Arytmetyka modularna

Jakub Bronowski

Seminarium WPAiT, 2025

Agenda

- Czym jest arytmetyka modularna?
- Podstawowe własności kongruencji
- Twierdzenie Fermata (małe) i Eulera
- Krótko o logarytmie dyskretnym
- Praktyczne użycie arytmetyki modularnej
  - generator liczb pseudolosowych Blum Blum Shub
  - kody kontrolne
  - szyfrowanie
  - wymiana kluczy
  - kryptograficznie bezpieczne przechowywanie danych logowania

Uwaga

Zapis $\mathbb{P}$ oznacza zbiór liczb pierwszych.

Czym jest arytmetyka modularna?

Definicja

Arytmetyka modularna (również: arytmetyka kongruencji, arytmetyka reszt) - system liczb całkowitych, w którym liczby "zawijają się"po osiągnięciu pewnej wartości (moduł). W arytmetyce modularnej wszystkie możliwe wyniki operacji arytmetycznych mieszczą się w zbiorze $\{0, 1, \ldots, M-1\}$, gdzie M - moduł.

Przykłady:

- dla klasycznego zegara: $M = 12$,
- dla minut i sekund: $M = 60$,
- dla komputerów arch. 64-bit: $M = 2^{64}$

Współczesną arytmetyką modularną sformalizował Carl Friedrich Gauss w dziele *Disquisitiones Arithmeticae.*

Kongruencja modulo

Definicja

Kongruencja - relacja równoważności dwóch liczb.

Kongruencja modulo M (również: przystawanie modulo M) zachodzi, gdy dla 2 liczb całkowitych a i b spełniony jest warunek:

$$(a - b) = kM$$

gdzie k - liczba całkowita.
Zapisujemy wedy

$$a \equiv b \pmod{M}$$

.
Przykłady:

- $38 \equiv 14 \pmod{12}$, ponieważ $(38 - 14) = 24 = 2 \cdot 12$,
- $49 \equiv 0 \pmod{7}$, ponieważ $(49 - 0) = 49 = 7 \cdot 7$,

Podstawowe własności kongruencji (1)

Definicja

$a \equiv b \pmod{M} \Leftrightarrow (a - b) = kM$, gdzie k - liczba całkowita.

1. $a \equiv a \pmod{M}$
2. Jeśli $a \equiv b \pmod{M}$, to $b \equiv a \pmod{M}$
3. Jeśli $a \equiv b \pmod{M}$ i $b \equiv c \pmod{M}$, to $a \equiv c \pmod{M}$
4. Jeśli $a \equiv b \pmod{M}$ i $c \equiv d \pmod{M}$, to $a + c \equiv b + d \pmod{M}$ oraz $a - c \equiv b - d \pmod{M}$
5. Jeśli $a \equiv b \pmod{M}$ i $c \equiv d \pmod{M}$, to $a \cdot c \equiv b \cdot d \pmod{M}$
6. Jeśli $a \equiv b \pmod{M}$, to dla dowolnego c zachodzi $a + c \equiv b + c \pmod{M}$
7. Jeśli $a \equiv b \pmod{M}$, to dla dowolnego c zachodzi $a \cdot c \equiv b \cdot c \pmod{M}$
8. Jeśli $a \equiv b \pmod{M}$, to dla każdego $k \in \mathbb{N}$ mamy $a^k \equiv b^k \pmod{M}$

Podstawowe własności kongruencji (2)

Twierdzenie

Nie zawsze da się wykonać dzielenie w arytmetyce modularnej.

Kiedy można wykonać dzielenie?

1. $a \cdot n \equiv b \cdot n \pmod{n \cdot M} \Leftrightarrow a \equiv b \pmod{M}$
2. Jeśli $a \cdot c \equiv b \cdot c \pmod{M}$ oraz $\mathrm{NWD}(c, M) = 1$, to $a \equiv b \pmod{M}$.

Małe Twierdzenie Fermata (MTF)

Twierdzenie

Jeżeli $p \in \mathbb{P}$ oraz $a \in \mathbb{Z}$ to:

$$a^p - a \equiv 0 \pmod{p}$$

Dodatkowo, przy założeniu, że $\mathrm{NWD}(a, p) = 1$, mamy:

$$a^{p-1} - 1 \equiv 0 \pmod{p}$$

lub równoważnie:

$$a^{p-1} \equiv 1 \pmod{p}$$

Ciekawostka:
Fermat nie podał dowodu swojego twierdzenia. Poprawności dowiódł Euler.

Twierdzenie Eulera i tocjent

Definicja

Tocjent $\varphi(n)$ - funkcja przypisująca ilość liczb względnie pierwszych z n w zbiorze $\{1, 2, \ldots, n\}$.
Na przykład $\varphi(9) = 6$, ponieważ liczby względnie pierwsze z 9 to: $1, 2, 4, 5, 7, 8$.

Twierdzenie

Dla $a, n \in \mathbb{Z}$ oraz $NWD(a, n) = 1$ zachodzi:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

gdzie $\varphi(n)$ - tocjent (również: funkcja Eulera).

Twierdzenie Eulera a MTF

Warto zauważyć, że jeśli potrafimy szybko wyliczyć $\varphi(p)$, to drastycznie zmiejszamy wykładnik w MTF.
Przykład:

$$\text{Niech } a = 7,\ n = 10, \text{ zauważamy } \mathrm{NWD}(7, 10) = 1$$

$$\varphi(10) = 4$$

$$\text{Z Tw. Eulera: } 7^{\varphi(10)} = 7^4 \equiv 1 \pmod{10}$$

$$\text{Policzmy } 7^{3333} \mod 10$$

$$7^{3333} \equiv 7^{833 \cdot 4 + 1} \equiv (7^4)^{833} \cdot 7^1 \equiv 1^{833} \cdot 7^1 \equiv 7 \pmod{10}$$

Zauważmy, że potęgowanie modulo ma złożoność $O(log(n))$, gdzie n - wykładnik.
Wykonując nieduży nakład pracy znacznie zmniejszamy wykładnik.

Krótko o logarytmie dyskretnym (1)

Definicja

Logarytm dyskretny to liczba całkowita x spełniająca równanie $a^x \equiv b \pmod{M}$ dla danych liczb całkowitych a, b i M.

Definicja

Logarytm dyskretny nie zawsze istnieje. Nie ma prostego warunku pozwalającego określić, czy logarytm dyskretny istnieje.

Przykład:

$$2^x \equiv 3 \pmod{7} \text{ - nie istnieje}$$

$$5^x \equiv 3 \pmod{7} \text{ - istnieje, } x = 5 \text{ lub } 11 \text{ lub } 17 \text{ lub } \dots$$

Krótko o logarytmie dyskretnym (2)

Siła logarytmu dyskretnego objawia się w wielkiej trudności jego rozwiązania, dlatego też wykorzystuje się go w kryptografii.
Sposoby rozwiązania logarytmu dyskretnego:

1. Pełen przegląd - $O(nlogn)$
2. algorytm Baby-step giant-step - $O(\sqrt{M})$, gdzie M - modulo

Praktyczne zastosowania arytmetyki modularnej

1. generator liczb pseudolosowych: Blum Blum Shub
2. kody kontrolne: Luhn, CRC
3. szyfrowanie: RSA
4. wymiana kluczy: Diffie-Hellman
5. Secure Remote Password (SRP v. 6a)

Generator liczb pseudolosowych - Blum Blum Shub

Definicja

Blum Blum Shub (BBS) - jeden z najbezpieczniejszych kryptograficznie generatorów liczb pseudolosowych. Definiowany jako:

$$x_{n+1} = x_n^2 \bmod M$$

gdzie $M = p \cdot q$, oraz p, q takie, że $p, q \in \mathbb{P} \wedge p \equiv q \equiv 3 \pmod 4$.
Liczba początkowa x_0 powinna być względnie pierwsza z M, może być losowa.
Zazwyczaj jako wyjście generatora przyjmuje się najmłodszy bit x_n, ale można też brać więcej bitów.

Generowanie liczb pseudolosowych w ten sposób jest dość powolne, ale w oparciu o tw. Eulera możemy przekształcić wzór i przyspieszyć obliczenia następnych wyrazów:

$$x_i = ({x_0}^{2^i \bmod \mathrm{NWW}(p-1, q-1)}) \pmod M)$$

Blum Blum Shub (Przykład)

Przykład

Niech $p = 7$, $q = 11$, wtedy $M = 77$. Wybieramy $x_0 = 20$ ($\mathrm{NWD}(20, 77) = 1$).
Obliczamy kolejne wartości:

- $x_1 = 20^2 \bmod 77 = 400 \bmod 77 = 15$
- $x_2 = 15^2 \bmod 77 = 225 \bmod 77 = 71$
- $x_3 = 71^2 \bmod 77 = 5041 \bmod 77 = 35$
- $x_4 = 35^2 \bmod 77 = 1225 \bmod 77 = 74$

Najmłodsze bity kolejnych wartości to: $x_1 = 15_{10}(1111_2)$, $x_2 = 71_{10}(1000111_2)$, $x_3 = 35_{10}(100011_2)$, $x_4 = 74_{10}(1001010_2)$.
Wygenerowana wartość: $1110_2 = 14_{10}$

Kody kontrolne - algorytm Luhna (1)

Definicja

Algorytm Luhna — prosty algorytm obliczania cyfry kontrolnej dla numerów identyfikacyjnych. Metodę stworzył naukowiec IBM Hans Peter Luhn w celu wykrywania przypadkowych błędów w zabezpieczonych numerach.
Struktura numeru to $XXXXXXXC$, gdzie $XXXXXXX$ to dowolnie długi numer identyfikacyjny, a C to cyfra kontrolna.

Typy numerów wykorzystujące algorytm Luhna:

- karty płatnicze
- IMEI (numer seryjny telefonu komórkowego)
- PESEL
- numer ubezpieczenia / rachunku bankowego
- Kody kreskowe UPC, EAN

Opisane w standardzie ISO/IEC 7812-1.

Kody kontrolne - algorytm Luhna (2)

Algorytm generowania cyfry kontrolnej:

1. Przechodząc po cyfrach liczby od prawej do lewej, podwój każdą drugą cyfrę.
2. Jeśli podwojenie cyfry dało wartość > 9, odejmij 9 od niej (tożamo: zsumuj jej cyfry).
3. Wyznacz s - suma wszystkich cyfr (przemnożonych i nieprzemnożonych).
4. Wyznacz cyfrę kontrolną jako $10 - (s \bmod 10)$, gdzie s to suma z kroku 3.
5. Doklej cyfrę kontrolną do na koniec numeru.

—

Algorytm weryfikacji cyfry kontrolnej:

1. Wykonaj kroki 1-4 z algorytmu generowania cyfry kontrolnej, uwzględniając cyfrę kontrolną.
2. Jeśli $s \bmod 10 = 0$, liczba jest poprawna. W przeciwnym przypadku nie jest.

Kody kontrolne - algorytm Luhna (3)

Niedoskonałości algorytmu Luhna:

- Nie wykrywa zamian na niesąsiednich pozycjach.
- Nie wykrywa zamian: $90 \leftrightarrow 09$, $44 \leftrightarrow 77$, $22 \leftrightarrow 55$, $33 \leftrightarrow 66$.

Zalety algorytmu Luhna:

- Prosty i szybki w implementacji.
- Można dodawać padding, o ile jest prefiksem złożonym z 0.

Kody kontrolne - CRC (1)

Definicja

CRC (Cyclic Redundancy Check) — system sum kontrolnych wykorzystywany do wykrywania przypadkowych błędów pojawiających się podczas przesyłania lub magazynowania danych binarnych.

Niech

$M(x)$ - wielomian wiadomości, $G(x)$ - dzielnik (generator), wielomian stopnia r.

Obliczanie CRC:

1. Dopisz do bitów wiadomości r zer (tożsame przemnożeniu przez x^r), otrzymujemy $M(x)\,x^r$.
2. Podziel $M(x)\,x^r$ przez $G(x)$ wykonując dzielenie modulo 2 (XOR):
$$M(x)\,x^r = Q(x)\,G(x) + R(x), \quad \deg R(x) < r.$$
3. CRC to reszta $R(x)$. Do wiadomości dołączamy bity odpowiadające $R(x)$. Wiadomość jest podzielna przez $G(x)$ i jest postaci:
$$T(x) = M(x)\,x^r + R(x),$$

Kody kontrolne - CRC (2)

Weryfikacja CRC:

1. Odbiorca dzieli otrzymane $T(x)$ przez $G(x)$ (dzielenie modulo 2).
2. Jeżeli reszta jest równa zero, ramka jest zgodna z CRC (brak wykrytych błędów); w przeciwnym razie wykryto błąd transmisji.

Przykład:

- Wiadomość: 11010011101100
- Generator: 1011 (czyli $G(x) = x^3 + x + 1$, stopień $r = 3$)

Po dzieleniu 11010011101100 000 przez 1011 otrzymujemy resztę 100.
Transmitowana ramka: 11010011101100 100 = 11010011101100100.
Odbiorca dzieli 11010011101100100 przez 1011 — reszta wynosi 0, dane poprawne.
Wykorzystanie: kontrola poprawności transmisji m.in. w Ethernet, ZIP, PNG, MPEG-2, SATA, SCSI, USB, PCI, Bluetooth.

RSA

1. **Generowanie kluczy**
   - Wybierz dwa duże (≥ 2048 bit) różne liczby pierwsze p, q.
   - Oblicz $n = p \cdot q$.
   - Oblicz $\varphi(n) = (p-1)(q-1)$.
   - Wybierz e takie, że $1 < e < \varphi(n)$ i $\mathrm{nwd}(e, \varphi(n)) = 1$.
     Często za e wybiera się $2^{16} + 1$ ze względu na efektywność.
   - Oblicz d jako odwrotność e modulo $\varphi(n)$: $d \equiv e^{-1} \pmod{\varphi(n)}$.

   Publiczny klucz: (n, e). Prywatny klucz: d, n.
2. **Szyfrowanie**
   $$c \equiv m^e \pmod{n}.$$
3. **Deszyfrowanie**
   $$m \equiv c^d \pmod{n}.$$

Krótki przykład: $p = 61,\ q = 53 \Rightarrow n = 3233,\ \varphi(n) = 3120$. Niech $e = 17$. Wtedy $d = 2753$. Dla $m = 65$: $c = 65^{17} \bmod 3233 = 2790$. Deszyfrując: $2790^{2753} \bmod 3233 = 65$.

Bezpieczeństwo opiera się na trudności faktoryzacji dużego $n = p \cdot q$.

Wymiana kluczy - Diffie-Hellman (1)

Definicja

Protokół umożliwiający dwóm stronom wynegocjowanie wspólnego tajnego klucza przy użyciu publicznych parametrów, bez wcześniejszego dzielenia sekretu.

Algorytm:

1. Wybierz $p \in \mathbb{P} \wedge p \geq 2048$ bit oraz g takie, że jest ono pierwiastkiem pierwotnym modulo p.
2. Strona pierwsza wybiera liczbę a, oblicza $A \equiv g^a \pmod p$ i wysyła A do strony drugiej.
3. Strona druga wybiera liczbę b, oblicza $B \equiv g^b \pmod p$ i wysyła B do strony pierwszej.
4. Obie strony obliczają:

$$s_A \equiv B^a \pmod p, \qquad s_B \equiv A^b \pmod p,$$

przy czym $s_A = s_B \equiv g^{ab} \pmod p$. Jest to wspólny sekret.

Pierwiastek pierotny*

Definicja

Niech $p \in \mathbb{P}$. Element g nazywamy pierwiastkiem pierwotnym modulo p jeśli zachodzi:

$$\{g^1, g^2, \dots, g^{p-1}\} \equiv \{1, 2, \dots, p-1\} \pmod{p},$$

czyli potęgi g generują wszystkie niezerowe reszty modulo p.

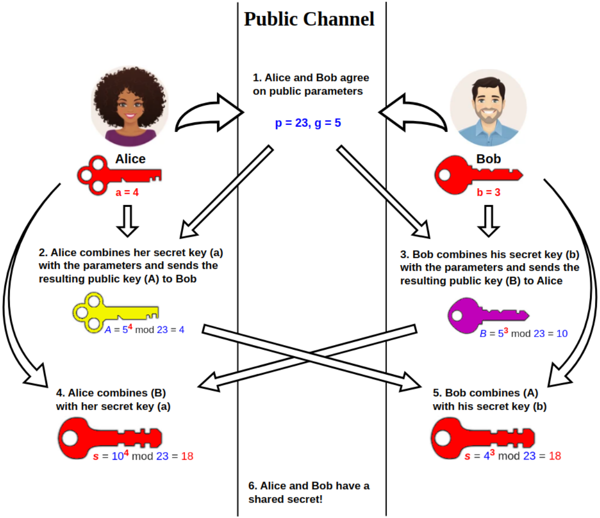
Twierdzenie

Dla każdej liczby pierwszej p istnieje pierwiastek pierwotny modulo p.

Przykład

Dla $p = 7$ elementy 3 oraz 5 są pierwiastkami pierwotnymi, bo kolejne potęgi 3 (lub 5) dają wszystkie reszty $1, 2, \dots, 6$ modulo 7.

Wymiana kluczy - Diffie-Hellman (2)



Secure Remote Password v6.a (SRP) (1)

Definicja

Protokół umożliwiający bezpieczne uwierzytelnienie jednej ze stron w drugim systemie.

Założenia początkowe:

- $q, N \in \mathbb{P} \land N \geq 1000 \text{ bit} \land N = 2 \cdot q + 1$ oraz N **jest jawne**,
- $g \mod N$ stanowiący generator grupy multiplikatywnej modulo N oraz g **jest jawne**,
- dysponujemy bezpieczną funkcja skrótu $hash(\cdot)$,
- $k = hash(N, g)$,
- posiadamy dobre źródło losowości.

Uwaga: symbol $\oplus$ oznacza konkatenację.

Secure Remote Password v6.a (SRP) (2)

Tworzenie konta:

1. Klient wybiera login I oraz hasło P, generuje losową sól s.
2. Klient oblicza $x = hash(s \oplus hash(I \oplus : \oplus P))$ (zgodnie z RFC2945).
3. Klient oblicza $v = g^x \bmod N$ i wysyła do serwera trójkę (I, s, v). Należy też usunąć x (tożsame z jawnym hasłem).
4. Serwer zapisuje trójkę (I, s, v) w swojej bazie danych użytkowników.

Secure Remote Password v6.a (SRP) (3)

Weryfikacja:

1. Klient wysyła do serwera login I oraz $A = g^a \mod N$ gdzie a - losowe,
2. Serwer wysyła do klienta sól s oraz $B = (k \cdot v + g^b \mod N)$ gdzie b - losowe.
3. Obie strony obliczają wspólny sekret: $u = hash(A \oplus B)$.
4. Klient oblicza: $S_C = (B - k \cdot g^x)^{(a+u \cdot x)} \mod N$.
5. Serwer oblicza: $S_S = (A \cdot v^u)^b \mod N$.
6. Obie strony obliczają klucz sesji: $K_C = hash(S_C)$, $K_S = hash(S_S)$
7. Klient wysyła do serwera dowód
   $M_1 = hash((hash(N) \text{ XOR } hash(g)) \oplus hash(I) \oplus s \oplus A \oplus B \oplus K_C)$.
8. Serwer weryfikuje M_1 i wysyła do klienta dowód $M_2 = hash(A \oplus M_1 \oplus K_S)$.
9. Jeśli $M_1 = M_2$, weryfikacja przebiegła poprawnie.

Safeguards:

- Przerywamy, jeśli:
  - $B \equiv 0 \pmod N$ lub $u \equiv 0 \pmod N$ (klient),
  - $A \equiv 0 \pmod N$ (serwer).
- Klient **pierwszy** wysyła swój dowód. Serwer ujawnia swój dowód, tylko gdy oba dowody zgadzają się.

Źródła I

- https://mathworld.wolfram.com/ModularArithmetic.html
- https://simple.wikipedia.org/wiki/Modular_arithmetic
- https://archive.org/details/disquisitionesa00gaus/page/X/mode/2up
- https://view.fis.agh.edu.pl/staff/lenda/number_theory/A31.pdf
- https://pl.wikipedia.org/wiki/MaC582e_twierdzenie_Fermata
- https://deltami.edu.pl/2017/04/male-twierdzenie-fermata/0A
- https://pl.wikipedia.org/wiki/Funkcja_CF86
- https://en.wikipedia.org/wiki/Euler27s_theorem
- https://cp-algorithms.com/algebra/discrete-log.html
- https://fizyka.umk.pl/~gniewko/didaktiki/MD2013-2014/wykC582ad9.pdf

Źródła II

- https://en.wikipedia.org/wiki/Blum_Blum_Shub
- https://asecuritysite.com/encryption/blum
- https://www.researchgate.net/profile/Lenore-Blum/publication/221354947_Comparison_of_Two_Pseudo-Random_Number_Generators/links/00463531f2e378e090000000/Comparison-of-Two-Pseudo-Random-Number-Generators.pdf
- https://en.wikipedia.org/wiki/Luhn_algorithm
- http://www.algorytm.org/sumy-kontrolne/algorytm-luhna-mod-10.html
- https://en.wikipedia.org/wiki/Cyclic_redundancy_check
- https://pl.wikipedia.org/wiki/Cykliczny_kod_nadmiarowy
- https://ucgosu.pl/2017/01/jak-dziala-crc/
- https://www.geeksforgeeks.org/dsa/modulo-2-binary-division/

Źródła III

- https://en.wikipedia.org/wiki/RSA_cryptosystem
- https://eduinf.waw.pl/inf/utils/010_2010/0219.php
- https://en.wikipedia.org/wiki/DiffieE28093Hellman_key_exchange
- https://www.math.brown.edu/johsilve/MathCrypto/SampleSections.pdf
- https://www.geeksforgeeks.org/dsa/primitive-root-of-a-prime-number-n-modulo-n/
- https://en.wikipedia.com/wiki/DiffieE28093Hellman_key_exchange
- https://zaufanatrzeciastrona.pl/post/jak-blizzard-zabezpiecza-wasze-hasla-czyli-protokol-srp-i-czy-mozna-go
- http://srp.stanford.edu/design.html
- https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol

Źródła IV

- https://datatracker.ietf.org/doc/html/rfc2945
- https://pl.wikipedia.org/wiki/Grupa_cykliczna