Overview & Fundamental Structures / Algorithms

Prof. dr. ir. Frederik Vercauteren

COSIC, ESAT, KU Leuven
Computer Algebra for Cryptography (B-KUL-H0E74A)

2022-2023

Overview of course

Fundamental concepts

Basic algorithms

Contact details

Lecturer:

- Wouter Castryck (ESAT B01.10)
  - wouter.castryck@esat.kuleuven.be

Teaching assistants:

- Thomas Decru (ESAT B01.15)
  - thomas.decru@esat.kuleuven.be
- Jiayi Kang (ESAT B01.21)
  - jiayi.kang@esat.kuleuven.be

Please post scientific questions on:

- Toledo → Course Documents → Forum for discussions and questions

Computer algebra

- Manipulating objects from
  - finite structures, e.g., finite fields $\mathbb{F}_q$, finite groups, ...
  - discrete or enumerable structures, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{F}_q[x]$, lattices, ...
  - ...

  that can be represented **exactly**.

Computer algebra

- Manipulating objects from
  - finite structures, e.g., finite fields $\mathbb{F}_q$, finite groups, ...
  - discrete or enumerable structures, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{F}_q[x]$, lattices, ...
  - ...

  that can be represented **exactly**.
- Often no notion of continuity: algorithms from numerical analysis may not apply.
- Typically: want **exact** solutions, obtained via symbolic manipulations.
- Sometimes: work with approximations, e.g.,
  - evaluate
    $$\mathbb{Z}_{\geq 1} \to \mathbb{Z}_{\geq 1} : n \mapsto \lfloor \sqrt{n} \rfloor$$
    by computing $\sqrt{n}$ to sufficient precision using Newton iteration and round,
  - compute number of solutions to $f(x, y) = 0$ over $\mathbb{F}_q$ using p-adic approximation,

  but end result is always **exact**.

Cryptography

- History: data confidentiality
- Nowadays: growing set of tools for secure communication, such as
  - data confidentiality and data integrity,
  - authentication and non-repudiation,
  - computing with encrypted data,
  - ...

Cryptography

- History: data confidentiality
- Nowadays: growing set of tools for secure communication, such as
  - data confidentiality and data integrity,
  - authentication and non-repudiation,
  - computing with encrypted data,
  - ...
- Key distinction:
  - **Symmetric-key cryptography**:
    - both parties share a secret key that is used to encrypt/decrypt & authenticate messages between them,
  - **Public-key cryptography** (= asymmetric cryptography):
    - each party has a key pair: a public key which is used to encrypt to the owner of the key, and a private key which is used to decrypt,
    - compare with letterbox: public key is letterbox, private key is key that opens its lock,
    - digital signatures: public key owner proves knowledge of corresponding private key

Cryptanalysis

- The security level of a cryptosystem:
  - system is λ-**bit secure** if best attacks require 2^{λ} bit operations,
  - common security levels: 80-bit (phased out), 112-bit and 128-bit.
- This is determined by **cryptanalysis**:
  - analyze theoretical **time complexity** (and also space complexity) of best attacks,
  - run them on small instances and extrapolate.
- Algorithms from computer algebra are fundamental in cryptanalysis
  - many cryptosystems were/are broken by the algorithms in this course,
  - continuous improvement leads to revised security estimates and parameters.

Post-quantum cryptography

- Quantum computers: work with qubits instead of bits.
- Can be in superposition of basic states, e.g. $\alpha|0\rangle + \beta|1\rangle$.

Post-quantum cryptography

- Quantum computers: work with qubits instead of bits.
- Can be in superposition of basic states, e.g. $\alpha|0\rangle + \beta|1\rangle$.
- **Shor's algorithm** breaks all conventional public-key cryptography:
  - RSA (by solving factorization problem),
  - DSA, ECDSA, EdDSA (by solving the discrete logarithm problem).

Post-quantum cryptography

- Quantum computers: work with qubits instead of bits.
- Can be in superposition of basic states, e.g. $\alpha|0\rangle + \beta|1\rangle$.
- **Shor's algorithm** breaks all conventional public-key cryptography:
  - RSA (by solving factorization problem),
  - DSA, ECDSA, EdDSA (by solving the discrete logarithm problem).
- For the time being: theoretical device
  - November 2022: IBM announces 433-qubit quantum computer,
  - breaking RSA-2048 may require millions of qubits.

Post-quantum cryptography

- Quantum computers: work with qubits instead of bits.
- Can be in superposition of basic states, e.g. $\alpha|0\rangle + \beta|1\rangle$.
- **Shor's algorithm** breaks all conventional public-key cryptography:
  - RSA (by solving factorization problem),
  - DSA, ECDSA, EdDSA (by solving the discrete logarithm problem).
- For the time being: theoretical device
  - November 2022: IBM announces 433-qubit quantum computer,
  - breaking RSA-2048 may require millions of qubits.
- NIST post-quantum cryptography standardization effort:
  - first standards in July 2022:
    - CRYSTALS-Kyber (key encapsulation),
    - CRYSTALS-Dilithium, Falcon, SPHINCS+ (digital signatures),
  - more schemes in pipeline + renewed call for digital signatures (June 2023).

Overview of course

- 9 lectures and 10 exercise sessions,
- 1. Introduction and fundamental algorithms, structures
  2. Multivariate polynomials, ideals and Gröbner bases
  3. Cryptographic applications of Gröbner bases: multivariate cryptography
  4. Lattices, hard problems SVP & CVP, lattice reduction
  5. Cryptographic applications of lattices: knapsack problems, Coppersmith's algorithm
  6. The Learning with Errors problem: cryptographic applications
  7. Fast multiplication and division: evaluation/interpolation approach, Karatsuba, Toom-Cook, DFT & FFT
  8. Polynomials: fast evaluation & interpolation, factorization (square-free, distinct degree, equal degree), Berlekamp's algorithm
  9. Primality testing, factorization algorithms, index calculus, applications to RSA

Exercise sessions

- 10 sessions in PC lab using computer algebra package MAGMA
- If you added your MAC address in Google Sheets:
  - follow installation instructions on Toledo,
  - if you forgot to do this, please add your MAC address and send notification e-mail.
- Quick experiments can be carried out in the MAGMA calculator.
- 2 introductory sessions + 4 sessions per project
- 2 available time slots
  - this week: on Thursday at 8am and at 4pm in room 01.52,
  - later weeks: to be finalized soon (see Toledo and official timetable).

Evaluation

- **Goal:** solve 2 projects on applications of computer algebra in cryptography.
- Each project results in MAGMA code and a written report (of max. 5 pages, excluding references).
- Report should be **concise** (no need to repeat the project description, focus on the solution & experiments).
- Oral exam = **discussion about the 2 projects**.
- Each project is 35% of marks and oral exam 30%.

Tips and tricks 

- Follow the specified API (types and number of arguments of inputs and ouputs must match).
- Read the submission instructions (it will be one .m file and one .pdf file).
- Know the difference between collaboration and plagiarism (sharing ideas is fine and even encouraged, but no copying of code or report contents).
- Test-run your own implementation!
- Overall goal: to independently solve problems by having access to all possible resources; the solution to the problems is not sketched during lectures.
- Know when you need to think vs. when you need to Google.

Overview of course

Fundamental concepts

Basic algorithms

"Big Oh" and "little oh" notation

- Express the complexity of an algorithm up to a constant factor
- Given two functions $f, g : \mathbb{R} \to \mathbb{R}$ we say that $f \in \mathcal{O}(g)$ if there exist positive constants C and D such that

$$|f(x)| \leq C|g(x)| \quad \text{for all } x > D$$

- Sometimes also write $f = \mathcal{O}(g)$
- $\widetilde{\mathcal{O}}$ ignores logarithmic factors, e.g. $\mathcal{O}(\log n) \in \widetilde{\mathcal{O}}(1)$

"Big Oh" and "little oh" notation

- Express the complexity of an algorithm up to a constant factor
- Given two functions $f, g : \mathbb{R} \to \mathbb{R}$ we say that $f \in \mathcal{O}(g)$ if there exist positive constants C and D such that

$$|f(x)| \leq C|g(x)| \quad \text{for all } x > D$$

- Sometimes also write $f = \mathcal{O}(g)$
- $\widetilde{\mathcal{O}}$ ignores logarithmic factors, e.g. $\mathcal{O}(\log n) \in \widetilde{\mathcal{O}}(1)$
- Given two functions $f, g : \mathbb{R} \to \mathbb{R}$ we say that $f \in o(g)$ if **for all positive** C, there exists some D such that

$$|f(x)| < C|g(x)| \quad \text{for all } x > D$$

"Big Oh" classes of functions

Notation	Name
$\mathcal{O}(1)$	constant
$\mathcal{O}(\log n)$	logarithmic
$\mathcal{O}((\log n)^c)$	polylogarithmic
$\mathcal{O}(n)$	linear
$\mathcal{O}(n^2)$	quadratic
$\mathcal{O}(n^c)$	polynomial
$\mathcal{O}(c^n)$	exponential

- Complexity of algorithm is given in the size of the input

"Big Oh" classes of functions

Notation	Name
$\mathcal{O}(1)$	constant
$\mathcal{O}(\log n)$	logarithmic
$\mathcal{O}((\log n)^c)$	polylogarithmic
$\mathcal{O}(n)$	linear
$\mathcal{O}(n^2)$	quadratic
$\mathcal{O}(n^c)$	polynomial
$\mathcal{O}(c^n)$	exponential

- Complexity of algorithm is given in the size of the input
- **Sub-exponential** L-notation (with $n = \log N$):

$$L_N(a, b) = \exp\left((b + o(1)) \log(N)^a \log\log(N)^{(1-a)}\right)$$

  - $L_N(0, b) = (\log N)^{b+o(1)}$ (polynomial in input size)
  - $L_N(1, b) = N^{b+o(1)}$ (exponential in input size)

"Big Oh" classes of functions

Notation	Name
$\mathcal{O}(1)$	constant
$\mathcal{O}(\log n)$	logarithmic
$\mathcal{O}((\log n)^c)$	polylogarithmic
$\mathcal{O}(n)$	linear
$\mathcal{O}(n^2)$	quadratic
$\mathcal{O}(n^c)$	polynomial
$\mathcal{O}(c^n)$	exponential

- Complexity of algorithm is given in the size of the input
- **Sub-exponential** L-notation (with $n = \log N$):

$$L_N(a, b) = \exp\left((b + o(1))\log(N)^a \log\log(N)^{(1-a)}\right)$$

  - $L_N(0, b) = (\log N)^{b+o(1)}$ (polynomial in input size)
  - $L_N(1, b) = N^{b+o(1)}$ (exponential in input size)
- **Exercise:** for all $\epsilon \in (0, \frac{1}{2}]$ there exists $b > 0$ such that $\sqrt{n}^{\sqrt{n}} \in L_N(\frac{1}{2} + \epsilon, b)$.

Modular arithmetic

- Fundamental to cryptography, coding theory, ...
- Given modulus $N \in \mathbb{Z}_{>0}$, consider the finite ring $\mathbb{Z}/N\mathbb{Z}$ (some write $\mathbb{Z}_N$)
- Each element of $\mathbb{Z}/N\mathbb{Z}$ is an equivalence class of the form

$$k + N\mathbb{Z}.$$

- $k_1 \equiv k_2 \bmod N$ means: $k_1 \in k_2 + N\mathbb{Z}$.
- Example: $11^8 \equiv (-2)^8 \equiv (-2)^4 \cdot (-2)^4 \equiv 16 \cdot 16 \equiv 3 \cdot 3 \equiv 9 \bmod 13$.

Modular arithmetic

- Fundamental to cryptography, coding theory, ...
- Given modulus $N \in \mathbb{Z}_{>0}$, consider the finite ring $\mathbb{Z}/N\mathbb{Z}$ (some write $\mathbb{Z}_N$)
- Each element of $\mathbb{Z}/N\mathbb{Z}$ is an equivalence class of the form

$$k + N\mathbb{Z}.$$

- $k_1 \equiv k_2 \bmod N$ means: $k_1 \in k_2 + N\mathbb{Z}$.
- Example: $11^8 \equiv (-2)^8 \equiv (-2)^4 \cdot (-2)^4 \equiv 16 \cdot 16 \equiv 3 \cdot 3 \equiv 9 \bmod 13$.
- Two common choices to represent a class:
  - Smallest non-negative integer in class, i.e., unique representant in $\{0, 1, 2, \ldots, N-1\}$
  - Smallest integer (in absolute value) in class, i.e., in $\{-\lfloor (N-1)/2 \rfloor, \ldots, \lfloor N/2 \rfloor\}$
  - After addition/multiplication, require a reduction to find representative of class
  - Lazy reduction: do several computations first (typically additions) before reducing
- MAGMA : `Z13 := Integers(13); a := Z13 ! 11; a^8;`

Modular inverse

- Multiplicative inverse: given element $a \in \mathbb{Z}/N\mathbb{Z}$, does an $x \in \mathbb{Z}/N\mathbb{Z}$ exist with

$$a \cdot x \equiv 1 \bmod N?$$

- Answer: yes if and only if $\gcd(a, N) = 1$
- XGCD algorithm on input a, N can be used to compute x if it exists
- MAGMA : `d, x, y := XGCD(a, N); // d = a*x + N*y`
- For $N = p$ a prime, each $a \not\equiv 0 \bmod p$ has multiplicative inverse, so the ring $\mathbb{Z}/p\mathbb{Z}$ becomes a field and is denoted $\mathbb{F}_p$, or sometimes $\mathrm{GF}(p)$
- MAGMA :
```
Fp := GF(31);  a := Fp ! 7;  x := a^-1;
Fp := GF(NextPrime(10^100));  a := Fp ! 11;  x := a^-1;
```

Multiplicative group

- All elements in $\mathbb{Z}/N\mathbb{Z}$ that are invertible, form a group for multiplication

$$(\mathbb{Z}/N\mathbb{Z})^* = \{a \in \mathbb{Z}/N\mathbb{Z} \mid \gcd(a, N) = 1\}$$

- Its number of elements is denoted $\varphi(N)$ (Euler-phi, Euler's totient function)

Multiplicative group

- All elements in $\mathbb{Z}/N\mathbb{Z}$ that are invertible, form a group for multiplication

$$(\mathbb{Z}/N\mathbb{Z})^* = \{a \in \mathbb{Z}/N\mathbb{Z} \mid \gcd(a, N) = 1\}$$

- Its number of elements is denoted $\varphi(N)$ (Euler-phi, Euler's totient function)
- Examples:
  - For $N = p$, a prime, we have $\varphi(p) = p - 1$
  - For $N = p \cdot q$, a product of two different primes, we have $\varphi(p \cdot q) = (p - 1)(q - 1)$
  - For $N = p^k$, a power of a prime, we have $\varphi(p^k) = (p - 1)p^{k-1}$
  - Totally general: $N = \prod_{i=1}^{s} p_i^{e_i}$ we have

$$\varphi(N) = \prod_{i=1}^{s} (p_i - 1)p_i^{e_i - 1}$$

- MAGMA : `phiN := EulerPhi(N);`

Order - Theorem of Lagrange

- The order of an element $a \in (\mathbb{Z}/N\mathbb{Z})^*$ is the smallest $r \in \mathbb{Z}_{>0}$ with $a^r \equiv 1 \bmod N$
- Each element $a \in (\mathbb{Z}/N\mathbb{Z})^*$ generates a subgroup of $(\mathbb{Z}/N\mathbb{Z})^*$:

$$\langle a \rangle = \{a, a^2, \ldots, a^r = 1\}$$

- MAGMA : `Z31 := Integers(31); o := Order(Z31 ! 7);`
- Lagrange: the order of $a \in (\mathbb{Z}/N\mathbb{Z})^*$ divides $\varphi(N) = \#(\mathbb{Z}/N\mathbb{Z})^*$

Order - Theorem of Lagrange

- The order of an element $a \in (\mathbb{Z}/N\mathbb{Z})^*$ is the smallest $r \in \mathbb{Z}_{>0}$ with $a^r \equiv 1 \bmod N$
- Each element $a \in (\mathbb{Z}/N\mathbb{Z})^*$ generates a subgroup of $(\mathbb{Z}/N\mathbb{Z})^*$:

$$\langle a \rangle = \{a, a^2, \ldots, a^r = 1\}$$

- MAGMA : `Z31 := Integers(31); o := Order(Z31 ! 7);`
- Lagrange: the order of $a \in (\mathbb{Z}/N\mathbb{Z})^*$ divides $\varphi(N) = \#(\mathbb{Z}/N\mathbb{Z})^*$
- Consequence: Euler's congruence

$$\text{if } \gcd(a, N) = 1, \text{ then } a^{\varphi(N)} \equiv 1 \bmod N$$

- Special case: Fermat's Little Theorem (FLT)

$$\text{if } p \text{ is prime then } \forall a \in \mathbb{Z} : a^p \equiv a \bmod p.$$

Univariate polynomial rings

- Given a field F, e.g., $F = \mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{F}_p$, we consider the polynomial ring

$$F[x] = \left\{ a(x) = \sum_{i=0}^{n} a_i x^i \;\middle|\; n \in \mathbb{Z}_{\geq 0}, a_i \in F \right\}$$

- Can assume $a_n \neq 0$: then n is called the **degree** and a_n the **leading coefficient**.
- If $a_n = 1$, then $a(x)$ is called **monic**.

Univariate polynomial rings

- Given a field F, e.g., $F = \mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{F}_p$, we consider the polynomial ring

$$F[x] = \left\{ a(x) = \sum_{i=0}^{n} a_i x^i \;\middle|\; n \in \mathbb{Z}_{\geq 0}, a_i \in F \right\}$$

- Can assume $a_n \neq 0$: then n is called the **degree** and a_n the **leading coefficient**.
- If $a_n = 1$, then $a(x)$ is called **monic**.
- Addition is simply $c(x) = a(x) + b(x) = \sum_{i=0}^{\max\{n,m\}} (a_i + b_i) x^i$
- Multiplication is $c(x) = \sum_{i=0}^{n+m} c_i x^i$ with

$$c_i = \sum_{\substack{0 \leq j \leq n, 0 \leq k \leq m, \\ j+k=i}} a_j \cdot b_k$$

- Note: on $\mathbb{F}_p$ the **functions** $x \mapsto x$ and $x \mapsto x^p$ are the same (FLT), yet $x, x^p \in \mathbb{F}_p[x]$ are distinct **polynomials**.

Modular arithmetic in polynomial rings

- Similar construction to $\mathbb{Z}/N\mathbb{Z}$, but now starting from $F[x]$
- Let $f(x)$ be polynomial of degree n, then can consider quotient ring $F[x]/(f(x))$, now every element is an equivalence class of the form

$$g(x) + f(x)F[x] = g(x) + (f(x)).$$

- Example: $x^2 = (x+1)(x-1) + 1 \equiv 1 \bmod x - 1$.

Modular arithmetic in polynomial rings

- Similar construction to $\mathbb{Z}/N\mathbb{Z}$, but now starting from $F[x]$
- Let $f(x)$ be polynomial of degree n, then can consider quotient ring $F[x]/(f(x))$, now every element is an equivalence class of the form

$$g(x) + f(x)F[x] = g(x) + (f(x)).$$

- Example: $x^2 = (x+1)(x-1) + 1 \equiv 1 \bmod x - 1$.
- $F[x]$ is a Euclidean ring, i.e., we can write each polynomial $g(x) \in F[x]$ as

$$g(x) = q(x) \cdot f(x) + r(x) \qquad \text{with degree of } r(x) < n$$

- Each equivalence class in $F[x]/(f(x))$ thus has a representative of the form

$$\sum_{i=0}^{m} g_i x^i \quad \text{with } m < n\,.$$

Modular inverse in $F[x]/(f(x))$

- Element $g(x) \in F[x]/(f(x))$ has modular inverse $h(x) \in F[x]/(f(x))$ if

$$g(x) \cdot h(x) = 1 \bmod f(x)$$

- Modular inverse exists if and only if $\gcd(g(x), f(x)) = 1$.
- MAGMA :
```
R<x> := PolynomialRing(Rationals());  f := x^5 - 1;
Rmodf<x> := quo<R | f>; (x-1)^-1;
> Runtime error in '^': Argument is not invertible
```
- XGCD algorithm can be used to compute $h(x)$
- Corollary: if $f(x)$ is irreducible, then $F[x]/(f(x))$ is a field

Modular inverse in $F[x]/(f(x))$

- Element $g(x) \in F[x]/(f(x))$ has modular inverse $h(x) \in F[x]/(f(x))$ if

$$g(x) \cdot h(x) = 1 \bmod f(x)$$

- Modular inverse exists if and only if $\gcd(g(x), f(x)) = 1$.
- MAGMA :
```
R<x> := PolynomialRing(Rationals());  f := x^5 - 1;
Rmodf<x> := quo<R | f>; (x-1)^-1;
> Runtime error in '^': Argument is not invertible
```
- XGCD algorithm can be used to compute $h(x)$
- Corollary: if $f(x)$ is irreducible, then $F[x]/(f(x))$ is a field
- **Theorem** For every prime p and every $n \geq 1$, there exists an irreducible polynomial $f(x)$ of degree n in $\mathbb{F}_p[x]$
- Obtain $\mathbb{F}_q = \mathbb{F}_p[x]/(f(x))$ finite field with $q = p^n$ elements

Finite fields in MAGMA

- `Fp := GF(31);`
- `R<x> := PolynomialRing(Fp);`
- `g := x^3 + x + 3;  // IsIrreducible(g); returns true`
- `Fq<w> := ext<Fp | g>;`
- `w^10;`

  `> 5*w^2 + 16*w + 27`
- `Roots(g, Fq);`

  `> [ <3*w^2 + 17*w + 2, 1>, <w, 1>, <28*w^2 + 13*w + 29, 1> ]`
- `w^31;`

  `> 28*w^2 + 13*w + 29`
- `w^(31^2);`

  `> 3*w^2 + 17*w + 2`

Frobenius

- Let $\mathbb{F}_q$ be a finite field of **characteristic** p, i.e. $q = p^n$
- Frobenius $=$ p-th powering
- Frobenius is additive: for all $a, b \in \mathbb{F}_q$ we have

$$(a + b)^p = a^p + b^p$$

Frobenius

- Let $\mathbb{F}_q$ be a finite field of **characteristic** p, i.e. $q = p^n$
- Frobenius $=$ p-th powering
- Frobenius is additive: for all $a, b \in \mathbb{F}_q$ we have

$$(a+b)^p = a^p + b^p$$

- FLT ($+$ Euclidean division): the polynomial $x^p - x$ splits completely over $\mathbb{F}_p$ as

$$x^p - x = \prod_{a \in \mathbb{F}_p} (x - a)$$

- Given polynomial $f(x) \in \mathbb{F}_p[x]$, can test if $f(x)$ has root in $\mathbb{F}_p$ simply by computing (see later)

$$\gcd(x^p - x, f(x))$$

Comparison

Structure:	$\mathbb{Z}$	$F[x]$
Units (invertible):	± 1	non-zero elements in F
Building blocks:	primes p	monic irreducible $f(x)$
Quotient ring:	$\mathbb{Z}/N\mathbb{Z}$	$F[x]/(f(x))$
Representation:	$[0, \ldots, N-1]$	$\sum_{i=0}^{m} g_i x^i$ with $g_i \in F, m < n$
Field:	N prime	$f(x)$ monic irreducible

Comparison

Structure:	$\mathbb{Z}$	$F[x]$
Units (invertible):	± 1	non-zero elements in F
Building blocks:	primes p	monic irreducible $f(x)$
Quotient ring:	$\mathbb{Z}/N\mathbb{Z}$	$F[x]/(f(x))$
Representation:	$[0, \ldots, N-1]$	$\sum_{i=0}^{m} g_i x^i$ with $g_i \in F, m < n$
Field:	N prime	$f(x)$ monic irreducible

- Analogy between $\mathbb{Z}$ and $F[x]$ is famous, especially for $F = \mathbb{F}_q$: Prime Number Theorem, Riemann Hypothesis, ...
- Usually $F[x]$ is easier (e.g., factorization, see Lecture 8)

Representation of integers

- Computers typically use 32-or 64-bit words
- Multiprecision integer: array of such words, together with length and sign bit

$$a = (-1)^s \sum_{0 \le i \le n} a_i \cdot 2^{wi}$$

- w: word-size; $s \in \{0, 1\}$: sign bit;
- $a_i \in \{0, \ldots, 2^w - 1\}$ the digits
- Resembles polynomial arithmetic, but have problem of carries since a_i are normalized:

$$a_i + b_i = c_i + \gamma 2^w$$

- Carry $\gamma \in \{0, 1\}$ influences the next term c_{i+1}

Modular exponentiation

- Let R be $\mathbb{Z}/N\mathbb{Z}$ or $F[x]/(f(x))$
- Given $a \in R$, often need (huge) powers $a^n \in R$ for $n \in \mathbb{Z}$
- **Function** SquareMultiply(a, n)

1: Write n in binary as $n = 2^k + n_{k-1}2^{k-1} + \ldots + n_0$
2: $b_k \leftarrow a$
3: **for** $i = k-1, \ldots, 0$ **do**
4: **if** $n_i = 1$ **then**
5: $b_i \leftarrow b_{i+1}^2 \cdot a$
6: **else**
7: $b_i \leftarrow b_{i+1}^2$
8: **end if**
9: **end for**
10: **return** b_0

Modular exponentiation - Application

- RSA cryptosystem (see later): need to compute $m^e \bmod N$
- Computing the inverse modulo p via FLT
- Recall FLT: if $p \nmid a$, then $a^{p-1} = 1 \bmod p$

$$\Rightarrow a^{-1} = a^{p-2} \bmod p$$

- Requires $\sim 1.5 \log_2 p$ operations modulo p
- Much more general via Lagrange ..., e.g.

$$\text{For } a \in \mathbb{F}_{p^n}, a \neq 0 : a^{-1} = a^{p^n-2}$$

- MAGMA :

```
ZN := Integers(101); a := ZN ! 13; b := a^7485718947987324;
```

- (Don't!) compare with `13^7485718947987324 mod 101;`

Euclidean domain

- Integers and polynomials over a field are examples of a **Euclidean domain**
- Euclidean domain: division with remainder is possible
  - Remainder has "smaller" size than the divisor measured by function $d : R \to \mathbb{N} \cup \{-\infty\}$:

$$\forall a, b \neq 0 \in R, \exists q, r \in R : a = qb + r \quad \text{and } d(r) < d(b)$$

Euclidean domain

- Integers and polynomials over a field are examples of a **Euclidean domain**
- Euclidean domain: division with remainder is possible
  - Remainder has "smaller" size than the divisor measured by function $d : R \to \mathbb{N} \cup \{-\infty\}$:

$$\forall a, b \neq 0 \in R, \exists q, r \in R : a = qb + r \quad \text{and } d(r) < d(b)$$

- Examples:
  - $R = \mathbb{Z}$, with $d(a) = |a|$
  - $R = F[x]$ with F a field, $d(a) = \deg(a)$
  - $R = \mathbb{Z}[\mathbf{i}] = \{a + b\mathbf{i} \mid a, b \in \mathbb{Z}\}$ with $d(a + b\mathbf{i}) = a^2 + b^2$

Greatest common divisor / least common multiple

- Let $a, b, c \in R$, then c is **a greatest common divisor** of a and b if
  - $c|a$ and $c|b$
  - if $d|a$ and $d|b$, then $d|c$ for all $d \in R$
- Elements $a, b \in R$ are **coprime** if and only if any gcd is invertible

Greatest common divisor / least common multiple

- Let $a, b, c \in R$, then c is **a greatest common divisor** of a and b if
  - $c|a$ and $c|b$
  - if $d|a$ and $d|b$, then $d|c$ for all $d \in R$
- Elements $a, b \in R$ are **coprime** if and only if any gcd is invertible
- Let $a, b, c \in R$, then c is **a least common multiple** of a and b if
  - $a|c$ and $b|c$
  - if $a|d$ and $b|d$, then $c|d$ for all $d \in R$
- MAGMA :
```
GCD(32, 120);  LCM(6, 8);
```
-
```
F3x<x> := PolynomialRing(GF(3));
GCD(x^3 + x^2 + 2x + 2, x^2 + x);
```
-
```
Zi<i> := GaussianIntegers();
LCM(2 + i, 2 - i);
```

Euclidean algorithm

- For a, b in Euclidean domain, denote with $r = a \bmod b$ and $q = a \operatorname{div} b$ elements such that $a = qb + r$ and $d(r) < d(b)$
- **Function** Euclid(a, b)

  1: $r_0 \leftarrow a$, $r_1 \leftarrow b$
  2: $i \leftarrow 1$
  3: **while** $r_i \neq 0$ **do**
  4: $\quad r_{i+1} \leftarrow r_{i-1} \bmod r_i, i \leftarrow i + 1$
  5: **end while**
  6: **return** r_{i-1}
- In many applications we need an explicit expression of the gcd as a linear combination of the inputs
- MAGMA : `d, x, y := XGCD(a, b); // d = a*x + b*y`

Euclidean algorithm

- Euclidean algorithm exploits the fact that

$$\gcd(a, b) = \gcd(a \bmod b, b)$$

- **Consequence**: if a is a large expression compared to b, can compute $a \bmod b$ on the fly and recover $\gcd(a, b)$ without ever writing out a

Euclidean algorithm

- Euclidean algorithm exploits the fact that

$$\gcd(a, b) = \gcd(a \bmod b, b)$$

- **Consequence**: if a is a large expression compared to b, can compute a mod b on the fly and recover $\gcd(a, b)$ without ever writing out a
- Example:
  - Given polynomial $g(x) \in \mathbb{F}_p[x]$ of degree n and p a large prime
  - To test if $g(x)$ has roots over $\mathbb{F}_p$ we need to compute

$$\gcd(x^p - x, g(x))$$

  - Using square and multiply, compute x^p mod $g(x)$ in $\mathcal{O}(\log p)$ operations in $\mathbb{F}_p[x]/(g(x))$, and then GCD algorithm using $\mathcal{O}(n^2)$ operations in $\mathbb{F}_p$

Extended Euclidean algorithm

- The extended version maintains the invariant $s_i a + t_i b = r_i$
- **Function** Extended Euclid(a, b)

1: $r_0 \leftarrow a,\ r_1 \leftarrow b,\ s_0 \leftarrow 1,\ s_1 \leftarrow 0,\ t_0 \leftarrow 0,\ t_1 \leftarrow 1$
2: $i \leftarrow 1$
3: **while** $r_i \neq 0$ **do**
4: $\quad q_i \leftarrow r_{i-1}\ div\ r_i$
5: $\quad r_{i+1} \leftarrow r_{i-1} - q_i r_i$
6: $\quad s_{i+1} \leftarrow s_{i-1} - q_i s_i$
7: $\quad t_{i+1} \leftarrow t_{i-1} - q_i t_i$
8: $\quad i \leftarrow i + 1$
9: **end while**
10: $k \leftarrow i - 1$
11: **return** r_k, s_k, t_k

Extended Euclidean algorithm - Complexity

- For $f, g \in F[x]$ with $\deg f = n \geq \deg g = m$, the Extended Euclidean Algorithm requires
  - at most $m + 1$ inversions and $2mn + O(n)$ additions and multiplications in F to compute only the r_i and q_i
  - at most $m + 1$ inversions and $6mn + O(n)$ additions and multiplications in F to compute all results
  - Typically: the degree drops exactly by 1 in every step
  - Note: specified as operations in F. For some F, e.g. $F = \mathbb{Q}$ the coefficients can grow huge.

Extended Euclidean algorithm - Complexity

- For $f, g \in F[x]$ with $\deg f = n \geq \deg g = m$, the Extended Euclidean Algorithm requires
  - at most $m + 1$ inversions and $2mn + O(n)$ additions and multiplications in F to compute only the r_i and q_i
  - at most $m + 1$ inversions and $6mn + O(n)$ additions and multiplications in F to compute all results
  - Typically: the degree drops exactly by 1 in every step
  - Note: specified as operations in F. For some F, e.g. $F = \mathbb{Q}$ the coefficients can grow huge.
- For $f, g \in \mathbb{Z}$ with $\log_{2^w} f = n \geq \log_{2^w} g = m$, with w the word-size used, the Extended Euclidean Algorithm requires $O(nm)$ word operations
  - **Exercise:** show that the number of steps in the XGCD algorithm is maximal when computing the gcd of two consecutive Fibonacci numbers

Extended Euclidean algorithm - Applications

- Constructed two quotient rings: $\mathbb{Z}/N\mathbb{Z}$ and $F[x]/(f(x))$
- Modular inverse:
  - given $a \in \mathbb{Z}$ find $b \in \mathbb{Z}$ such that $a \cdot b = 1 \bmod N$
  - given $a(x) \in F[x]$ find $b(x) \in F[x]$ such that $a(x) \cdot b(x) = 1 \bmod f(x)$
- Inverse exists iff $\gcd(a, N) = 1$ or $\gcd(a(x), f(x)) = 1$

Extended Euclidean algorithm - Applications

- Constructed two quotient rings: $\mathbb{Z}/N\mathbb{Z}$ and $F[x]/(f(x))$
- Modular inverse:
  - given $a \in \mathbb{Z}$ find $b \in \mathbb{Z}$ such that $a \cdot b = 1 \bmod N$
  - given $a(x) \in F[x]$ find $b(x) \in F[x]$ such that $a(x) \cdot b(x) = 1 \bmod f(x)$
- Inverse exists iff $\gcd(a, N) = 1$ or $\gcd(a(x), f(x)) = 1$
- Extended Euclidean algorithm results in

$$1 = sN + ta \quad \text{or} \quad 1 = s(x)f(x) + t(x)a(x)$$

- Reducing the above modulo N (or $f(x)$) shows that we can take

$$t \equiv a^{-1} \bmod N \qquad \text{and} \qquad t(x) \equiv a(x)^{-1} \bmod f(x)$$

Chinese Remainder Theorem

- Let $N = p \cdot q$ (e.g. RSA modulus see later)
- Given $a, b \in [0, \ldots, N-1]$ can compute

$$(a + b) \bmod N, (a - b) \bmod N, (a \cdot b) \bmod N$$

- **Chinese remainder theorem**: working modulo N is equivalent to working modulo p **and** modulo q
- Represent $a \bmod N$ as tuple $(a \bmod p, a \bmod q)$

Chinese Remainder Theorem - Example

- Let $n = 15 = 3 \cdot 5$, then every number between 0 and 14 can be represented by unique "coordinates" $(\bmod 3, \bmod 5)$:

	0	1	2	3	4
0	0	6	12	3	9
1	10	1	7	13	4
2	5	11	2	8	14

- All elements in the 3×5 matrix are being used.
  (note $\gcd(3, 5) = 1$)
- MAGMA : `CRT([1, 2], [3, 5]) eq 7;`

Chinese Remainder Theorem - Example

Is this always the case?
Example: Let $n = 24 = 4 \cdot 6$, then

	0	1	2	3	4	5
0	0, 12		8, 20		4, 16	
1		1, 13		9, 21		5, 17
2	6, 18		2, 14		10, 22	
3		7, 19		3, 15		11, 23

Note $\gcd(4, 6) \neq 1$

Theorem

If m_1 and m_2 are coprime, then one can replace the modulus $m_1 \cdot m_2$, by the moduli m_1, m_2.

Chinese Remainder Theorem - Two moduli case

Theorem

Suppose we want to solve a system of congruences to two **coprime** *moduli* m_1, m_2*:*

$$X \equiv a_1 \bmod m_1$$

$$X \equiv a_2 \bmod m_2$$

Then there exists a unique simultaneous solution $\tilde{x} \bmod m_1 \cdot m_2$*.*

Chinese Remainder Theorem - Two moduli case

Theorem

Suppose we want to solve a system of congruences to two **coprime** *moduli* m_1, m_2*:*

$$X \equiv a_1 \bmod m_1$$

$$X \equiv a_2 \bmod m_2$$

Then there exists a unique simultaneous solution $\tilde{x} \bmod m_1 \cdot m_2$*.*

- Theorem works for any Euclidean ring, so $\mathbb{Z}$ and $F[x]$
- MAGMA :
```
R<x> := PolynomialRing(GF(7));
CRT([x - 1, 5], [x^2 + 3, x]);
```

Chinese Remainder Theorem - Two moduli case

- Let R be an Euclidean ring, and $m_1, m_2 \in R$ are coprime

$$X \equiv a_1 \bmod m_1$$
$$X \equiv a_2 \bmod m_2$$

- First equation equivalent to:

$$\text{there exists } k \in R : \tilde{x} = a_1 + k \cdot m_1$$

Chinese Remainder Theorem - Two moduli case

- Let R be an Euclidean ring, and $m_1, m_2 \in R$ are coprime

$$X \equiv a_1 \bmod m_1$$
$$X \equiv a_2 \bmod m_2$$

- First equation equivalent to:

$$\text{there exists } k \in R : \tilde{x} = a_1 + k \cdot m_1$$

- Second equation then implies:

$$\tilde{x} = a_1 + k \cdot m_1 \equiv a_2 \bmod m_2$$

- Determines $k \bmod m_2$ as $k = (a_2 - a_1) \cdot m_1^{-1} \bmod m_2$, so solution is

$$\tilde{x} = a_1 + m_1 \cdot ((a_2 - a_1) \cdot m_1^{-1} \bmod m_2)$$

Chinese Remainder Theorem - General case

- Chinese Remainder Theorem works for r **pairwise coprime** moduli $m_1, \ldots, m_r$
- The system of equations

$$\begin{aligned} X &\equiv a_1 \bmod m_1 \\ X &\equiv a_2 \bmod m_2 \\ &\vdots \\ X &\equiv a_r \bmod m_r \end{aligned}$$

  has a unique solution $\tilde{x}$ modulo $m = m_1 \cdot m_2 \cdots m_r$.
- Closed formula is easy to derive as linear combination of the a_i

$$\tilde{x} \equiv \sum_{i=1}^{r} a_i \cdot \frac{m}{m_i} \cdot v_i \bmod m \quad \text{with } v_i \equiv \left(\frac{m}{m_i}\right)^{-1} \bmod m_i$$

Chinese Remainder Theorem - Applications

- Speed up RSA (see later): need to compute $c^d \bmod N$
  - Square and multiply, cost is $\sim 1.5 \cdot \log d$ operations modulo N
  - Using CRT in combination with FLT we compute

$$c^{d \bmod (p-1)} \bmod p \quad \text{and} \quad c^{d \bmod (q-1)} \bmod q$$

  - Note $d \bmod (p-1)$ is only half the size of d, and modulo p is roughly 4 times faster than modulo N
  - Overall: speed-up of factor 3 (taking into account overhead)

Chinese Remainder Theorem - Applications

- Speed up RSA (see later): need to compute $c^d \bmod N$
  - Square and multiply, cost is $\sim 1.5 \cdot \log d$ operations modulo N
  - Using CRT in combination with FLT we compute

$$c^{d \bmod (p-1)} \bmod p \quad \text{and} \quad c^{d \bmod (q-1)} \bmod q$$

  - Note $d \bmod (p-1)$ is only half the size of d, and modulo p is roughly 4 times faster than modulo N
  - Overall: speed-up of factor 3 (taking into account overhead)
- Computations where output is defined over $\mathbb{Z}$ or $\mathbb{Q}$ and has bounded height
  - Bounded height: coefficients a/b with $|a|, |b| < B$
  - Execute computation modulo different primes p_i such that $\prod p_i > 2B$ for $\mathbb{Z}$ (or $2B^2$ for $\mathbb{Q}$)

The RSA Cryptosystem

Rivest, Shamir, Adleman (1978): *A method for obtaining digital signatures and public key cryptosystems*

Key generation

- Find 2 primes p and q of at least 1024 bits and set $n = p \cdot q$
- Compute Euler-phi
$$\varphi(n) = (p-1)(q-1)$$
- Choose e co-prime to $\varphi(n)$ ($\neq \pm 1$) (why coprime?)
- Compute $d \equiv e^{-1} \bmod \varphi(n)$
- **public key**: (e, n)
- **private key**: (d, n) or (p, q)
- Note: can also use $\lambda(n) = \text{lcm}(p-1, q-1)$ instead of $\varphi(n)$
- Popular choice, e.g. Belgian eID cards (issued before 2020): $e = 2^{16} + 1$

The RSA Cryptosystem - Basic (insecure) version

Encryption:

- If Bob wants to encrypt a message for Alice, he does the following:
  - Obtain Alice's authentic public key (n, e).
  - Represent the message as a number $0 < m < n$.
  - Compute $c = m^e \bmod n$.
  - Send the ciphertext c to Alice.

Decryption

- Decryption: to recover m from c, Alice does the following:
  - Use the private key d to recover $m = c^d \bmod n$.

Exercise: use CRT to prove that RSA works, explain why $\lambda(n)$ can be used instead of $\varphi(n)$

RSA: Security

- Security of RSA is based on the RSA problem:
  - Given $n = p \cdot q$ a modulus
  - Public exponent e coprime to $\varphi(n)$
  - Integer $y \in \mathbb{Z}/n\mathbb{Z}$
- RSA Problem (RSAP): compute e-th root x, i.e.

$$x^e \equiv y \bmod n$$

- Note: RSA problem is easy if we know factors of n

RSA: Security

- Security of RSA is based on the RSA problem:
  - Given $n = p \cdot q$ a modulus
  - Public exponent e coprime to $\varphi(n)$
  - Integer $y \in \mathbb{Z}/n\mathbb{Z}$
- RSA Problem (RSAP): compute e-th root x, i.e.

$$x^e \equiv y \bmod n$$

- Note: RSA problem is easy if we know factors of n
- Practice: use padding scheme such as OAEP to pad the message before applying RSA encryption

RSA: Factorisation records

- Factoring n takes $\mathcal{O}(L_n[\frac{1}{3}, 1.922])$ operations

year	# digits	computing effort
1992	110	75 MY
1994	129	5000 MY
1996	130	1000 MY
1999	140	2000 MY
1999	155	8400 MY
2003	174	50000 MY
2005	200	165000 MY
2009	232	4400000 MY
2019	240	2000000 MY
2020	250	6000000 MY

- MY = Mips Year = Million Instructions per Second Year, e.g. 3GHz for one year gives 3000MY
- 250 digits = 829 bits — 1024 bit factorisation roughly 200 times harder than 829 bits

Recap

- Modular arithmetic in $\mathbb{Z}/N\mathbb{Z}$ and $F[x]/(f(x))$
- Constructed finite fields $\mathbb{F}_{p^n}$ for each prime p and $n \in \mathbb{N}_0$
- Inverses via XGCD algorithm, and possibly FLT
- Fast exponentiation using square-multiply
- CRT to simplify / speed-up computations
- Additional background information can be found in:
  `https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch2.pdf`