrect). It is an open problem how to design a very efficient hash function that is provably as hard to break as one of the public key cryptosystems described below. It is also becoming an urgent problem: on May 2, 1996, Hans Dobbertin of the German Information Security Agency, who was responsible for breaking MD4 in 1995, announced a new cryptanalysis of MD5 that 'might be reason enough to substitute MD5 in future applications'. The life expectancy of SHS is uncertain, since its design is very similar to that of MD5.

3 Factoring

Factoring a composite integer n means finding integers p and q, both >1, such that $n=p\cdot q$. This is, in its generality, believed to be a hard problem, even though there is no firm mathematical ground on which this assumption can be based: the only evidence is our failure to find an efficient factoring method.

The supposed difficulty of factoring is crucial for the security of the RSA public key cryptosystem, which was invented in 1977 by Rivest, Shamir, and Adleman. Each user of RSA has its own modulus $n=p\cdot q$, where p and q are two large primes, and two integers e and d such that $e\cdot d \equiv 1 \bmod (p-1)(q-1)$. The values n and e are made public as that user's public key, but d is kept secret (along with p and q) and is the user's secret key. Since large primes can efficiently be generated, and because d can quickly be found using the extended Euclidean algorithm given e, p, and q (for properly chosen e), such n, e, and d can easily be found for each user of RSA.

To send a message $m\in\mathbf{Z}/n\mathbf{Z}$ to the owner of public key (n,e), one computes $E(m)=m^e$ and transmits the encrypted message $E(m)$. The owner of (n,e), upon receipt of $E(m)$, retrieves the unencrypted message m by computing $E(m)^d=m$ (cf. Fermat's little theorem). Both the encryption and the decryption can efficiently be done using the 'repeated square and multiply' method. This is, however, considerably slower than, for instance, DES. RSA can also be used to sign a message m as $S(m)=m^d$; the validity of the signature can be checked by verifying that $S(m)^e=m$.

It has not been proved that factoring n is necessary to break RSA (i.e., to decrypt $E(m)$ knowing n, e, and $E(m)$ but without knowing the proper d), but it is obviously sufficient. There exist several variations of RSA that replace the multiplicative group $(\mathbf{Z}/n\mathbf{Z})^*$ by some other group that depends on a composite modulus, and in which the operations are carried out. None of them, however, seems to have any substantial advantages over RSA (despite claims to the contrary), and all of them can be broken by factoring the modulus.

Since the invention of RSA considerable progress has been made towards more efficient factoring methods. Trial division and Pollard's rho method (1975) find the smallest prime factor p of n in exponential time: $O(p)$ and $O(\sqrt{p})$, respectively. The elliptic curve method (ecm, 1985) can be expected, on loose heuristic grounds, to take time $(\log n)^2 \exp((1+o(1))\sqrt{2\log p \log\log p})$, for $p \to \infty$. Note that this is subexponential in p. These are examples of *special purpose* factoring methods, since their run time depends on the factors to be found. The largest factor found by any of these methods (ecm) so far has 47 decimal digits. None of them is a threat to RSA, if the primes in the modulus have at least, say, 75 decimal digits.

The run time of *general purpose* factoring methods depends solely on the size of the number to be factored. The most important ones are the Continued Fraction method (CFRAC, 1970), Quadratic Sieve (QS, 1981), and the Number Field Sieve (NFS, 1989). The first two have heuristic expected run time $\exp((1+o(1))\sqrt{\log n \log\log n})$ for $n \to \infty$, though QS is in practice much to be preferred to CFRAC. They share this run time with many other factoring algorithms, among others the worst case $p \approx \sqrt{n}$ of ecm. NFS was the first algorithm to substantially improve upon this time, with heuristic expected run time $\exp((c+o(1))(\log n)^{1/3}(\log\log n)^{2/3})$ for $n \to \infty$ and c some constant between ≈ 1.53 (for 'nice' numbers like $2^{2^9}+1$) and ≈ 1.92 (for general numbers). NFS is currently considered to be more efficient than QS for numbers that have more than, say, 110 decimal digits.

The largest number factored with QS, in 1994, is the famous 129–digit 'RSA–challenge' that appeared in the August 1977 issue of Scientific American. Rivest estimated, in 1977, that this factorization would require 40 quadrillion years. With QS it took 8 months, using the idle cycles of computers worldwide. The total run time of this factoring effort has been estimated as 5000 mips years, (i.e., 5000 years on a VAX 780). The largest number factored with NFS, in 1996, is a 130–digit RSA–modulus, with total run time estimated as 550 mips years. Using this figure and the asymptotic run time of NFS (omitting the $o(1)$ for convenience), one can get an impression of the effort required to factor 512–bit (155–digit) RSA moduli, and conclude that such moduli are on the verge of being breakable. With a moderate amount of progress in factoring, 768–bit keys (a size that is becoming more popular lately) could become vulnerable as well. A polynomial–time factoring algorithm would most likely render RSA useless.

4 Discrete logarithms

The most common discrete logarithm problem is the following. Given some prime p, a generator g of $(\mathbf{Z}/p\mathbf{Z})^*$, and $y \in (\mathbf{Z}/p\mathbf{Z})^*$, find $x \in \{0, 1, \ldots, p-1\}$ such that $g^x = y$. Like factoring, this is in its generality believed to be a hard problem, and, again like factoring, the only evidence that it is hard is that we have not yet been able to solve it efficiently.

The supposed difficulty of discrete logarithms is the basis for the security of the Diffie–Hellman key exchange protocol (1976). A prime p and generator g are publicly known. Party A picks a random $a \in \{0, 1, \ldots, p-1\}$, computes $g^a \in (\mathbf{Z}/p\mathbf{Z})^*$ and sends it to party B. Party B picks a random $b \in \{0, 1, \ldots, p-1\}$, computes $g^b \in (\mathbf{Z}/p\mathbf{Z})^*$ and sends it to A. Both parties can now compute the common key $g^{ab} = g^{ba} \in (\mathbf{Z}/p\mathbf{Z})^*$. As mentioned earlier, this key exchange protocol should be used with care, since otherwise it is susceptible to a *man in the middle attack*.

It has not been proved that computing discrete logarithms is necessary to break the Diffie–Hellman protocol (i.e., to compute g^{ab} given p, g, g^a, and g^b), but it is obviously sufficient. Many other public key cryptosystems have been proposed that can be broken if discrete logarithms can be computed efficiently.

However, unlike factoring based systems which are more or less equally hard to break, here the situation is a bit more complicated.

It does not seem to be possible to reduce factoring to discrete logarithms, or vice versa. Nevertheless, there is a strong similarity between the solution methods for the two problems: with a few exceptions (such as ecm), the ideas behind most factoring algorithms can be used to solve discrete logarithms as well. Examples are linear search for x and Pollard rho, which find x in time $O(x)$ and $O(\sqrt{x})$ operations in $(\mathbf{Z}/p\mathbf{Z})^*$, respectively. The 'Gaussian integers' method finds x in time $\exp((1+o(1))\sqrt{\log p \log\log p})$, for $p\to\infty$, and is based on ideas similar to the ones that led to QS. And then there is a Number Field Sieve based method that finds x in time $\approx\exp((1.92+o(1))(\log p)^{1/3}(\log\log p)^{2/3})$ for $p\to\infty$. Although a 'live' cryptosystem using a 60−digit prime was broken in 1991, practical experience with discrete logarithm algorithms is limited. Efforts are underway to change this situation.

An important distinction between the exponential and subexponential time discrete logarithm methods is that in the former the group $(\mathbf{Z}/p\mathbf{Z})^*$ can be replaced by any group, but in the latter, arithmetic properties of the set $\{1,2,\ldots,p-1\}$ (which is used to represent $(\mathbf{Z}/p\mathbf{Z})^*$) are crucial. This has several interesting consequences, of which I mention a few.

If g generates only a subgroup of order $q<p-1$ of $(\mathbf{Z}/p\mathbf{Z})^*$, and $y\in\ <g>$, then x can still be found in $O(\sqrt{x})\leq O(\sqrt{q})$ operations in $(\mathbf{Z}/p\mathbf{Z})^*$ (using Pollard's rho method, or using Shanks' 'baby−step−giant−step' method), or in time subexponential in p using any of the subexponential methods. But a method that runs in time subexponential in q is not known. If g generates the group of points of some elliptic curve modulo p, then x can again be found in $O(\sqrt{x})$ operations in the elliptic curve group. But no method is known that runs in time subexponential in the order of g or even in p. If, on the other hand, g generates $(\mathbf{F}_{p^m})^*$ or a subgroup thereof, for some fixed integer $m>1$, then x can be found in time subexponential in p^m. The latter is a consequence of the fact that the relevant arithmetic properties of the set of polynomials modulo p of degree $<m$ is similar to those of the set $\{1,2,\ldots,p^m-1\}$.

This apparent lack of discrete logarithm algorithms that run in time subexponential in the order of a subgroup or of an entirely different group that cannot be represented in the way the subexponential algorithms require, is exploited in the design of several public key systems. In DSS (the U.S. government's Digital Signature Standard) an order q subgroup of $(\mathbf{Z}/p\mathbf{Z})^*$ is used to improve the speed of the cryptosystem, while at the same time reducing the size of the resulting signatures, without, hopefully, compromising the security: breaking it requires time either $O(\sqrt{q})$ or $\approx\exp((1.92+o(1))(\log p)^{1/3}(\log\log p)^{2/3})$, both of which are supposedly infeasible for the specified sizes $\lceil\log_2 q\rceil=159$ and $\lceil\log_2 p\rceil\geq 511$ (the Russian variant of DSS requires $\lceil\log_2 q\rceil=239$). Elliptic curve based cryptosystems achieve the same objectives simply by choosing p small, but large enough to make $O(\sqrt{p})$ attacks infeasible.

Since $\lceil\log_2 q\rceil$ is fixed at 159 and the increase of processor speed has not leveled off yet, it is conceivable that a moderate amount of progress in exponential time discrete logarithm algorithms could make DSS vulnerable within the foreseeable future. Also, some specialists find it too early to give up hope for better than exponential time attacks on elliptic curve based systems.

5 Conclusion

We have seen that the factoring and discrete logarithm problems are remarkably similar: both are easy to formulate, believed to be hard purely based on our lack of success solving them efficiently, suitable for the design of public key cryptosystems, and, most remarkably, they seem to be susceptible to very similar solution methods. The last point is quite worrisome: even though not all factoring methods can be turned into discrete logarithm methods (ecm is the most notable exception), most can. Thus, it is conceivable that a newly invented factoring method that wipes out all factoring based cryptosystems, would have the same effect on discrete logarithm based cryptosystems. Obviously, there is a strong need for diversification in the design of public key cryptosystems.

These issues, though crucial for the design of secure networks, are actually the least of our current worries. It is not unlikely that instead of enjoying the fruits of our number theoretic incompetence we will soon be harvesting the rotten fruits of our incompetence at properly implementing or use the much needed security measures. As soon as we leave the realm of mathematics, security considerations become much more confused and complicated: human factors, compatibility issues, trust management, key management, key escrow, export restrictions, to mention only a few, are crucial issues that have an enormous potential to be exploited by a worldwide army of hackers that cannot necessarily distinguish a prime from a composite. This is not to say that security on the net cannot be achieved, but the subject requires study of much more than the mathematical issues alone. using

References

[1] A. K. Lenstra and H. W. Lenstra, Jr. *Handbook of theoretical computer science. Volume A, Algorithms and complexity.* (J. van Leeuwen, ed.). Algorithms in number theory. Chapter 12. Elsevier, Amsterdam, 1990.

[2] D. R. Stinson. *Cryptography, theory and practice.* CRC Press. Boca Raton. London, Tokyo. 1995.