Computational Number Theory and Algebra April 18, 2012

Lecture 1

Lecturers: Markus Bläser, Chandan Saha *Scribe: Chandan Saha*

1 Overview

The past few decades have witnessed a growing interest among computer scientists and mathematicians, in the field of computational number theory and algebra. Computational number theory is the branch of computer science that involves finding efficient algorithms for algebraic and number theoretic problems: Factoring large integers, checking if an integer is prime, factoring polynomials, multiplying large integers and matrices, and solving polynomial equations are a few among a plethora of problems that have made this area so rich and fascinating. Owing to the fundamental nature of these problems, this is a subject of intense theoretical pursuit, and the tools and techniques developed to solve these problems have provided researchers with deep mathematical insights. The objective of this course is to make you become familiar with some of these tools and techniques, and thereby develop a sense of how algebraic methodologies are applied to design algorithms. Our focus would be on the theoretical aspects and asymptotic analysis of algorithms (instead of performance study of real world implementations).

Interest in this subject has escalated in the recent times because of their striking applications in key areas like cryptography, coding theory and computational complexity theory. To cite a few examples, the security and efficiency of cryptographic protocols such as the RSA cryptosystem and the Diffie-Hellman key exchange protocol, rely on the hardness of problems like integer factoring and discrete logarithm and on the efficiency of prime number generation and large integer multiplication. Further, the efficiency of some of the decoding algorithms for error correcting codes like Reed-Solomon and BCH codes, hinge on fast algorithms for solving a system of linear equations and factoring polynomials over finite fields. Another important application of algorithmic algebra is the development of computer algebra systems. These systems are indispensable tools for research in many computation intensive fields of physics, chemistry, biology, mathematics, geology and meteorology.

In today's class, we will see three such applications:

- The RSA cryptosystem
- The Diffie-Hellman key exchange protocol, and
- Reed-Solomon codes.

But, before we proceed into this subject matter, a few more words on what all do we expect to learn from this course.

As mentioned before, the course will cover some of the fundamental and widely used techniques for designing algebraic and number theoretic algorithms. Algebraic algorithms are used for performing operations like:

- **Polynomial operations:** Polynomial addition, multiplication, gcd computation, factoring, interpolation, multipoint evaluation, etc.
- **Matrix operations:** Matrix addition, multiplication, inverse and determinant computation, solving a system of linear equations, etc.
- **Abstract algebra operations:** Finding the order of a group element, computing discrete logarithm, etc.

Whereas number theoretic algorithms are used for performing operations like:

- **Operations with integers and rationals:** Addition, multiplication, gcd computation, square root finding, primality testing, integer factoring, etc.

Several important techniques are used to design algorithms for these above-mentioned operations. We will discuss some of these techniques in this course. These include:

1. **Chinese Remaindering:** Used in determinant computation and polynomial factoring.
2. **Discrete Fourier Transform:** Used in polynomial and integer multiplication.
3. **Automorphisms:** Used in polynomial and integer factoring, and primality testing.
4. **Hensel Lifting:** Used in polynomial factorization and division.
5. **Short Vectors in a Lattice:** Used in polynomial factoring and breaking cryptosystems.
6. **Smooth Numbers:** Used in integer factoring and computing discrete logarithm.
7. **Gröbner Bases:** Used in solving multivariate polynomial equations.
8. **Elliptic Curves:** Used in integer factoring, discrete logarithm, and cryptography.
9. **Algebraic independence:** Used in polynomial identity testing (complexity theory).

Prerequisite: This is an introductory graduate level course. We expect basic familiarity with linear algebra and properties of finite fields, or an undergraduate course in algebra, and elementary probability theory. Above all, we expect mathematical maturity from you.

Reference materials:

- Primary textbook: 'Modern Computer Algebra' by von zur Gathen and Gerhard.
- The book 'A Computational Introduction to Number theory and algebra', by Victor Shoup.
- The book 'Introduction to Finite Fields', by Lidl & Niederreiter.
- Lecture notes of a similar course offered by Manindra Agrawal. (http://www.cse.iitk.ac.in/users/manindra/CS681/index.html).
- Lecture notes of the course 'Algebra and Computation' offered by Madhu Sudan. (http://people.csail.mit.edu/madhu/FT98/).
- Course homepage: www.mpi-inf.mpg.de/departments/d1/teaching/ss12/CompNumberTheory/

General notations and conventions:

The set of integers, rationals, reals and complex numbers will be denoted by $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$, respectively. $\mathbb{Z}^+$ is the set of positive integers and $\mathbb{Z}_N$ is the ring of integers modulo $N \in \mathbb{Z}^+$. The multiplicative subgroup of $\mathbb{Z}_N$, consisting of all $m \in \mathbb{Z}^+$ with $\gcd(m, N) = 1$, is denoted by $\mathbb{Z}_N^\times$. For a positive real a, $\lfloor a \rfloor$ is the largest integer less than a, and $\lceil a \rceil$ is the smallest integer greater than a. $\mathbb{F}$ represents any arbitrary field, and $\mathbb{F}_q$ is the finite field with q elements. $\mathbb{F}_q^\times = \mathbb{F}_q \backslash \{0\}$ is the multiplicative subgroup of $\mathbb{F}_q$. The determinant of a square matrix M will be denoted by $\det(M)$.

Order ('big-oh') notation - Given two functions $t_1(n)$ and $t_2(n)$ from $\mathbb{Z}^+ \to \mathbb{Z}^+$, we say $t_1(n) = O(t_2(n))$ if there exist positive constants c and n_0 such that $t_1(n) \leq c \cdot t_2(n)$ for every $n \geq n_0$. We write $t_1(n) = o(t_2(n))$ if for every constant c, there is an $n_0 > 0$ such that $t_1(n) < c \cdot t_2(n)$ for all $n \geq n_0$. We use the notation $\mathsf{poly}(n_1, \ldots, n_k)$ to mean some arbitrary but fixed polynomial in the parameters $n_1, \ldots, n_k$. Throughout this course, log will refer to logarithm base 2 and ln, the natural logarithm. Sometimes, for brevity we will use the notation $\tilde{O}(t(n))$ to mean $O(t(n) \cdot \mathsf{poly}(\log t(n)))$.

2 Three applications in cryptography and coding theory

In this section, we take up the examples of the RSA cryptosystem and the Reed-Solomon codes, and show how they are intimately connected to problems like integer and polynomial factoring. As another motivating application, we give a brief account of the Diffie-Hellman key exchange protocol where the hardness of the discrete log problem comes into play.

2.1 The RSA cryptosystem

The RSA, named after its inventors Rivest, Shamir, and Adleman [RSA78], is a popular public-key cryptosystem. A cryptosystem is an encryption-cum-decryption scheme for communication between a sender and a receiver. Such a system is *secure* if it is infeasible for a (potentially malicious) third party to eavesdrop on the encrypted message and decrypt it efficiently. In a public-key cryptosystem, the receiver publishes a common key (also known as the *public key*) using which anyone can encrypt a message and send it to the receiver. On the other hand, only the receiver knows a secret private key using which the message can be decrypted efficiently. The RSA key generation procedure is as follows.

Algorithm 1 RSA: Key generation

1. Fix a key length, say, 2^r bits.
2. Randomly select two distinct primes p and q each of 2^{r-1} bits.
3. Let $n = pq$ and $\varphi(n) = (p-1)(q-1)$. /* φ is the totient function.*/
4. Randomly select an e such that $3 \leq e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Find the smallest d such that $d \cdot e = 1 \mod \varphi(n)$.
6. The encryption key is the pair (n, e).
7. The decryption key is d.

In practice, the primes p and q chosen in step 2 are fairly large in size, say about 512 bits. Evidently, a fast primality testing algorithm is required to pick such large primes. Other basic operations like integer multiplication (in step 3), gcd computation (in step 4) and modular inverse computation (in step 5) also play a significant role in determining the efficiency of the process, as the numbers involved are extremely large. The encryption key (n, e) is known to all, whereas the decryption key d is known only to the receiver. The RSA encryption scheme is as follows.

Algorithm 2 RSA: Encryption

1. Let m be the message to be encrypted.
2. Treat m as a number less than n, and assume that $\gcd(m, n) = 1$.
3. Compute $c = m^e \mod n$.
4. c is the encrypted message.

The assumption that $\gcd(m, n) = 1$ is not very binding - in practice, we can ensure this using a little bit of 'padding' on m. Note the usage of the modular exponentiation operation in step 3. At the receiver's end the message is decrypted using d as follows.

Algorithm 3 RSA: Decryption

1. Compute $c^d \mod n = m^{ed} \mod n = m$.

Since n is given as part of the public key, if we can factor n efficiently we can compute the private key $d = e^{-1} \mod \varphi(n)$ using extended Euclidean algorithm (to be covered in a later lecture). However, as yet no *efficient* (i.e. randomized polynomial time) algorithm is known for integer factoring. The current

best factoring algorithms ([JP92], [LJMP90]) have subexponential, but superpolynomial, time complexity. Although it is not known if breaking RSA is equivalent to integer factoring, a recent result by Aggarwal and Maurer [AM09] shows that under a reasonable restriction this is indeed the case.

If you are interested in further details on potential attacks on RSA cryptosystem, you may consult the survey articles by Koblitz and Menezes [KM04], Boneh [Bon99], or the book by Katzenbeisser [Kat01]. In a later lecture, we will discuss a lattice based attack on the RSA, which was first proposed by Coppersmith [Cop97].

Another problem which is historically closely related to integer factoring is the discrete logarithm problem. In the next section, we show how the *hardness* of this problem is used to design a secure key exchange protocol.

2.2 Diffie-Hellman key exchange protocol

Unlike a public-key cryptosystem where the encryption key is known to all, in a symmetric cryptosystem the two communicating parties use the same secret key for encryption as well as decryption. However, they are only allowed to exchange this key by communicating through a public channel that is accessible to all. Diffie and Hellman [DH76] presented a scheme that makes this possible without compromising the security of the communication. This scheme has applications in internet protocol security.

Let $\mathbb{F}_q$ be a finite field, where q is a large prime power. Let g be the generator of the multiplicative group $\mathbb{F}_q^\times$. Assume that the numbers q and g are known to all. The key-exchange protocol is as follows.

Algorithm 4 Diffie-Hellman key exchange

1. X chooses $a \in \mathbb{Z}_{q-1}$ uniformly at random, computes $x = g^a \in \mathbb{F}_q^\times$.
2. X sends x to Y.
3. Y chooses $b \in \mathbb{Z}_{q-1}$ uniformly at random, computes $y = g^b \in \mathbb{F}_q^\times$.
4. Y sends y to X.
5. X computes y^a, Y computes x^b.
6. X and Y use $y^a = x^b = g^{ab}$ as the secret key.

Since the communication uses a public channel, the numbers g^a and g^b are visible to all. If one can efficiently compute a from g and g^a, similarly b from g and g^b, one can also get the private key g^{ab}. The problem of computing z from g and g^z in $\mathbb{F}_q$ is the discrete logarithm problem. But just like integer factoring the current best algorithm [Gor93] for computing discrete logarithm has subexponential (but, once again super-polynomial) time complexity. It is not known though if breaking the Diffie-Hellman protocol is equivalent to computing discrete logarithm. For further details, you may refer to the article by Maurer and Wolf [MW99].

So far we have seen how the absence of efficient algorithms for certain number theoretic problems is exploited to design secure cryptographic protocols. Now we will see how fast algorithms for certain algebraic problems are used for designing efficient error-correcting codes.

2.3 The Reed-Solomon codes

The family of Reed-Solomon codes [RS60] is one of the most important and popular class of error-correcting codes which is used in many commercial applications, like encoding data on CD and DVD, and in data transmission and broadcast systems. Error-correcting codes enable us to encode a message into a codeword that is tolerant to errors during transmission. At the receiver's end the possibly corrupted codeword is decoded back to the original message.

Fix a finite field $\mathbb{F}_q$ - this is going to be the alphabet set. Treat a message m as a q-ary string i.e., a sequence of k elements of $\mathbb{F}_q$: $(m_0, \ldots, m_{k-1})$. Message m will be transformed into a codeword (also, a q-ary

string) consisting of n elements of $\mathbb{F}_q$. We will need the assumption that $q \geq n$. The Reed-Solomon encoding procedure is as follows.

Algorithm 5 Reed-Solomon codes: Encoding

1. Let $P_m(x) = \sum_{i=0}^{k-1} m_i x^i$.
2. Let $c_j = P_m(e_j)$ for $0 \leq j < n$ where $e_0, \ldots, e_{n-1}$ are distinct elements of $\mathbb{F}_q$.
3. The sequence $(c_0, \ldots, c_{n-1})$ is the codeword corresponding to m.

Note the use of multipoint evaluations of the polynomial $P_m(x)$ in step 2 of the procedure. The assumption $q \geq n$ is also used in step 2. Since, message length is k, there are q^k possible messages, and hence there are also q^k codewords (each corresponding to a unique message). The set of all codewords is called a *code*. *Distance* of a code is defined as the minimum Hamming distance between pairs of codewords (in the code) as q-ary strings. Claim: Distance of a Reed-Solomon code is exactly $n - k + 1$, and this is also the 'best' distance any code (coding scheme) can achieve. I leave the proof of this claim as an excercise.

Exercises:

1. Given integers m, e and n in binary, show that m^e modulo n can be computed in time $\mathsf{poly}(\log(emn))$.
2. Show that distance of a Reed-Solomon code is exactly $n - k + 1$.
3. Show that $n - k + 1$ is the best distance any code can achieve.

References

[AM09] Divesh Aggarwal and Ueli Maurer. Breaking RSA Generically is Equivalent to Factoring. In *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53. Springer-Verlag, 2009.

[Bon99] Dan Boneh. Twenty Years of Attacks on the RSA Cryptosystem. *Notices of the AMS*, 46:203–213, 1999.

[Cop97] Don Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[Gor93] Daniel M. Gordon. Discrete Logarithms in *GF(p)* Using the Number Field Sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.

[JP92] Hendrik W. Lenstra Jr. and Carl Pomerance. A Rigorous Time Bound for Factoring Integers. *Journal of the American Mathematical Society*, 5:483–516, 1992.

[Kat01] Stefan Katzenbeisser. *Recent Advances in RSA Cryptography.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[KM04] Neal Koblitz and Alfred J. Menezes. A Survey of Public-Key Cryptosystems. *SIAM Rev.*, 46(4):599–634, 2004.

[LJMP90] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse, and John M. Pollard. The Number Field Sieve. In *STOC*, pages 564–572, 1990.

[MW99] Ueli M. Maurer and Stefan Wolf. The Relationship Between Breaking the Diffie–Hellman Protocol and Computing Discrete Logarithms. *SIAM J. Comput.*, 28(5):1689–1721, 1999.

[RS60] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.