UNIVERSITY OF TRENTO

Complexity, Cryptography, and Financial Technologies

Lecture 6 – Introduction to Finite Fields and Number Theoretic Reference Problems

Chan Nam Ngo

Why do we need to study Finite Fields and the Number Theoretic Reference Problems?

- **To be able to**
  - understand the construction
  - and prove the security
  - or at least understand the security proof
- **of the**
  - upcoming cryptographic primitives
- **because they are based on Finite Fields and the Number Theoretic Reference Problems**

Informally, Finite Field is

- A finite set of numbers
- in which
  - the addition, subtraction, multiplication and division
  - can be carried out without any error
- Finite field is useful for crypto because
  - all arithmetic operations
  - must work without error for cryptography
- Stepping stones to Finite Field
  - Group
  - Ring

Group

- **Denoted as {G, +}**
  - G is the group
  - + is the binary operation (not necessarily addition)
- **As an example,**
  - the set of all integers N
  - and the addition operation +
  - is a group, denoted as {N,+}

Group Properties

- **closure**
  - if a, b $\in$ G, and c = a + b then c $\in$ G
  - {N,+} satisfies this? e.g. 3 = 1 + 2; 1,2,3 $\in$ N
- **commutativity (Abelian Group)**
  - a + b = b + a
  - {N,+} satisfies this? e.g. 1 + 2 = 2 + 1
- **associativity**
  - (a + b) + c = a + (b + c)
  - {N,+} satisfies this? e.g. (1 + 2) + 3 = 1 + (2 + 3)
- **identity element**
  - there exists an identity i s.t. for all elements a: a + i = a
  - What is the identity i of {N,+}?
    - Hint: 7 + ? = 7
- **inverse element**
  - there exists an inverse element b for each element a s.t. a + b = i where i is the identity
  - What is the inverse element of 9 in {G,+}?
    - Hint: 9 + ? = 0

Ring

- **Denoted as {R,+,*}**
  - R is the ring
  - + and * are two binary operations
    - + is normally addition
    - * is normally multiplication
  - Satisfies closure, commutativity, associativity (w.r.t. *)
- **{R,+,*} additionally satisfies**
  - distributivity (w.r.t *)
    - a*(b + c) = a*b + a*c
- **Is {N,+,*} a ring?**

Field

- **Denoted as {F,+,*}**
  - F is a field
  - + and * are two binary operations
- **A field is a ring with additional properties**
  - identity element for *
    - normally denoted as 1
    - if a ∈ F, a*1 = a
  - with regarding to identity element for +
    - normally we denote i as 0
    - if a*b = 0, then a = 0 or b = 0
  - multiplicative inverse
    - if a ∈ F AND a ≠ 0
    - then there exists b
    - such that a*b = 1
- **Is {N,+,*} a field?**

Modular Arithmetic

- **Modulo**
  - Given <u>any</u> integer a, e.g. 7
  - and a <u>positive</u> integer n, e.g. 3
  - we call a mod n the <u>remainder</u>, e.g. 1
    - 0 < a mod n < n – 1 (0 < 1 < 2)
- **if a mod n is 0 (e.g. a = 6 and n = 3)**
  - we call n a divisor of a
  - and write a | n
  - this implies the existence of an integer b where a = b*n (e.g. b = 2)
- **Congruence**
  - We call a and b congruent modulo n
  - If a mod n = b mod n
  - We can write a = b (mod n)
  - e.g. 7 = 1 (mod 3), 7 = 8 (mod 3)

- **The modulo n arithmetic**
  - maps the infinite set of all integers
  - into the finite set {0,...,n-1}
- **Additional properties of modulo n arithmetic**
  - (a mod n) + (b mod n) = (a + b) mod n
  - (a mod n) - (b mod n) = (a - b) mod n
  - (a mod n) * (b mod n) = (a * b) mod n

Finite Field (2)

- **Let us denote {Z,+,*} where**
  - Z = {0,...,n-1} (the set of integers from 0 to n-1)
  - + and * are modulo n addition and multiplication
- **We go ahead and check the properties**
  - Commutativity? YES
  - Associativity? YES
  - Distributivity? YES
  - Identity? YES
  - Inverse? Only additive inverse
    - We denote additive inverse of a as –a
    - We denote multiplicative inverse of a as a^{-1}

Why {Z,+,*} is still not a Finite Field?

- **Let us denote {Z,+,*} where**
  - Z = {0,...,n-1} (the set of integers from 0 to n-1)
  - + and * are modulo n addition and multiplication
- **{Z,+,*} is not a field ..., let's look at Z_6**

a	0	1	2	3	4	5
-a	0	5	4	3	2	1
a^{-1}	x	1	x	5	2	3

Prime Finite Field

- **To make Z_n a finite field, we must**
  - guarantee there is a multiplicative inverse
  - for every elements in Z_n
- **Multiplicative inverse only exists for elements that are relatively prime to n**
  - which means gcd(a,n) = 1
  - where gcd is short for Greatest Common Divisor
  - we can also say a and n are coprimes
  - Euclid's (extended) GCD algorithm for finding gcd(a,n) (Homework!!!)
- **We make n a prime, normally we denote such prime finite field as F_p**
  - $\{F_p,+,*\}$ where p is prime is a finite field
  - because all elements in F_p are relatively prime to p
  - To find multiplicative inverse, see Bezout's Identity (Homework!!!)

Primality Testing

- **To generate a large prime,**
  - randomly pick a large number
  - then run the Miller-Rabin primality test
- **Miller-Rabin Primality Test**
  - Most commonly used due to practical performance
  - Only a probabilistic assessment of primality
    - if output "not a prime" ("composite") → 100%
    - if output "prime" → may be a prime (probability > ½)
  - Based on Fermat's Little Theorem
    - Let p be a prime
    - If an integer a coprimes p
    - then $a^{p-1} = 1 \pmod{p}$
  - Algorithm
    - Randomly pick a large number n
    - {"composite", "probably prime"} ← Miller-Rabin(n)

{"composite", "probably prime"} ← Miller-Rabin(n)

UNIVERSITY OF TRENTO

- Randomly pick an integer a in [1,n-1]
- If a does not coprime n, i.e. gcd(a,n) ≠ 1
  - (e.g. test with Euclid's GCD algorithm)
  - Return "composite"
- Otherwise, write n – 1 in the form of $2^r d$ with d odd
- If $a^d = 1 \pmod n$
  - Return "probably prime"
- For all i = 0 to r-1 do
  - If $(a^{2^i m}) = -1 \pmod n$
    - Return "probably prime"
- Return "composite"

{"composite", "probably prime"} ← Miller-Rabin(252601)

UNIVERSITY OF TRENTO

- Pick a = 85132
- gcd(85132,252601) = 1
- 252601 – 1 = 252600 = $2^3 31575$
- $85132^{31575} = 191102 \neq 1$
- $85132^{2*31575} = 184829 \neq -1$
- $85132^{4*31575} = 1$
- Return "composite"

{“composite”, “probably prime”} ← Miller-Rabin(280001)

UNIVERSITY OF TRENTO

- Pick a = 105532
- gcd(105532,280001) = 1
- 280001 – 1 = 280000 = $2^6 4375$
- $105532^{4375} = 236926 \neq 1$
- $105532^{2*4375} = 168999 \neq -1$
- $105532^{4*4375} = 280000 = -1$
- Return “probably prime”

Discrete Logarithms (DLOG)

UNIVERSITY OF TRENTO

- **Fix a prime p and a group Z_p**
- **Let g be a generator of Z_p**
  - all elements of Z_p can be obtained from a power of g
  - Z_{11} has a generator g = 2 because
    - $\{2^0 = 1, 2^1 = 2, 2^8 = 3, 2^2 = 4, 2^4 = 5, 2^9 = 6, 2^7 = 7, 2^3 = 8, 2^6 = 9, 2^5 = 10\}$
- **Given y, find x s.t. $g^x = y$**
- **DLOG solving algorithms**
  - p is small, very easy, by exhaustive search
  - p is very large ($\sim 2^{512}$)
    - multiplicative group, hard (sub-exponential)
    - elliptic curve group, very hard (exponential)
      - this is why elliptic curve is important in crypto
      - we will introduce elliptic curve in an upcoming lecture
- **Related cryptographic primitives**
  - Diffie-Hellman Key Exchange
  - El-Gamal Cryptosystem

DLOG Algorithms - Baby-step Giant-step

- Given $y = g^x$
- Set $m = \sqrt{n}$ where n is the order of Z_p
  - n is the number of elements in Z_p
- We can write $x = i*m+j$ ($0 \le i < m$, $0 \le j < m$)
- Hence $g^x = g^{i*m+j}$
- Construct a table (j, g^j) for $0 \le j < m$, sorted by g^j
- Set $z = y$
- For i from 0 to m – 1 do
  - If $z = g^j$ for a j in the table (j,g^j)
    - Return $x = i*m+j$
  - Set $z = z*g^{-m}$ and continue

DLOG Algorithms - Baby-step Giant-step (2)

- Set m = $\sqrt{n}$ where n is the order of Z_p

→ runtime is O($\sqrt{n}$) but also requires O($\sqrt{n}$) storage

→ n = $2^{512} – 1$ ← runtime is exponential

- Example
  - p = 113, g = 3, n = 112, y = g^x = 57
  - m = $\sqrt{112}$ = 11

j	0	1	8	2	5	9	3	7	8	9	10
3^j	1	3	7	9	17	21	27	40	51	63	81

  - z = yg^{-mi}

i	0	1	2	3	4	5	6	7	8	9
z	57	29	100	37	112	55	26	39	2	3

→ x = 9*11+1 = 100

DLOG Algorithms - Others

- **Pollard's rho algorithm ← Preferable**
  - Randomized algorithm based on cycle finding
  - Same runtime as Baby-Step Giant-Step but less storage
- **Pohlig-Hellman algorithm**
  - Take advantage of factorization of n
  - Only efficient if n can be factored to relatively small primes
- **Index-calculus ← Most powerful**
  - Only for certain groups
  - Algorithm is sophisticated
  - Runtime is sub-exponential

Diffie-Hellman (DH) Key Exchange

- Alice and Bob wants to obtain a shared secret key for secure communication
- but Eve can see every information exchanged between Alice and Bob
- Can we construct a protocol such that Eve cannot derive the secret key from the public transcript?
- Based on problems related to DLOG
  - Computational DH
    - Given $a = g^x$, $b = g^y$, find $c = g^{xy}$
  - Decisional DH
    - Given $a = g^x$, $b = g^y$ and $c = g^z$, determine if $z = xy$

- **Alice**
  - Pick random x
  - Send g^x to Bob
  - Receive g^y
  - Compute $(g^y)^x$

- **Bob**
  - Pick random y
  - Send g^y to Alice
  - Receive g^x
  - Compute $(g^x)^y$

Diffie-Hellman (DH) Key Exchange (2)

- **Eve sees g^x and g^y**
- **But Eve cannot compute g^{xy} or g^{yx}**
  - Computational DH Assumption
    - Given $a = g^x$, $b = g^y$, find $c = g^{xy}$ is hard

- **Alice**
  - Pick random x
  - Send g^x to Bob
  - Receive g^y
  - Compute $(g^y)^x$
- **Bob**
  - Pick random y
  - Send g^y to Alice
  - Receive g^x
  - Compute $(g^x)^y$

Diffie-Hellman (DH) Key Exchange - Example

- **$p = 23, g = 5$**

- **Alice**
  - $x = 4$
  - $g^x = 4 \rightarrow$ To Bob
  - $(g^y)^x = 10^4 = 18$

- **Bob**
  - $y = 3$
  - To Alice $\leftarrow g^y = 10$
  - $(g^x)^y = 4^3 = 18$

Man In The Middle Attack

- **MITM Attack**
  - Eve intercepts g^x and g^y
  - Eve picks random z and sends g^z to both Alice and Bob
  - Eve can compute both g^{yz} and g^{xz}
  - Eve can use g^{yz} and g^{xz} to “bridge” the communication between Alice and Bob so they don’t find out about the attack
- **Alice and Bob can use digital signature to guarantee message authenticity**
  - Alice and Bob can tell if the message is indeed from the other party
- **but require a Public Key Infrastructure**

- **Alice**
  - Pick random x
  - Send g^x to Eve
  - Receive g^z from Eve
  - Compute $(g^z)^x$
- **Bob**
  - Pick random y
  - Send g^y to Eve
  - Receive g^z from Eve
  - Compute $(g^z)^y$

El-Gamal Cryptosystem – Public Key Encryption

- **(pk,sk) ← KeyGen()**
  - Fix a large prime p, a group Z_p and a generator g
  - Randomly pick x in Z_p
  - Compute $y = g^x$
  - Return pk = (p,g,y) and sk = (x)
- **c ← Enc(pk,m)**
  - Randomly pick r in Z_p
  - Compute $R = g^r$ and $M = my^r = mg^{xr}$
  - Return c = (R,M)
- **m = Dec(sk,c)**
  - Return $m = M/R^x = mg^{xr}/g^{rx}$

El-Gamal Cryptosystem – Public Key Encryption (2)

- **(pk,sk) ← KeyGen()**
  - Fix a large prime p, a group Z_p and a generator g
  - Randomly pick x in Z_p
  - Compute $y = g^x$
  - Return pk = (p,g,y) and sk = (x) ← Eve sees only $y = g^x$
- **c ← Enc(pk,m)**
  - Randomly pick r in Z_p
  - Compute $R = g^r$ and $M = my^r = mg^{xr}$
  - Return c = (R,M) ← Eve sees only g^r and mg^{xr}
- **m = Dec(sk,c)**
  - Return $m = M/R^x = mg^{xr}/g^{rx}$ ← cannot decrypt without x

El-Gamal Cryptosystem – Public Key Encryption - Example

UNIVERSITY OF TRENTO

- **(pk,sk) ← KeyGen()**
  - $p = 809$, $g = 16$
  - $x = 68$
  - $y = g^x = 46$
  - Return pk = (809,16,46) and sk = (68)
- **c ← Enc(pk,100)**
  - $r = 89$
  - $R = 16^{89} = 342$ and $M = 100*46^{89} = 745$
  - Return c = (342,745)
- **m = Dec(sk,c)**
  - Return $m = 745/342^{68} = 100$

El-Gamal Cryptosystem – Digital Signature

- **$(vk,sk) \leftarrow$ KeyGen()**
  - Fix a large prime p, a group Z_p and a generator g
  - Randomly pick x in Z_p
  - Compute $y = g^x$
  - Return vk = (p,g,y) and sk = (x)
- **$s \leftarrow$ Sign(sk,m)**
  - Pick k in Z_p s.t. gcd(k,p-1) = 1
  - Compute $R = g^k \pmod p$
  - Compute $S = (m-xR)/k \pmod{p-1} = (m - xg^k)/k$ $\rightarrow m = Sk + xR$
  - Return s = (R,S)
- **$\{0,1\} \leftarrow$ Verify(vk,s,m)**
  - Return 1 if $g^m = y^R R^S \pmod{p-1}$
    - $g^m = g^{Sk + xR} = g^{xR} g^{kS} = y^R R^S$

El-Gamal Cryptosystem – Digital Signature (2)

- **(vk,sk) ← KeyGen()**
  - Fix a large prime p, a group Z_p and a generator g
  - Randomly pick x in Z_p
  - Compute $y = g^x$
  - Return vk = (p,g,y) and sk = (x) ← Eve sees only $y = g^x$
- **s ← Sign(sk,m)**
  - Pick k in Z_p s.t. gcd(k,p-1) = 1
  - Compute $R = g^k \pmod p$
  - Compute $S = (m-xR)/k \pmod{p-1} = (m - xg^k)/k$

  → Eve cannot sign without x
  - Return s = (R,S)
- **{0,1} ← Verify(vk,s,m)**
  - Return 1 if $g^m = y^R R^S \pmod{p-1}$

El-Gamal Cryptosystem – Digital Signature - Example

- **(vk,sk) ← KeyGen()**
  - p = 467, g = 2
  - x = 127
  - $y = 2^{127} = 132$
  - Return vk = (467,2,132) and sk = 127
- **s ← Sign(sk,100)**
  - k = 213 and gcd(213,466) = 1
  - $R = 2^{213} = 29 \pmod{467}$
  - S = (m-xR)/k = (100-127*29)/ 213 = 51 (mod 466)
  - Return s = (29,51)
- **{0,1} ← Verify(vk,s,m)**
  - $2^{100} = 132^{29} * 29^{51} \pmod{466}$

Quadratic Residuosity Problem

- **Let p be a prime and a be an integer**
- **Determine if $x^2 = a \pmod p$ has a solution x**
  - a is called a quadratic residue (QR) modulo p if x exists
  - otherwise a is called quadratic non-residue (QNR)
- **The Legendre symbol is defined as**
  - $\left(\frac{a}{p}\right) = \begin{cases} 1 \; if \; a \; is \; a \; QR \\ -1 \; if \; a \; is \; a \; QNR \\ 0 \; if \; a = 0 \; mod \; p \end{cases}$
- **Deciding on QR/QNR**
  - p is small, very easy, by exhaustive search
  - p is large, infeasible
  - p is an odd prime,
    - $x^2 = a \pmod p$ has a solution x only if $a^{(p-1)/2} = 1 \pmod p$

Quadratic Residuosity Problem (2)

- Let N = pq, where p and q are large and <u>unknown</u> primes
- An integer a is QR modulo N if and only if a is QR modulo p and QR modulo q
- The Jacobi symbol is defined as
  - $\left(\frac{a}{N}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$
- If $\left(\frac{a}{N}\right)$ = 1, a is
  - either a QR modulo p and q ($\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1$)
  - or QNR modulo p and q ($\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$)

Quadratic Residuosity Problem (2)

- **Let N = pq where p and q are large and unknown primes**
- **Given an integer a where $\left(\frac{a}{N}\right) = 1$, determine whether a is a QR modulo N or not**
  - p and q are known, very easy
  - p and q are unknown, very hard
    - The Integer Factorization Problem
- **Related cryptographic primitives**
  - Goldwasser-Micali Cryptosystem
  - Blum Blum Shub Pseudo Random Generator

Integer Factorization Problem

- **also called Factoring**
  - Knowing that $N = pq$ with large prime numbers p and q. Find p and q
- **Algorithm**
  - Trial Division
    - Try small primes up to $\sqrt{N}$
  - Pollard's rho Factorization algorithm
    - Make use of Floyd's cycle finding algorithm
  - Pollard's p-1 Factorization algorithm
    - Find M s.t. $d = \gcd(N,M) \neq 1, N$. Then d will be p.
  - Difference of Squares
    - Find a and b s.t. $N = a^2 - b^2$
  - etc.

Difference of Squares

- $N = 25217$
- $b = 1$, $N + b^2 = 25217 + 1^2 = 25218$, not a perfect square
- $25217 + 2^2 = 25221$, not a perfect square
- $25217 + 3^2 = 25226$, not a perfect square
- $25217 + 4^2 = 25233$, not a perfect square
- ...
- $25217 + 8^2 = 25281 = 159^2$
- $25217 = (159+8)(159-8) = 167*151$

Goldwasser-Micali Cryptosystem – Public Key Bit Encryption

- **(pk,sk) ← KeyGen()**
  - Fix two large primes p and q
  - Compute N = pq
  - Find a QNR x s.t. $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right)$ = -1 (hence $\left(\frac{x}{N}\right)$ = 1)
  - Return pk = (N, x) and sk = (p,q)
- **c ← Enc(pk,b)**
  - Pick a random r s.t. gcd(r,N) = 1
  - Return $c = r^2x^b$
- **b = Dec(sk,c)**
  - Return b = 0 if c is QR modulo N ($c = r^2x^0 = r^2$)
  - Otherwise return b = 1 ($c = r^2x^1 = r^2x$)

Goldwasser-Micali Cryptosystem – Public Key Bit Encryption (2)

- **(pk,sk) ← KeyGen()**
  - Fix two large primes p and q
  - Compute N = pq
  - Find a QNR x s.t. $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$ (hence $\left(\frac{x}{N}\right) = 1$)
  - Return pk = (N, x) and sk = (p,q) ← Eve cannot see p, q
- **c ← Enc(pk,b)**
  - Pick a random r s.t. gcd(r,N) = 1
  - Return $c = r^2x^b$
- **b = Dec(sk,c)**
  - Return b = 0 if c is QR modulo N ($c = r^2x^0 = r^2$)
  - Otherwise return b = 1 ($c = r^2x^1 = r^2x$) ← Cannot decide QR modulo p and q without sk

Goldwasser-Micali Cryptosystem – Example

- **(pk,sk) ← KeyGen()**
  - p = 7, q = 11, N = 7*11 = 77
  - x = 6 and $\left(\frac{6}{7}\right) = \left(\frac{6}{11}\right) = -1$ (hence $\left(\frac{6}{77}\right) = 1$)
  - pk = (77,6) and sk = (7,11)
- **c ← Enc(pk,1)**
  - r = 2 and gcd(2,77) = 1
  - $c = 2^2 6^1 = 24$
- **b = Dec(sk,c)**
  - $24^{(7-1)/2} = -1$
  - Return 1

Blum Blum Shub Pseudo Random Generator

UNIVERSITY OF TRENTO

- **To generate a pseudo random bit sequence $b_1, b_2, \ldots b_n$**
- **Fix two large and secret primes p and q**
  - s.t. p = q = 3 (mod 4)
  - guarantee a QR has a square root that is also a QR
- **Compute N = pq**
- **Select a random seed s s.t. gcd(s,N) = 1**
- **Compute $x_0 = s^2$**
- **For i from 1 to n do**
  - $x_i = (x_{i-1})^2$
  - Set b_i = the least significant bit of x_i
- **To predict bit b_{i+1}?**
  - Difficult, see the proof in the original paper

Blum Blum Shub Pseudo Random Generator - Example

- $n = 5$
- $p = 11, q = 9$
- $N = 11*9 = 99$
- $s = 3$ and $gcd(3,99) = 1$
- $x_0 = 3^2 = 9$
- $x_1 = 81, x_2 = 82, x_3 = 36, x_4 = 42, x_5 = 92$
- Output 110000

Suggested Readings

- **Handbook of Applied Cryptography – Book by Menezes, C. van Oorschot and Vanstone**
  - See
    - Chapter 2 for Finite Fields
    - Chapter 3 for Number Theoretic Reference Problems
    - Chapter 5 for Pseudo Random Generators
    - Chapter 8 for Public Key Cryptosystems
    - Chapter 11 for Digital Signature Schemes
  - Also available on the author's website

Lab on Finite Fields and others

- **libsnark will be our main crypto library**
  - https://github.com/scipr-lab/libsnark
  - At the beginning we will only make use of libsnark's dependency
    - GMP for arithmetics
    - Boost for multi-threading, etc.
    - Built in Finite Field and Elliptic Curve lib
  - At the end we will use libsnark for implementing zk-SNARK
- **Students TODO:**
  - Register on Google Classroom
  - Obtain invitation to a private github repo created by instructors
  - **Watch for announcement on Google Classrom**
  - Pull project templates or some codes (prepared by instructors) from the private github repo
    - e.g. Repo/Lab1/Template
  - Implement something during lab session
  - Submit into a submission folder for each lab session
    - e.g. Repo/Lab1/Student/FirstName_LastName/