Best Kept Secrets: Elliptic Curves and Modern Cryptosystems

Thomas Coffee
MIT 18.704 Fall 2004

Introduction

In just a few short decades, we have seen the most seemingly obscure branches of mathematics seized upon as vital foundations for modern commercial and social activity in the burgeoning information age. Realms of algebra and number theory previously the exclusive province of professional mathematicians are now the basis of patents, industries, and dedicated government agencies. Cryptography, the science of secrets, has suddenly caught up with the frontiers of academic theory, and is driving the development of mathematical machinery previously explored in the context of apparently esoteric results like Fermat's Last Theorem. Valuable and often widespread practices now rest upon other difficult and unsolved problems that remain.

Somewhat unexpectedly, the theory of elliptic curves has emerged as a key player in the cryptographic landscape of the modern world. In this paper, we will show how the rich algebraic structures built upon these curves underlie the state of the art in modern cryptosystems.

Foundations of Cryptography

Communication channels are vulnerable to eavesdropping. Information in our universe has a tendency to diffuse and dissipate, and enormous resources are required to counteract this tendency. Encrypting the information itself renders physical security unnecessary: eavesdropping is useless if one cannot understand the information on the channel.

This function is performed by means of a *cryptosystem*, shown in (1). Alice has a class of meaningful messages $\mathcal{P}$ that she might wish to send to Bob; these are called *plaintext* messages. To protect these from eavesdropping during transmission, she uses a mapping f to encode these as corresponding messages from a different class C; these are called *ciphertext* messages. Bob must be able to reconstruct the plaintext from the ciphertext, applying the inverse mapping f^{-1}. We wish to prevent anyone else from applying f^{-1} and thereby recovering the plaintext.

$$\text{Alice } \mathcal{P} \xrightarrow{f} C \xrightarrow{f^{-1}} \mathcal{P} \text{ Bob} \tag{1}$$

The mappings f and f^{-1} may be generally defined by a *structure* along with specifying *parameters*. The structure is the component of the cryptosystem that is difficult to change (it may be implemented in permanent devices or conventions). We assume that the structure is universally known: otherwise its revelation would permanently compromise the system. The parameters

may be easily altered in order to achieve many distinct potential mappings within the structure; hence, the system may depend upon secret parameters without catastrophic risk.

The Computational Complexity Gap

An eavesdropper Eddy will generally attempt to determine f^{-1} so that he can read all messages on the channel. Eddy may have a variety of information available to him. He will presumably have information about certain regularities among the potential messages in $\mathcal{P}$ (for example, word frequencies in English), and about the structure of f and f^{-1}. Since the channel is vulnerable, we assume he will have some arbitrary samples of ciphertext messages. He may also have information about the corresponding plaintext for some samples. With greater influence or access, he may be able to select the plaintext or ciphertext for some of these samples at his own discretion. (If he has access to an implementation of f, he will be able to generate unlimited samples on chosen plaintext.)

A good cryptosystem must be difficult to break, yet not overly burdensome to use. Given the information available to Eddy, it will take him a certain amount of computational effort to determine f^{-1} with some high probability. Clearly, Alice and Bob wish to make this effort very large. However, the same structural changes that drive the effort required for Eddy to determine f^{-1} can also drive the computational effort required for Alice and Bob to use their system for communications.

Hence, cryptosystem design is driven by maximizing the *gap* between the computational requirements of eavesdroppers and users. In algorithmic terms, this means the cryptosystem must allow Alice and Bob to perform their operations with significantly lower asymptotic complexity (with respect to the parameters) than that required for Eddy to crack the system. The necessary magnitude of the gap is driven by the value of information to all parties, that is, the investment of computational resources that can each be expected of Alice, Bob, and Eddy. It is also driven by the cost of computation: if all parties' computational resources for a given investment increase uniformly, the feasible scale of parameters will increase, and a system with a given complexity gap will become more secure. The dramatic cost reductions in computer hardware of the past few decades have multiplied the value of the complexity gap enormously.[1]

The Trouble with Secret Keys

Traditional cryptography assumes that the forward mapping f and the inverse mapping f^{-1} can be easily derived from one another (that is, with low asymptotic complexity in the parameters). Hence both Alice and Bob must know f and f^{-1}, but no one else can know either. This is known as *symmetric* or *secret-key* cryptography.

Secret-key cryptography has one important disadvantage: in order to agree upon their secret key f, Alice and Bob must establish a physically secure communication channel. As we have discussed, the costs of doing this can often be large, depending upon the degree of protection desired. The number of such secure channels required to set up pairwise private communica-

tions among a group of n people grows as $O(n^2)$; to include each individual on the planet as of this writing would require $\sim 10^{19}$ such exchanges.

The problem is actually even worse, because the secret keys have limited lifespan. The complexity gap is finite, and increases in available computational power drive parameter size rapidly upward. A set of parameters providing ~1-second encryption and decryption time for Alice and Bob while forcing ~1-year cracking time for Eddy in the year 2000 may allow ~1-second cracking time for Eddy by the year 2025 for the same investment. Implementing a system with parameters robust until 2025 may cost Alice and Bob ~10-minute encryption and decryption back in 2000, unacceptable for operational use. Hence keys must be replaced on a regular basis, depending upon the time value of information and developments in computing hardware. Note that a larger complexity gap is the only hope for longer security horizons given fixed investments among all parties.

A New Class of Cryptosystems

In 1976, Diffie and Hellman introduced an alternative type of cryptosystem in which f^{-1} *cannot* be easily determined from f. In this scheme, Bob can generate f and f^{-1} on his own, then make f freely available without revealing f^{-1} to anyone. No physically secure channels are required, since the mapping f is made public. In addition, Bob's unique knowledge of the inverse mapping f^{-1} allows him to authenticate his own messages, which Alice and others can verify using f. This approach is known as *asymmetric* or *public-key* cryptography. For obvious reasons, f is called the *public key* and f^{-1} is called the *private key*.[2]

The enormous practical advantages of public-key cryptography come at two key costs. First, to enable new senders to contact him without secure exchange, Bob must completely reveal the mapping f: this gives Eddy all the information he could possibly desire short of the inverse mapping itself, with the exception of chosen-ciphertext samples. Second, to enable new recipients to easily enter the arena, people like Bob must be able to generate public/private key pairs with only modest effort: that is, *generating* as well as *operating* the system must be substantially easier than cracking the system. The result is intense pressure on both these forms of the complexity gap.[3]

A function that is difficult to invert is called a *one-way function*, and many types are well known. For public-key cryptography, however, we need something somewhat more specialized. First, we need a one-way function that Bob can invert easily given an additional (secret) piece of information, known as a *trapdoor one-way function*. In other words, the inverse mapping must be a hard problem to which Bob has the solution, or something computationally close to it. Second, Bob must be able to generate the problem and its solution quickly, without making the problem easy to solve.

Hard Problems with Simple Holes

Given the steep requirements of public-key cryptosystems, we begin to see how we might wind up dealing with some of the most sophisticated mathematical problems yet known. Even after we have found a candidate trapdoor one-way function—a nontrivial task in itself—we must be able to provide assurances about the computational simplicity of generating key pairs, and about the computational difficulty of inverting the public key. The latter problem is so challenging, in fact, that none of the public-key cryptosystems widely used today are provably secure: they have simply resisted attack for long enough to inspire confidence. Here we introduce the two general techniques underlying these systems.[4]

Integer Factorization

The most well-known public-key algorithm was first published by Rivest, Shamir, and Adleman, hence is known as RSA. It is based on the supposed difficulty of performing general integer factorization. The algorithm runs as follows:

(1) Find two large primes p and q and let $n = p \cdot q$. Then $\phi(n) == (p-1)(q-1)$.

(2) Find an integer $e < n$ such that $\gcd(e, \phi(n)) == 1$.

(3) Find an integer d such that $e \cdot d \equiv 1 \pmod{\phi(n)}$.

(4) Define $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ by $f(x) = x^e \pmod{n}$, then $f^{-1}(x) == x^d \pmod{n}$.

We first verify the soundness of the algorithm. Fermat's Little Theorem implies that $x^{k \cdot \phi(n)} \equiv 1 \pmod{p}$ and $\pmod{q}$ for any integer k, hence likewise $\pmod{n}$.[5] The condition in (2) ensures that (3) is possible. Given (3), we have $f^{-1}(f(x)) \equiv x^{d \cdot e} \equiv x^{1+k\,\phi(n)} \equiv x \pmod{n}$.

The public key is determined by n and e, the private key by n and d. Hence the security of the algorithm rests on the difficulty of determining d from n and e alone.[6] Unlike Eddy, Bob knows the factorization of n, which he uses to find $\phi(n)$ and then d via (3). The complexity gap is largely determined by the driving algorithms of each task:

- Generating the system: discovering unpublished primes
- Operating the system: computing modular exponents
- Cracking the system: factoring large integers

We note briefly that the modular exponent $x^d \pmod{n}$ can in general be accomplished in $O(\log d \log^2 n)$ time by writing the binary representation of d and summing appropriate terms from the successive squares of x modulo n. With d satisfying (3), this bound can be simplified to $O(\log^3 n)$. The other two problems are discussed in the sections below, where we illustrate a progression to the current state of the art.

Discovering Unpublished Primes

The simplest primality test for large integers n is exhaustive: divide n by all integers $\leq n$, requiring time $O(\log n \log n!)$. Noting that each factor $d \geq \sqrt{n}$ has a corresponding factor $n/d \leq \sqrt{n}$, we can confine ourselves to the latter subset of candidate divisors, providing a slight improvement. With more memory, exhaustive division can be somewhat improved using Eratosthenes' prime sieve, accumulating successive primes as the candidates for subsequent division, reducing the complexity to $O(n \log \log n)$. In fact, checking against a known list of small primes proves efficient for general integer factorization. However, these methods are entirely inadequate for integers with large prime factors, being superexponential in the number of digits of n.

We can take advantage of some special properties of primes to construct simple negative tests for primality, which can be turned into probabilistic positive tests. To begin with, Fermat's Little Theorem implies that $n > 1$ is composite whenever $a^{n-1} \not\equiv 1 \pmod{n}$ for an integer a prime to n. We can also easily show that if n satisfies this condition for at least one such integer a, it will satisfy this condition for at least half the possible values of a, which suggests a probabilistic algorithm based on random values of a.[7] On the other hand, there exist composite n that satisfy $a^{n-1} \equiv 1 \pmod{n}$ for *all* such a, known as *Carmichael numbers* or *pseudoprimes*. In 1994, Alford et al. showed that there are infinitely many psuedoprimes,[8] so we cannot hope to rule them out with a checklist. Pomerance has previously established bounds on the density of pseudoprimes,[9] but as it turns out we can avoid the issue completely with a stronger test.

In 1980, Miller and Rabin constructed a stronger probabilistic primality test[10] making use of the following related theorem, which we will not prove here: if n is an odd prime and $n - 1 = 2^s t$ with t odd, then for each $b \in \mathbb{Z}_n$, we have either $b^t \equiv 1 \pmod{n}$ or $\exists\ r, 0 \leq r < s : b^{2^r t} \equiv -1 \pmod{n}$. In this case, it can be shown that at most $1/4$ of possible values of b will satisfy this condition if n is odd and composite. Hence, trying k randomly chosen values of b will give us a probabilistic bound of $1/4^k$ on the primality of n. This test can be carried out in $O(k \log^3 n)$ time. More interesting still, the unproven Generalized Riemann Hypothesis (GRH) guarantees the converse of the test for some value of b less than $2 \log^2 n$. Hence postulating the GRH provides a deterministic algorithm with complexity $O(\log^5 n)$.

Probabilistic algorithms can pose a problem for authentication, since they leave the cryptosystem vulnerable to deliberate selection of weak parameters, which can be used post hoc to repudiate the security of the private key. Fortunately, while awaiting proof of the GRH in 2002, Agrawal et al. constructed a surprisingly simple fast deterministic algorithm for primality testing.[11] The key criterion is a generalization of Fermat's Theorem to polynomials, and holds conversely: n is prime *if and only if* $(x - a)^n \equiv (x^n - a) \pmod{n}$ for a prime to n. In order to make the algorithm feasible, this congruence is reduced modulo the polynomial $(x^r - 1)$ for a suitably chosen r. The details become complicated, but the result can ultimately be obtained in $O(\log^{12+\epsilon} n)$ time, recently improved to $O(\log^{6+\epsilon} n)$ by Lenstra and Pomerance.[12] This is nearly as fast as the conditional strong pseudoprime test of Miller and Rabin.

Factoring Large Integers

The naïve methods we mentioned earlier for primality testing are equally applicable to integer factorization, but again they are wildly suboptimal. Our logical journey toward today's best factoring algorithms begins again with Fermat, who proposed a technique uniquely suitable for cracking RSA cryptosystems. Fermat observed that whenever $n == p \cdot q$ and $p \approx q$, n will be equal to a difference of squares with one small term, that is, $n == \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 == r^2 - s^2 == (r+s)(r-s)$ where s is small. Moreover, we can find this factorization quickly by computing $r^2 - n$ beginning from $r = \lceil \sqrt{n} \rceil$ and stepping upwards until we find a perfect square. The method is fairly limited by the assumption $p \approx q$, but can be generalized.

Instead of requiring $r^2 - s^2 == n$, we can relax the constraint to $r^2 - s^2 \equiv 0 \pmod{n}$ where $r \not\equiv s \pmod{n}$. Whereas the former produces immediate factors $(r+s)$ and $(r-s)$, the latter can be used to quickly obtain nontrivial factors $\gcd(r+s, n)$ and $\gcd(r-s, n)$ via the Euclidean algorithm. We now seek a method of generating solutions to the congruence equation. This can be done by manipulating the "parity" of prime factors: define a *factor base* B to be a set of distinct elements $p_i \in \text{Primes} \bigcup \{-1\}$, and define a B-number to be any number $\prod {p_j}^{\alpha_j}$ formed by a product of elements of B. For convenience, let $x \bmod n$ denote the residue of $x \in \left(-\frac{n}{2}, \frac{n}{2}\right]$. Now suppose we have a set of B-numbers $a_i = {b_i}^2 \bmod n == \prod {p_j}^{\alpha_{ij}}$ that are residues of squares, and suppose the total of all powers of each p_i occurring in the a_i is even, that is, $\sum_i \alpha_{ij} \equiv 0 \pmod{2}$ for each j. Then clearly $\prod_i a_i$ is a square, which is congruent modulo n to the square of $\prod_i b_i \bmod n$. Of course, we may be unlucky and find that the roots of these squares are congruent modulo n, in which case we must find another set a_i.

The efficiency of this approach relies on a few different procedures. To generate values b_i that produce likely B-numbers, we can let B consist of small primes and choose the b_i to produce a_i of small magnitude. The latter can be done efficiently using a continued fraction method developed by Legendre and deployed by Morrison and Brillhart.[13] To guarantee efficiency, we also need a bound on the density of integers divisible by primes below a certain threshold, which can be developed from Stirling's approximation and the Prime Number Theorem. Ultimately, the complexity of this approach can be estimated by $O\left(\exp\left[C\sqrt{\log n \log\log n}\right]\right)$ for a constant C.[14] Until recently, the best known factoring algorithm was an improvement of this approach by Pomerance called the "quadratic sieve" that reduced C from (roughly) $\sqrt{2}$ to $1+\epsilon$. Finally, in 1993 Lenstra and Lenstra generalized the strategy further to develop the "number field sieve," with a running time of $O\left(\exp\left[C(\log n)^{1/3}(\log\log n)^{2/3}\right]\right)$, which currently holds the lead in integer factorization.

As we can see, the best known factoring algorithms, while subexponential, have not yet achieved polynomial running time in the digits of n, unlike the algorithms required for generating and operating the RSA cryptosystem. We now compare the other major family of techniques for public-key cryptography.

Discrete Logarithms

Diffie and Hellman proposed a distinct, though similar, approach aimed at secret key exchange. Here we describe the ElGamel cryptosystem, a simple extension of their approach. The algorithm is based on the supposed difficulty of the discrete logarithm problem, and runs as follows:

(1) Find an integer q that is a prime or a power of a prime.

(2) Find a generator g of the finite field $\mathbb{F}_q$.

(3) Let $a(q)$ return a random nonzero element of $\mathbb{F}_q$ and fix $b = a(q)$.

(4) Define $f : \mathbb{F}_q \longrightarrow \mathbb{F}_q^2$ by $f(x) = \left(g^a,\, x \cdot g^{a \cdot b}\right)$ with $a = a(q)$, then $f^{-1}(c,\, x) == x \Big/ c^b$.

The soundness of the algorithm is fairly obvious. The condition in (1) implies that $\mathbb{F}_q$ is a finite field, which must have a multiplicative generator g, though this is not strictly necessary.[15] Bob computes $x \cdot g^{a \cdot b} \Big/ g^{a \cdot b} == x$.

The public key is determined by q, g, and g^b, the private key by b. Hence the security of the algorithm rests on the difficulty of determining b from g and g^b alone. Alice need only know g^b to compute $g^{a \cdot b}$ for her randomly chosen a, but Eddy requires b in order to compute $g^{a \cdot b}$ from g^a. We will again look at the driving algorithms for each major task:

- Generating the system: discovering primes or prime powers; finding generators of $\mathbb{F}_q$
- Operating the system: computing modular exponents
- Cracking the system: finding discrete logarithms

We note that the problems of discovering primes and computing modular exponents have already been addressed. Rubin and Silverberg propose an efficient method for finding large prime powers (if desired), making use of primality testing, on which we will not elaborate.[16] Wang has demonstrated a deterministic polynomial-time algorithm for finding generators of $\mathbb{F}_q$ provided the Extended Riemann Hypothesis (ERH) is true;[17] otherwise, fast probabilistic algorithms exist for finding generators.[18] We will focus here on the discrete logarithm problem.

Computing Discrete Logarithms

An algorithm by Silver, Pohlig, and Hellman breaks down the discrete logarithm problem substantially when the prime factors of $q - 1$ are small.[19] However, these cases are easily avoided, so we will not pursue this result further. Instead we will turn to the *index calculus* algorithm for discrete logs, which bears some strong parallels to the factor base approach to integer factorization.[20] We assume that $q == p^n$ for a prime p, and α is a multiplicative generator of $\mathbb{F}_q$. For a given $y \in \mathbb{F}_q$, we wish to find $x \,(\mathrm{mod}\, q - 1)$ such that $y == g^x$.

Consider the polynomial ring on $\mathbb{F}_p$, and note that $\mathbb{F}_q$ is isomorphic to $\mathbb{F}_p[X] / f(X)$ for any polynomial f of degree n. Since $g \in \mathbb{F}_q$, it can be written as a polynomial $g(X) \in \mathbb{F}_p[X]$ of degree $\leq n - 1$. Since $g^{(q-1)/(p-1)}$ is a generator of $\mathbb{F}_p$, solving the discrete logarithm problem with this base in $\mathbb{F}_p$ will solve our original problem; we will approach this by constructing a table of these discrete logs.

Analogous to the factor base algorithm, we select a "basis" of polynomials $B \subset \mathbb{F}_q$. Again, there is a subtle balancing act involved in sizing this set appropriately, which we will not explore fully. To compute the discrete logs of all $a(X) \in B$, we use the following procedure: choose a random integer t satisfying $1 \le t < q-1$, and let $c(X) = g(X)^t \bmod f(X)$. Now we determine whether $c(X)$ is in the span of B, that is, whether $c(X) == c_0 \prod_{a \in B} a(X)^{\alpha_{c,a}}$. If so, we can take discrete logarithms of both sides to obtain $\log_{g(X)} c(X) - \log_{g(X)} c_0 \equiv \sum_{a \in B} \alpha_{c,a} \log_{g(X)} a(X)$. We know $\log_{g(X)} c(X) == t$, and we assume we know the discrete logs of constants. Thus we have a linear equation in $\mathbb{Z}_{q-1}$ with unknowns $\log_{g(X)} a(X)$ for $a(X) \in B$. Trying different values of t, we assemble enough independent equations of this form to solve mod $q-1$, allowing us to compute the discrete logs of the elements of B, and hence any discrete log of interest.

Now we carry out a similar search to find t satisfying $1 \le t < q-1$ such that $y_1(X) = y(X)\, g(X)^t \pmod{f(X)}$ is of the form $y_0 \prod_{a \in B} a(X)^{\alpha_a}$. When this happens, we can compute $\log_{g(X)} y_1(X) == \log_{g(X)} y_0 + \sum_{a \in B} \alpha_a \log_{g(X)} a(X)$, and subsequently $\log_{g(X)} y(X) == \log_{g(X)} y_1(X) - t$, which is the discrete log we were looking for. The complexity of this algorithm (in q) is comparable to that of integer factorization, hence both approaches remain of considerable interest for cryptographic applications.

Elliptic Curve Cryptosystems

We assume the reader is somewhat familiar with the algebra of elliptic curves.[21] Elliptic curves may be defined over fields of characteristic greater than 3 in the canonical form $\left\{(x,\, y) \in \overline{\mathbb{F}}_q \times \overline{\mathbb{F}}_q \;\; y^2 == x^3 + ax + b\right\} \bigcup O$, where $\overline{\mathbb{F}}_q$ is the algebraic closure of $\mathbb{F}_q$, $a, b \in \mathbb{F}_q$, $4a^3 + 27b^2 \neq 0$, and O is the point at infinity. Over fields of characteristic 2, there are two types of curves, one characterized by the canonical equation $y^2 + cy == x^3 + ax + b$, $c \neq 0$, and the other by $y^2 + xy == x^3 + ax^2 + b$. The algebraic formulas that describe the group law on points of elliptic curves in $\mathbb{R}$ and $\mathbb{Q}$ extend naturally to finite fields.

The multiplicative groups formed by elliptic curves on finite fields may be used in place of the traditional groups on $\mathbb{Z}_n$ or $\mathbb{F}_q$ used in the RSA and ElGamal cryptosystems and their variants. It turns out that making this substitution in the RSA approach is inconsequential: Eddy's best approach is still integer factorization. However, in the case of ElGamal, this modification destroys the index calculus attack, leaving Eddy only weaker methods for breaking the system.

Here is the elliptic curve analog of the ElGamal system:

(1) Find an integer q that is a prime or a power of a prime.

(2) Find a generator $g \in E(\mathbb{F}_q)$ of the point group (or a point of large order).

(3) Let $a(q)$ return a random nonzero element of $\mathbb{F}_q$ and fix $b = a(q)$.

(4) Define $f : \overline{\mathbb{F}}_q^2 \longrightarrow \overline{\mathbb{F}}_q^4$ by $f(x) = (a \cdot g,\, x + a \cdot b \cdot g)$ with $a = a(q)$, then $f^{-1}(c,\, x) == x - b \cdot a \cdot g == x - a \cdot b \cdot g$.

The group law for points on elliptic curves has replaced integer multiplication; all operations are otherwise identical. Moreover, the computing power required to generate and operate the system is comparable to the traditional approach.

It is worth noting a few known attacks to which elliptic curve cryptosystems are vulnerable. Such systems are still vulnerable to the general Pollard ρ-method, a "Monte Carlo" approach using an iterated self-mapping on a multiplicative group,[22] but this approach is superexponential in the bit length of q. More specific to elliptic curves, Semaev and others found an isomorphism between $E(\mathbb{F}_p)$ and the *additive* group $\mathbb{F}_p$ whenever $|E(\mathbb{F}_p)| == p$, providing a polynomial-time algorithm for this class of curves.[23] In a similar vein, Menezes et al. found a way to embed $E(\mathbb{F}_p)$ in the multiplicative group on $\mathbb{F}_{q^k}$ for some integer k whenever $n \mid q^k - 1$, providing a subexponential-time algorithm. Both these classes can be avoided by appropriate application of Hasse's Theorem, and in the vast majority of cases by choosing a curve at random.

Thus far no analogs to the index calculus algorithm have been developed for elliptic curve cryptosystems. Given the rapid evolution of the field and many recent surprises, it is far from clear that this situation will persist. However, it is arguable that elliptic curves offer a much wider selection of groups and thus may be far more successful at evading general attacks.

To get an idea of the advantages of elliptic curve systems given modern hardware and algorithms, we can report that as of 2000, RSA systems with keys of 1024 bits were roughly matched with elliptic curve systems with keys of 160 bits.[24] Comparisons with RSA will tend to shift due to the different drivers behind these two systems; with respect to discrete logarithm systems, elliptic curve systems can be expected to maintain a consistent lead, barring any theoretical breakthroughs. Specialized embedded hardware optimizations of elliptic curve cryptography can provide some advantages even beyond those calculated from key size reduction.

The computational edge provided by algorithms like elliptic curve cryptography can have unexpected impact. For example, secure authentication and communication software is now being widely deployed in micro-scale devices, wherein power consumption for computing becomes an enormous cost driver. As a result, we may expect the interest and energy devoted to the theoretical problems of number theory to grow for some time to come.

Notes

[1] In fact, current computing power has pushed parameter sizes for widely used public-key systems high enough that we examine only asymptotic complexity in this paper. For smaller values of parameters, multipliers missing from Big-O estimates can have significant practical impact.

[2] The terms "public key" and "private key" sometimes refer to other, smaller pieces of information from which the mappings f and f^{-1} can be easily computed. In this usage, both the public key and the private key may be involved in computing f^{-1}.

[3] As a result, public-key methods tend to require more resources on the part of Alice and Bob than do secret-key methods. Where performance is an issue, public-key methods are often used purely for exchanging

secret keys, which are then used for encryption and decryption of messages. This is known as a *digital envelope*, and is conveniently extensible to multi-party communications.

[4] We neglect an important class of public-key cryptosystems based on the so-called "knapsack problem" that are not widely used. The original versions developed by Merkle and Hellman were broken in 1984 (Shamir) and 1985 (Brickell). Chor and Rivest introduced revised versions in 1984 and 1988. These were cracked for some parameter classes in 1995 (Shnorr and Hörner), but their approach still appears viable.

[5] Recall Fermat's Little Theorem: $a^{p-1} \equiv 1 \pmod{p}$ for p prime and $p \nmid a$. This follows from the fact that multiplication by a is a permutation of $\mathbb{Z}_p$ for $p \nmid a$, hence $a^{p-1}(p-1)! == \prod a \cdot \mathbb{Z}_p \equiv \prod \mathbb{Z}_p == (p-1)! \pmod{p}$, which gives the theorem since $p \nmid (p-1)!$. Since $k \cdot \phi(n)$ is a multiple of $(p-1)$ and $(q-1)$, taking exponents gives us the desired result. Congruence $(\mathrm{mod}\, n)$ follows from $\gcd(p, q) == 1$. Note that if $\gcd(x, n) == 1$, the result follows directly from Euler's generalization of the Little Theorem.

[6] An RSA cryptosystem can be cracked without determining the private key if one develops a feasible method for computing modular roots: that is, finding x given $f(x) == x^e \pmod{n}$. It is not known whether this problem is equivalent to integer factorization, but substantially more progress has been made on factoring.

[7] To show this, note that if n satisfies $a^{n-1} \equiv 1 \pmod{n}$ for $a = a_1, a_2$, it must do so for $a = a_1 a_2^{-1}$. Suppose we have $b \in \mathbb{Z}_n$ that does not satisfy the relation. Then for any a_i that does, we know that $b \cdot a_i$ does not, otherwise the relation would hold for $b \cdot a_i \cdot a_i^{-1} == b$. This accounts for at least half the possible values of a.

[8] See Alford WR, Granville A, Pomerance C. 1994. There are infinitely many Carmichael numbers. Ann. of Math. 140: 703-722.

[9] See Pomerance C. 1981. On the distribution of pseudoprimes. Math. Comp. 37: 587-593.

[10] See Rabin MO. 1980. Probabilistic algorithms for testing primality. J. Number Theory 12: 128-138.

[11] See Agrawal M, Kayal N, Saxena N. 2002. PRIMES is in P. Preprint. http://www.cse.iitk.ac.in/primality.pdf.

[12] See Lenstra HW, Pomerance C. 2003. Primality Testing with Gaussian Periods. Manuscript.

[13] See Morrison MA, Brillhart J. 1975. A method of factoring and the factorization of F_7. Math. Comp. 29: 183-205.

[14] See Pomerance C. Analysis and comparison of some integer factoring algorithms. *Computational Methods in Number Theory, Part I.* Amsterdam: Mathematisch Centrum, 1982.

[15] We stipulate that α is a generator to maximize the order of α, and thus avoid simplifying the problem of cracking the system.

[16] See Rubin K, Silverberg A. 2004. Using primitive subgroups to do more with fewer bits. Cryptology ePrint Archive.

[17] See Wang Y. 1961. On the least primitive root of a prime. Scientia Sinica 10(1): 1-14.

[18] See Shoup V. 1992. Searching for primitive roots in finite fields. Math. Comp. 58: 369-380.

[19] See Hellman ME and Pohlig S. 1978. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. IEEE Trans. Inform. Theor. 24: 106-110.

[20] This presentation is adapted from Koblitz N. *A Course in Number Theory and Cryptography*. New York: Springer-Verlag, 1994.

[21] For two good introductions, see (1) Silverman JH, Tate J. *Rational Points on Elliptic Curves.* New York: Springer-Verlag, 1992. (2) Washington LC. *Elliptic Curves*. Boca Raton: CRC Press, 2003.

[22] See Koblitz N. for details.

[23] Semaev I. 1998. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. Math. Comp. 67: 353-356.

[24] As reported by RSA Laboratories: http://www.rsasecurity.com/rsalabs/node.asp?id=2245