Chapter 5: Number Theory and Algebra

Lectures 25

Factorisation methods

Trial factorisation

Fermat Factorisation

Pollard's ρ Method

Quadratic Sieve Method

Shor's Algorithm

others...

Trial factorisation

Input: an integer n
Output: the factors of n

- Trial divide n by primes up to $\sqrt{n}$ until the smallest factor a is found.
- If $a > 1$, then repeat with $\frac{n}{a}$.

This is good for small n but very slow in general.

Fermat Factorisation

Input: an odd integer n
Output: a two-factorization of n

- For $t = \lceil \sqrt{n} \rceil, \ldots, n$:
  - If $s^2 = t^2 - n$ is square, then return $n = ab = (t + s)(t - s)$.

Note that

- we can write $n = ab$ with $a \geq b \geq 1$ if and only if

$$n = t^2 - s^2 = (t + s)(t - s) \quad \text{where} \quad t = \frac{1}{2}(a + b) \quad \text{and} \quad s = \frac{1}{2}(a - b)$$

- Checking whether a number is square is algorithmically quick and easy.
- Calculating each term $s^2 = t^2 - n$ is iteratively linear:

$$(t + 1)^2 - n = (t^2 - n) + (2t + 1)$$

Fermat Factorisation

Input: an odd integer n
Output: a two-factorization of n

- For $t = \lceil \sqrt{n} \rceil, \ldots, n$:
  - If $s^2 = t^2 - n$ is square, then return $n = ab = (t+s)(t-s)$.

Example

Find factor (pair) of $n = 9869$.

$\lceil \sqrt{n} \rceil \approx \lceil 99.3 \rceil =$

t	$2t+1$	$s^2 = t^2 - n$	$s \in \mathbb{Z}$?
100	201	131	No
101	203	332	No
102	205	535	No
103	207	740	No
104	209	947	No
105		1156	Yes: $s = \sqrt{1156} = 34$

We find that $a = s + t = 139$ and $b = t - s = 71$, so $\boxed{n = ab = 71 \times 139}$

Pollard's ρ Method

Input: integers n, x_0
Output: possibly a factor d of n

- Iterate $x_{i+1} \equiv f(x_i) \pmod{n}$ where $f(x) = x^2 - 1$
  until $d = \gcd(x_i - x_{2i}, n) = n$; return failure
  or $1 < d < n$; return d

Note that

- We can replace $f : \mathbb{Z} \to \mathbb{Z}$ by many other functions that satisfy $x_i \equiv x_{2i} \pmod{p}$ for many possible factors p of n.
- We can also replace x_{2i} by other x_j.
- Calculating d can be done fairly quickly with the Euclidean Algorithm.
- This algorithm pseudo-randomly chooses pairs x_i, x_{2i}
- The algorithm has expected time $O(\sqrt{p})$ to find a factor p.

Example

Find a factor of $n = 91643$ if possible.

Choose $x_0 = 3$, say, and calculate $x_{i+1} = x_i^2 - 1 \pmod{91654}$:

$$
\begin{array}{lll}
x_0 \equiv 3 & & \gcd(x_{2i} - x_i, n) \\
x_1 \equiv 8 & & \\
x_2 \equiv 63 & x_2 - x_1 \equiv 55 & 1 \\
x_3 \equiv 3968 & & \\
x_4 \equiv 74070 & x_4 - x_2 \equiv 74007 & 1 \\
x_5 \equiv 65061 & & \\
x_6 \equiv 35193 & x_6 - x_3 \equiv 31225 & 1 \\
x_7 \equiv 83746 & & \\
x_8 \equiv 45368 & x_8 - x_4 \equiv 62941 & 113
\end{array}
$$

We have found a factor $d = 113$.

It is a prime and so is $\dfrac{n}{d} = 811$, so $n = 91643 = 113 \times 811$.

Quadratic Sieve Method

- Combines Fermat Factorisation with Pollard's ρ Method
- It tries to cleverly find pairs s, t so that $t^2 \equiv s^2 \pmod{n}$;
  then $n \mid (t^2 - s^2) = (t - s)(t + s)$;
  calculating $\gcd(n, t \pm s)$ provides a factor of n.
- This is a very fast method.

Shor's Algorithm

- This algorithm is designed for use on quantum computers.
- It works as follows:
  - Find some a so that $\operatorname{ord}_n(a) = 2k$ for some integer k.
  - Then $a^{2k} \equiv 1 \pmod{n}$, so

$$n \mid (a^{2k} - 1) = (a^k - 1)(a^k + 1)$$

  - Calculate $\gcd(n, a^k \pm 1)$ to find factors.

The quantum computing is used for finding a quickly.

So far, quantum computers are very primitive
- but Prof. Michelle Simmons' team (UNSW) is making great advances!

Random Number Generation

- Middle Squares Method
- Linear Congruential
- Polynomial Congruential and LFSR
- N-LFSR
- Cryptographic generators
- Multiplexing of sequences
- others...

These generate pseudo-random numbers deterministically via algorithms.

Truly random processes, like observing nuclear decay, provide random numbers non-deterministically.

Middle Squares Method

Input: An integer n and an integer seed x_0
Output: Pseudo-random sequences of n digits.

- Iterate:
  - $x_{i+1} = x_i^2$
  - Add leading 0s so that x_{i+1} has $2n$ digits.
  - Crop x_{i+1} to middle n digits.

This method is easy to use - but is slow, and has short periodicity.

Middle Squares Method

Input: An integer n and an integer seed x_0
Output: Pseudo-random sequences of n digits.

- Iterate:
  - $x_{i+1} = x_i^2$
  - Add leading 0s so that x_{i+1} has $2n$ digits.
  - Crop x_{i+1} to middle n digits.

Example

Let $n = 4$ and let $x_0 = 2100$.

Then

$$\begin{aligned} x_0 &= 2100 \\ x_1 &= 04410000 \\ x_2 &= 16810000 \\ x_3 &= 65610000 \\ x_4 &= 37210000 \\ x_5 &= x_1 \end{aligned}$$

Linear Congruential

Input: Integers a, b, m and an integer seed x_0
Output: Pseudo-random numbers x_i

- Iterate:
  - $x_{i+1} \equiv ax_i + b \pmod{m}$

This method is easy to use and is relatively useful.

- Maple uses this method with
$$\begin{aligned} a &= 427419669081 \\ b &= 0 \\ m &= 999999999989 \end{aligned}$$
  and $x_0 = 1$ or truly random seeds x_0, like the date and time, say.
- Unfortunately, this method cannot be used for cryptography since a, b can often be determined from m and any three x_{i-1}, x_i, x_{i+1}.

Polynomial Congruential and LFSR

Input: A prime p and $a_0, \dots, a_{n-1} \in \mathbb{Z}_p$ and integer seeds $x_0, \dots, x_{n-1}$
Output: Pseudo-random numbers x_i

- Iterate:
  - $x_{i+n} \equiv a_{n-1} x_{i+n-1} + \cdots + a_0 x_i \pmod{p}$

This method is easy to use and has maximal possible period length ($p^n - 1$) when the recursion's characteristic polynomial

$$f(r) = r^n - a_{n-1} r^{n-1} - \cdots - a_0$$

is primitive over $\mathbb{Z}_p$.

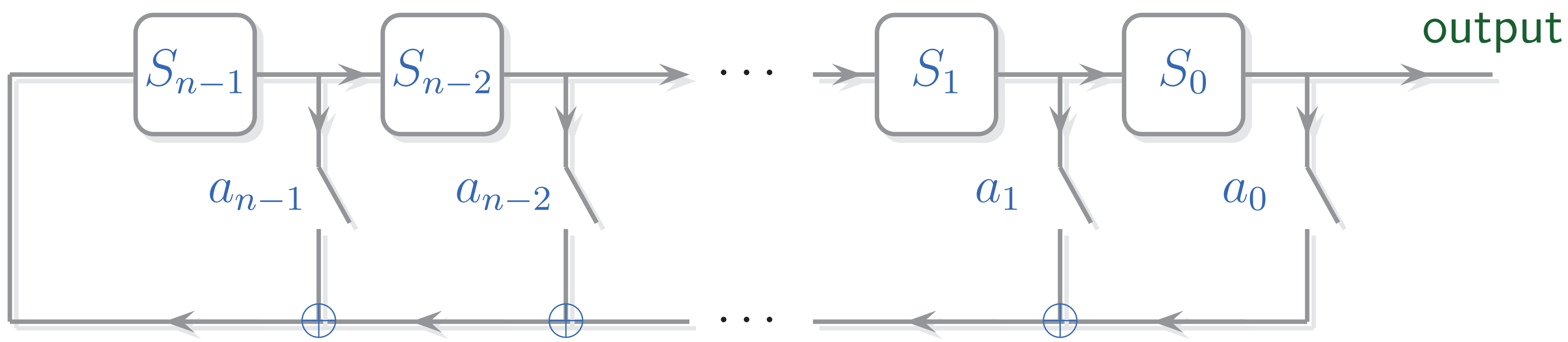
- Unfortunately, like the linear congruential method, this method cannot be used for cryptography since $a_0, \dots, a_{n-1}$ can often be determined from p and any $2n + 1$ consecutive x_is.

Polynomial Congruential and LFSR

Input: A prime p and $a_0, \dots, a_{n-1} \in \mathbb{Z}_p$ and integer seeds $x_0, \dots, x_{n-1}$
Output: Pseudo-random numbers x_i

- Iterate:
  - $x_{i+n} \equiv a_{n-1}x_{i+n-1} + \cdots + a_0 x_i \pmod{p}$

For $p = 2$, one often uses Linear Feedback Shift Registers (LFSR).

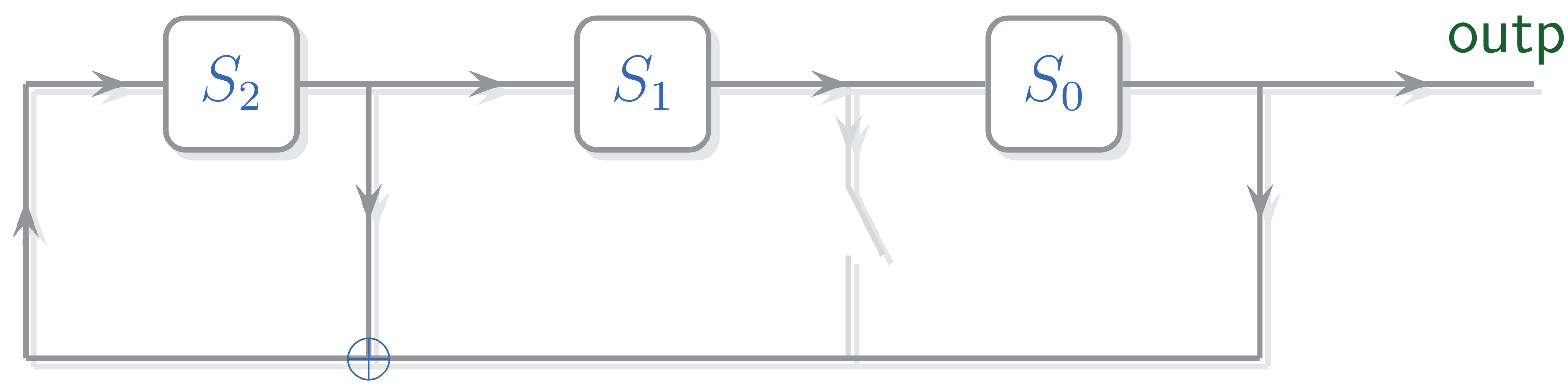


- $s_{n-1}, \dots, s_0$ are switch registers, initially $x_{n-1}, \dots, x_0$.
- Switch i is off if $a_i = 0$ and on if $a_i = 1$.
- $\oplus$ indicates binary addition (XOR; super-fast)

Example

$f(x) = x^3 + x^2 + 1$ is primitive over $\mathbb{Z}_2$ and represents the recurrence

$$x_{i+3} = x_{i+2} + x_i = a_2 x_{i+2} + a_1 x_{i+1} + a_0 x_i \quad \text{for } (a_2, a_1, a_0) = (1, 0, 1)$$



Set initial values $(x_2, x_1, x_0) = (0, 0, 1)$:

S_2	S_1	S_0	output
0	0	1	
1	0	0	1
1	1	0	0
1	1	1	0
0	1	1	1
1	0	1	1
0	1	0	1
0	0	1	0

Non-linear Feedback Shift Registers (N-FLSR)

Input: Integer seeds $x_0, \ldots, x_{n-1}$ and a non-linear function $f : \mathbb{Z}^n \to \mathbb{Z}$
Output: Pseudo-random numbers x_i

- Iterate:
  - $x_{i+1} = f(x_i, \ldots, x_{i+n-1})$

Not much is known about these in general.
One efficient N-FLSR is given by

$$x_0 \equiv 2 \pmod{4}$$
$$x_{i+1} \equiv x_i(x_i + 1) \pmod{2^k}, \ k > 2$$