Kryptografia i bezpieczeństwo danych - elementy teorii liczb

Sławomir Samolej
ssamolej.kia.prz.edu.pl
ssamolej@prz.edu.pl

W poprzednim wykładzie wspomnieliśmy, że teoria liczb jest stosowana w protokołach wymiany kluczy. Ten wykład służy wprowadzeniu podstawowych zagadnień z teorii liczb, aby można było łatwiej wyjaśnić wprowadzane później protokoły wymiany.

Tło

Teoria liczb jest stosowana do konstruowania:

- Protokołów wymiany kluczy
- Podpisów elektronicznych
- Szyfrowania z kluczem publicznym

W czasie tego wykładu: przyspieszony kurs dotyczący powiązanych zagadnień

Więcej informacji: można przeczytać części książki Shoup'a wskazanej w spisie literatury na końcu prezentacji

2

Teoria liczb stosowana jest w wielu przypadkach. W kryptografii najczęściej w do opracowywania protokołów wymiany kluczy, podpisów elektronicznych oraz do szyfrowania z kluczem publicznym. Zanim zostaną wprowadzone powyższe zagadnienia warto zapoznać się z najważniejszymi obszarami teorii liczb. Wiedzę podaną na wykładzie można ugruntować np. na podstawie darmowej książki, do której odwołanie znajdzie się na końcowym slajdzie prezentacji.

Notacja

Od tej chwili:

- N oznacza dodatnią liczbę całkowitą integer.
- p oznacza liczbę pierwszą.

Notacja: $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$

Można wykonywać dodawanie i mnożenie modulo N.

3

Od tej chwili przyjmiemy następującą notację. N oznacza liczbę całkowitą dodatnią, a p oznacza liczbę pierwszą. Literka „Z” z podwójną ukośną linią i indeksem dolnym „N” oznacza zbiór {0, 1, 2, …, N-1}. Na takim zbiorze liczb można dokonywać dodawania i mnożenia modulo N.

Arytmetyka modularna

Przykłady: niech N = 12

9 + 8 = 5 w $\mathbb{Z}_{12}$

5 × 7 = 11 w $\mathbb{Z}_{12}$

5 − 7 = 10 w $\mathbb{Z}_{12}$

Arytmetyka w $\mathbb{Z}_N$ działa, np. x·(y+z) = x·y + x·z w $\mathbb{Z}_N$

4

Arytmetyka modulo daje zawsze wyniki w zakresie ustalonego zbioru wartości od 0 do N-1.
Jeśli suma jest mniejsza niż N, to wygląda tak samo, jak w zwyczajnej arytmetyce. Jeśli suma jest większa od N, to należy wziąć z niej resztę z dzielenia przez N. Mnożenie daje takie same rezultaty jak normalnie, jeśli wynik jest mniejszy od N, w przeciwnym wypadku wynikiem z mnożenia jest reszta z dzielenia zwyczajnego mnożenia przez N. Odejmowanie również „wygląda normalnie” dopóki wynik jest dodatni i większy od 0. Jeśli wynik jest mniejszy od 0 to dodaje się do niego N, aby otrzymać wartość dodatnią. Prawa arytmetyki znane z „tradycyjnej” arytmetyki nadal działają, chociaż operacje arytmetyczne zostały zmodyfikowane, np. rozdzielność mnożenia względem dodawania.

Największy wspólny dzielnik

Def: Dla całkowitych. x,y: **nwd(x, y)** jest największym wspólnym dzielnikiem x i y (ang. gcd. – greatest common divisor)

Przykład: nwd(12, 18) = 6

$2 \times 12 - 1 \times 18 = 6$

Fakt: dla wszystkich liczb całkowitych x, y istnieją takie a,b, że:

a·x + b·y = nwd(x,y)

a i b mogą być znalezione z zastosowaniem rozszerzonego alg. Euklidesa.

Jeśli nwd(x,y)=1 mówimy, że x i y są względnie pierwsze

np: $nwd(3,5)=1$

5

Koncepcja największego wspólnego dzielnika dwóch liczb jest znana. Warto zwrócić uwagę na twierdzenie mówiące, że jeśli mamy dwie liczby całkowite x i y , to można znaleźć takie dwie liczby a, b, że a*x + b*y = nwd(x, y). Inaczej nwd można wyrazić w postaci kombinacji liniowej liczb, dla których go poszukujemy. Okazuje się, że znalezienie tej pary liczb a, b jest możliwe z zastosowaniem efektywnego algorytmu, nazywanego rozszerzonym algorytmem Euklidesa. Jeśli nwd(x, y) = 1, to takie dwie liczby całkowite nazywamy względnie pierwszymi.

Odwrotność modularna (1)

W przypadku liczb wymiernych, odwrotność 2 wynosi ½ .
Jak jest w przypadku liczb $\mathbb{Z}_N$?

Def: **Odwrotność** x w $\mathbb{Z}_N$ jest element y należący do $\mathbb{Z}_N$

taki, że $x \cdot y = 1 \cup \mathbb{Z}_N$

y jest oznaczany jako x^{-1} .

Przykład: Niech N będzie liczbą nieparzystą. Odwrotnością 2 w $\mathbb{Z}_N$ jest: $\frac{N+1}{2}$

bo: $2 \cdot \left(\frac{N+1}{2}\right) = N+1 = 1 \cup \mathbb{Z}_N$

6

Jeśli posługujemy się liczbami wymiernymi, to możemy w ich zbiorze zdefiniować odwrotność liczby. Np. odwrotnością liczby 2 jest ½. Dlatego, że 2 * ½ = 1. Jak wygląda odwrotność w zbiorze „Z_N”? Na wzór definicji odwrotności w zbiorze liczb wymiernych, możemy sformułować odwrotność w „naszym” zbiorze: Odwrotnością liczby x w zbiorze Z_N jest liczba y spełniająca zależność x * y = 1 (modulo N). Tak wartość odwrotna oznaczana jest x^{-1}.

Odwrotność modularna (2)

Które elementy ze zbioru $\mathbb{Z}_N$ posiadają odwrotność?

Lemat: x należący do $\mathbb{Z}_N$ posiada odwrotność wtedy i tylko wtedy, gdy nwd(x, N) = 1

Dowód:

nwd(x,N)=1 ⇒ ∃ a, b: a·x + b·N = 1 ⇒ a·x = 1 w $\mathbb{Z}_N$

⇒ $x^{-1} = a$ w $\mathbb{Z}_N$

nwd(x,N) > 1 ⇒ ∀a: nwd(a·x, N) > 1 ⇒ a·x ≠ 1 w $\mathbb{Z}_N$

nwd(x,N)=2 ⇒ ∀a: a·x jest parzyste ⇒ a·x ≠ b·N+1 (parzyste / niepamyste)

7

Powstaje pytanie, które z liczb należące do Z_N, posiadają odwrotność. Istnieje twierdzenie (tutaj nazywane Lematem = tw. pomocniczym), które mówi, że dany element w zbiorze ZN ma odwrotność wtedy i tylko w tedy, gdy x jest względnie pierwszy do N, co oznacza, że największy wspólny dzielnik danej liczby i N wynosi 1. Dowód można przeprowadzić w oparciu o wcześniejsze twierdzenie dotyczące możliwości przedstawienia największego wspólnego dzielnika w postaci kombinacji liniowej liczb dla, których ten dzielnik jest wyznaczany.
Jeśli założymy, że nwd(x, N) = 1, to można to wyrażenie przekształcić w a*x + b * N = 1. Wyrażenie b * N w arytmetyce modularnej można opuścić i otrzymać a*x = 1. A można znaleźć i jest ono odwrotnością liczby x. Jeśli jedna założymy, że nwd(x, N) > 1, to posługując się tym samym zapisem dojdziemy do wniosku, że a*x nie wynosi 1 w Z_N i wtedy liczba nie ma odwrotności.
Proszę zwrócić uwagę, że podane twierdzenie pozwala na drodze informatycznej obliczyć odwrotność modulo danej liczby. Należy zastosować rozszerzony algorytm Euklidesa do podanej zależności i wyliczyć a, które jest odwrotnością liczby.

Rozszerzenie notacji

Def: $\mathbb{Z}_N^*$ = (zbiór odwracalnych elementów w $\mathbb{Z}_N$) =

= { x∈ $\mathbb{Z}_N$: nwd(x,N) = 1 }

Przykład:

1. Dla liczby pierwszej p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^*$ = { 1, 5, 7, 11}

Dla x w $\mathbb{Z}_N^*$, można znaleźć x^{-1} używając rozszerzonego algorytmu Euklidesa.

8

Wprowadźmy kolejne oznaczenie Z_N^*jest to zbiór wszystkich „odwracalnych" (mający odwrotność_) elementów w zbiorze Z_N. Inaczej musi być dla nich spełniony warunek nwd(x,N) = 1.
Rozważmy przykład. Niech będzie dana liczba pierwsza p. Wiadomo, że wszystkie liczby z zakresie od 0 do p-1 będą względnie pierwsze w stosunku do p. Z tego zbiotu trzeba wyłączyć 0, ponieważ nwd(0,N) = 0. Wtedy dla danej liczby pierwszej zbiór $Z_p^*=Z_p$ /{0}, czyli są to wszystkie liczby mniejsze od p, bez 0. Wszystkie liczby z tego zbioru są odwracalne.
Wracając do poprzedniego przykładu, zbiór Z_{12}^*={1,5,7,11}, czyli wszystkie liczby, które są pierwsze względem 12. Znowu, dla pewnego x należącego od Z^*_N można znaleźć jego odwrotność z zastosowaniem rozszerzonego algorytmu Euklidesa.

Rozwiązywanie modularnych równań liniowych

Rozwiąż: **$a \cdot x + b = 0$** **w** $\mathbb{Z}_N$

Rozwiązanie: **$x = -b \cdot a^{-1}$** **w** $\mathbb{Z}_N$

Znajdź a^{-1} w $\mathbb{Z}_N$ Używając rozszerz. alg. Eucl. Złożoność: $O(\log^2 N)$

Co jeśli chcielibyśmy rozwiązać modularne równanie kwadratowe?

- dalsza część wykładu…

9

Jeśli rozważymy równanie liniowe w arytmetyce modularnej postaci: ax + b =0, to po „przeniesieniu b na drugą stronę równania”, otrzymujemy rozwiązanie: $x = -b*a^{-1}$ w zbiorze Z_N. Znowu, do znalezienia a^{-1} można się posłużyć rozszerzonym algorytmem Euklidesa. Czas rozwiązania algorytmu wynosi $\log^2 N$ (czyli jest bardzo dobry).

Podsumowanie na tym etapie

N oznacza n-bitową l. całkowitą dodatnią. p oznacza liczbę pierwszą.

- Z_N = { 0, 1, …, N-1 }
- $(Z_N)^*$ = (zbiór odwracalnych elementów w Z_N) =
  = $\{ x \in Z_N : nwd(x,N) = 1 \}$

Możemy znaleźć odwrotności z zast. rozsz. alg. Euclid. : czas = $O(n^2)$

10

Twierdzenie Fermata (1640)

Tw: Niech p będzie liczbą pierwszą

$$\forall\, x \in (Z_p)^* : \quad x^{p-1} = 1 \text{ in } Z_p$$

Przykład: p=5. $3^4 = 81 = 1$ w Z_5

Więc: $x \in (Z_p)^* \Rightarrow x \cdot x^{p-2} = 1 \Rightarrow x^{-1} = x^{p-2}$ in Z_p

inny sposób znajdowania odwrotności, ale mniej wydajny, niż rozszerzony alg. Euclidesa

11

Omówmy następujące twierdzenie sformułowane przez Fermata. Załóżmy, że dysponujemy pewną liczną pierwszą. Wtedy dla jakiegokolwiek elementu należącego do $(Z_p)^*$ spełniona jest zależność $x^{p-1} = 1$ w zbiorze Z_p. Przykładowo, jeśli rozważymy p=5, to biorąc liczbę z $Z_p = 3$ możemy obliczyć $3^4 = 81$ co wynosi 1 w Z_5. Ciekawostka: Twierdzenie nie zostało udowodnione przez Fermata, tylko sformułowane (hipoteza). Zostało ono udowodnione przez Euler'a 100 lat później (w zasadzie dowód dotyczył bardziej uogólnionej wersji tego twierdzenia).
Zastanówmy się nad prostym zastosowaniem tego twierdzenia. Załóżmy, że mamy element x należący do zbioru $(Z_p)^*$, rozpisujemy zależność $x^{p-1} = 1$ na $x*x^{p-2} = 1$. W arytmetyce modularnej wychodzi nam, że odwrotność x wynosi x^{p-2} w Z_p. Wystarczy więc podnieść x do potęgi p-2 (modulo p), aby znaleźć jego odwrotność. To jest dobry algorytm do znajdowania odwrotności, ale ma dwa ograniczenia. Po pierwsze działa tylko w arytmetyce modularnej dla liczb pierwszych i po drugie dłużej się wykonuje niż rozszerzony alg. Euklidesa (złożoność obliczeniowa wynosi $O(\log^3 p)$).

Zastosowanie: generowanie losowych liczb pierwszych

Załóżmy, że chcemy wygenerować dużą liczbę pierwszą

np., liczbę pierwszą p o długości 1024 bitów (np. $p \approx 2^{1024}$)

Krok 1: wybierz losową liczbę całkowitą z zakr. $p \in [\ 2^{1024}\ ,\ 2^{1025}-1\ ]$

Krok 2: sprawdź, czy $2^{p-1} = 1$ w zbiorze Z_p

Jeśli tak, zwróć p i zatrzymaj. Jeśli nie, idź do kroku 1 .

Prosty algorytm (nie najlepszy). **Pr[p nie jest pierwsza] < 2^{-60}**

12

Jakie może być proste zastosowanie twierdzenia Fermata? Załóżmy, że chcemy wygenerować dużą liczbę pierwszą, np. o długości 1024 bity. Interesuje nas liczba o wartości bliskiej 2^{1024}. Losujemy taką liczbę z zakresu 2^{1024} do $2^{1025}-1$ i bezpośrednio stosujemy tw. Fermata. Jeśli liczba je spełnia, to zakładamy, że jest pierwsza, jeśli nie, to losujemy dalej.

Nie mamy tu pewności, ponieważ nie analizujemy zbioru liczb względnie pierwszych w stosunku do p (działamy w jej „okolicy"). Ale jest twierdzenie (bardzo trudne do udowodnienia), że jeśli liczba przejdzie nasz test (oparty na tw. Fermata) do z bardzo dużym prawdopodobieństwem jest pierwsza. Jeśli analizujemy liczby 1024 bitowe, to prawdopodobieństwo, że nie trafiliśmy na l. pierwszą jest mniejsze niż 2^{-60}. Co więcej, jeśli dokonujemy takiego testu na coraz większych liczbach, to to prawdopodobieństwo jeszcze bardziej maleje w dużym tempie.

Zaletą pokazanego podejścia jest prostota. Są opracowane inne efektywne algorytmy, dla których jeśli ustalimy kandydata na liczbę pierwszą, to na 100% potwierdzą, że jest ona pierwsza.

Jeśli chodzi o spodziewaną liczbę „pętli" algorytmu, to jest dowód mówiący, że znalezienie liczby spełniającej nasze kryterium osiąga się nie później niż po kilkuset próbach.

Struktura $(Z_p)^*$

Tw (Euler): $(Z_p)^*$ jest **cykliczną grupą**, to znaczy, że

$\exists\, g \in (Z_p)^*$ takie, że $\{1, g, g^2, g^3, \ldots, g^{p-2}\} = (Z_p)^*$

g jest nazywany a **generatorem** $(Z_p)^*$

Przykład: p=7. $\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = (Z_7)^*$

Nie każdy element jest generatorem: $\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$

13

Kolejnym twierdzeniem na temat $(Z_p)^*$, które udowodnił Euler jest to, że taki zbiór jest grupą cykliczną. Jaka jest tego konsekwencja? Otóż wśród elementów zbioru istnieje taki element g, który podnoszony do kolejnych potęg [do potęgi p-2] (modulo p) da wszystkie elementy zbioru $(Z_p)^*$. Taki element nazywany jest generatorem.
Rozważmy prosty przykład. Niech wartością p będzie 7. Okazuje się, że liczba 3 jest generatorem dla naszej grupy cyklicznej. Gdy podniesiemy wartość 3 do kolejnych potęg (modulo 7), gdzie najwyższa z potęg wynosi 5 (7-2), to otrzymujemy wszystkie elementy zbioru $(Z_p)^*$. Warto zaznaczyć, że nie wszystkie elementy zbioru $(Z_p)^*$ są generatorami, przykładowo 2 nie pozwala wygenerować wszystkich elementów $(Z_p)^*$.

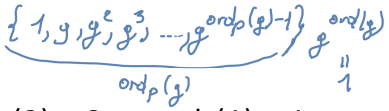
Rząd

Dla $g \in (Z_p)^*$ zbiór $\{1, g, g^2, g^3, \ldots\}$ jest nazywany

grupą wygenerowaną przez g, oznaczaną $<g>$

Def: **rząd** $g \in (Z_p)^*$ jest rozmiarem $<g>$

$$\mathbf{ord_p(g) = |<g>| = (najmniejsze\ a>0\ s.t.\ g^a = 1\ w\ Z_p)}$$



Przykłady: $ord_7(3) = 6$; $ord_7(2) = 3$; $ord_7(1) = 1$

Tw (Lagrange): $\forall g \in (Z_p)^*$: $\mathbf{ord_p(g)}$ dzieli $p-1$

14

Wprowadźmy pojęcie, które jest często stosowane. Jeśli mamy g należące do $(Z_p)^*$, to zbiór $\{1, g, g^2, g^3, \ldots\}$ jest nazywany grupą wygenerowaną przez g i oznaczaną $<g>$. Rzędem (order) nazywamy rozmiar $<g>$. Wracając do arytmetyki modularnej rząd można wyznaczyć jako najmniejszą wartość liczby całkowitej a, takiej, że $g^a = 1$ w Z_p (modulo p). Kolejne przykłady pokazują wartości rzędu dla arytmetyki modulo 7, jeśli rozważumy liczby g jako 3, 2 i 1.
Z rzędem grupy jest powiązane twierdzenie sformułowane przez Lagrange'a. Obliczony porządek zawsze dzieli liczbę p-1. W omówionych wcześniej przykładach: 6 dzieli 7-1, podobnie 3 dzieli 7-1. Ogólnie rząd będzie zawsze dzielnikiem wartości p-1.

Uogólnienie przez Euler'a twierdzenia Fermata (1736)

Def: Dla liczby całkowitej N definiujemy $\varphi(N) = |(Z_N)^*|$ (funkcja φ Euler'a)

Przykład: $\varphi(12) = |\{1,5,7,11\}| = 4$; $\varphi(p) = p-1$

Dla $N=p\cdot q$: $\varphi(N) = N-p-q+1 = (p-1)(q-1)$

Tw. (Euler): $\forall\, x \in (Z_N)^* : \quad x^{\varphi(N)} = 1 \quad \text{in } Z_N$

Przykład: $5^{\varphi(12)} = 5^4 = 625 = 1 \quad \text{in } Z_{12}$

Uogólnienie tw. Fermata. Podstawa systemu kryptografii RSA

15

Twierdzenie Fermata dotyczyło zbioru liczb modulo liczba pierwsza. Euler uogólnił to twierdzenie do pozostałych liczb. Sformułował funkcję $\varphi(N) = |(Z_N)^*|$, która wyliczała rozmiar zbioru liczb względnie pierwszych w stosunku do wartości modułu (liczby, która jest podstawa danego systemu modularnego i może to być dowolna liczba całkowita dodatnia). Przykładowo, dla N=12 wartość funkcji $\varphi(N)$ wynosi 4. Gdybyśmy rozważyli wartość funkcji Eulera'a dla pewnej liczby pierwszej p, to okazuje się, że wynosi ona p-1. Na następnych wykładach będziemy się interesowali specjalnymi wartościami N złożonymi z iloczynu dwóch liczb pierwszych. Dla liczby N=p*g, gdzie p i q są pierwsze $\varphi(N) = N-p-q+1 = (p-1)(q-1)$. To jest fakt, do którego wrócimy, kiedy będziemy omawiać system RSA.

Dla funkcji φ Eulera Euler udowodnił następujące twierdzenie: Dla każdego x należącego do zbioru $(\mathbf{Z_N})^*$ wartość $x^{\varphi(N)} = 1$ w zbiorze $\mathbf{Z_N}$. To jest uogólnienie twierdzenia Feramta, które dotyczyło takiej samej zależności, ale tylko jeśli N było liczbą pierwszą.

Rozważmy przykłady. Ile wynosi $5^{\varphi(12)}$? Wiemy, że $\varphi(12)=4$, stąd wynik podnoszenia do potęgi wynosi 625 i jest to liczba dająca resztę jeden przy dzieleniu przez 12.

Okazuje się, że tw. Euler'a jest też specjalnym przypadkiem bardzie ogólnego twierdzenia Lagrange'a.

Pierwiastki równań wielomianowych wyższych rzędów w arytmetyce modularnej (1)

Wiemy jak rozwiązać równanie liniowe w arytmetyce modularnej:

$\mathbf{a \cdot x + b = 0}$ w Z_N Rozwiązanie: $\mathbf{x = -b \cdot a^{-1}}$ w Z_N

A co gdybyśmy chcieli otrzymać rozwiązania dla równań wielomianowych wyższych rzędów ?

Przykład: niech p będzie l. pierwszą i $c \in Z_p$. Czy możemy rozwiązać:

$x^2 - c = 0$, $y^3 - c = 0$, $z^{37} - c = 0$ w Z_p

16

Wiemy jak rozwiązać równanie liniowe w arytmetyce modularnej. Wystarczy, że zastosujemy algorytm znajdujący odwrotność dla a i mnożąc tę wartość przez –b. Teraz będziemy się zastanawiać, jak rozwiązań równania wyższego rzędu. Szczególnie (w przyszłych zastosowaniach) będą nas interesować rozwiązania równań modulo liczby pierwsze (modulo p). Postaci wielomianów, które będą nas interesowany to $x^a - c = 0$ w zbiorze Z_p. Rozwiązania takich równań dają pierwiastek a-tego rzędu z c.

Pierwiastki równań wielomianowych wyższych rzędów w arytmetyce modularnej (2)

Niech p będzie liczbą pierwszą i $c \in Z_p$.

Def: $x \in Z_p$ taki, że $x^e = c$ w Z_p jest nazywany e-tym pierwiastkiem c.

$6^3 = 216 = 7$ w $\mathbb{Z}_{11}$

Przykłady:

$7^{1/3} = 6$ w $\mathbb{Z}_{11}$

$3^{1/2} = 5$ w $\mathbb{Z}_{11}$

$1^{1/3} = 1$ w $\mathbb{Z}_{11}$

$2^{1/2}$ nie istnieje w $\mathbb{Z}_{11}$

17

W rozważaniach zakładamy, że p to liczba pierwsza, a c jest liczbą należącą do zbioru Z_p. Mówimy, że x należący do Zp spełniający równanie $x^e = c$ jest e-tym pierwiastkiem liczby c. Poniżej znajdują się przykłady obliczeń takich wartości w arytmetyce modularnej względem 11. Przykładowo trzeci pierwiastek z 7 modulo 11 wynosi 6, bo $6^3 = 216$; 216 mod 11 = 7.
Podobnie można sprawdzić wyniki dwóch innych przykładów. Warto wiedzieć, że nie dla wszystkich wartości zbioru Zp da się obliczyć e-te pierwiastki. Przykładowo nie istnieje pierwiastek drugiego stopnia z 2 w arytmetyce modulo 11.

Łatwy przypadek

Kiedy istnieje $c^{1/e}$ w Z_p? Czy można wyliczyć te wartość w efektywny sposób?

Prosty przypadek: załóżmy, że nwd(e , p-1) = 1

Wtedy dla wszystkich c w $(Z_p)^*$: $c^{1/e}$ istnieje Z_p i jest łatwy do znalezienia.

Dowód: niech $d = e^{-1}$ w Z_{p-1}. Wtedy $c^{1/e} = c^d$ w Z_p

$d \cdot e = 1$ in Z_{p-1} $\Rightarrow \exists\, k \in \mathbb{Z} : d \cdot e = k(p-1)+1 \Rightarrow (c^d)^e = c^{d \cdot e} = c^{k(p-1)+1} = [c^{p-1}]^k \cdot c = c$ w Z_p

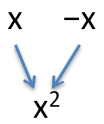
18

Zastanówmy się, kiedy istnieją e-te pierwiastki oraz, czy łatwo je obliczyć. Zaczniemy od prostej sytuacji. Załóżmy, że e i p-1 są względem siebie pierwsze. Wtedy okazuje się, że zawsze istnieje pierwiastek równania. Dlaczego tak jest?
Po pierwsze, skoro e i p-1 są względem siebie pierwsze, to istnieje odwrotność e modulo p-1. Można też udowodnić, że $c^{1/e} = c^d$ w Z_p. Dostajemy też efektywny algorytm rozwiązania tego przypadku. Najpierw znajdujemy odwrotność e modulo p-1, a potem podnosimy c do tej znalezionej odwrotności. Tak „pieczemy dwie pieczenie na jednym ogniu”.
Zastanówmy się dlaczego to twierdzenie zachodzi. Wyjdźmy od zależności $d*e = 1$ w Z_p. Stąd wynika, że istnieje takie k należące do Z_{p-1}, że $d*e = k(p-1)+1$. Rozważmy, ile wynosi $(c^d)^e$? Z własności potęgowania wartość ta wynosi c^{d*e}. Podstawiając wcześniej wprowadzoną zależność otrzymujemy $c^{k(p-1)+1}$, czyli $[c^{p-1}]^k * c$, co równa się c mod p.

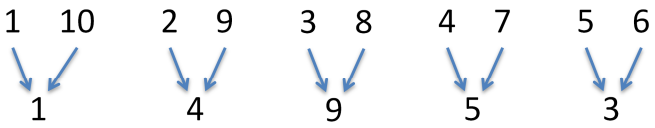
Przypadek e=2 (pierwiastki kwadratowe)

Jeśli p jest nieparzystą liczbą pierwszą nwd(2, p-1) ≠ 1

Fakt: w $\mathbb{Z}_p^*$, x ⟶ x² jest funkcją 2-do-1



Przykład: w $\mathbb{Z}_{11}^*$:



Def: x w $\mathbb{Z}_p$ jest resztą kwadratową jeśli posiada pierwiastek kwadratowy w $\mathbb{Z}_p$

p nieparzysta liczba pierwsza ⇒ liczba reszt kwadratowych w $\mathbb{Z}_p$ wynosi (p-1)/2 + 1

19

Ostatnim łatwym przypadkiem jest sytuacja, kiedy e nie jest relatywnie pierwsze do p-1. Dobrym przykładem ilustrującym obliczanie wartości pierwiastków w takim przypadku jest rozważenie rozwiązań dla e=2 (pierwiastki równania kwadratowego). Chcemy obliczyć pierwiastki kwadratowe z c w Z_p.
Jeśli p jest nieparzysta, to p-1 jest parzysta, czyli nwd(2, p-1) ≠ 1. Algorytm z wcześniejszego slajdu nie będzie się nadawał do obliczania pierwiastków. Poszukiwanie pierwiastków modulo nieparzyste p jest nazywane funkcją 2 do 1. Poszukujemy wartości x i – x, które podniesione do kwadratu dają x^2. Rozważmy obliczanie pierwiastków kwadratowych modulo 11. Jeśli weźmiemy 1 i -1 mod 11, czyli 1 i 10 to zauważymy, że każda z tych liczb podniesiona do kwadratu modulo 11 da wartość 1, podobnie 2 i 9 (-2 modulo 11) podniesione do kwadratu modulo 11 dają wartość 4, i tak dalej…
Zwróćmy uwagę, że elementy 1, 4, 9, 5, 3 mają pierwiastki kwadratowe. Okazuje się, że żadne inne elementy w zbiorze Z_p tych pierwiastków kwadratowych nie posiadają. Takie specjalne wartości zbioru Z_p, które posiadają swoje pierwiastki kwadratowe nazywa się resztami kwadratowymi. Można łatwo obliczyć liczbę takich reszt kwadratowych w zbiorze Zp (gdzie p to nieparzysta liczba pierwsza). Wynosi ona (p-1)/2+1 (To +1 wynika z tego, że 0 zawsze ma pierwiastek kwadratowy równy 0). Ważnym wnioskiem z tej zależności jest to, że tylko około połowa elementów zbioru Z_p jest resztą kwadratową.
Zwróćmy uwagę, że w pierwszym przypadku, jeśli e i p-1 były względem siebie pierwsze, to każdy element w Zp miał pierwiastek. W drugim przypadku, tylko połowa.

Twierdzenie Euler'a

Tw: x w $(Z_p)^*$ jest resztą kw. $\Leftrightarrow x^{(p-1)/2} = 1$ w Z_p (p niep. L. pier.)

Przykład:

in $\mathbb{Z}_{11}$:	1^5,	2^5,	3^5,	4^5,	5^5,	6^5,	7^5,	8^5,	9^5,	10^5
=	1	-1	1	1	1,	-1,	-1,	-1,	1,	-1

Uwaga: $x \neq 0 \Rightarrow x^{(p-1)/2} = \left(x^{p-1}\right)^{1/2} = 1^{1/2} \in \{1, -1\}$ w Z_p

Def: $x^{(p-1)/2}$ jest nazywany **Symbolem Legendre'a** x ponad p (1798)

20

Naturalnym pytaniem jest więc, czy możemy stwierdzić, że dany element zbioru $(Z_{p)})^*$ ma pierwiastek kwadratowy? Kolejnym osiągnięciem Euler'a było określenie kryterium sprawdzenia, czy dany składnik zbioru (Zp)* jest resztą kwadratową. Dany element x należący do zbioru (Zp)* jest resztą kwadratową, jeśli wartość $x^{(p-1)/2} = 1$. W Tabeli obliczono wartość wyrażenia $x^{(p-1)/2}$ dla wszystkich elementów Z11 i okazuje się, że tylko dla reszt kwadratowych wynosi ono 1.
Zwróćmy uwagę, że obliczając wartość $X^{(p-1)/2}$ dla x różnego od 0, zawsze otrzymamy wartość 1 lub -1.
Pokazane twierdzenie wskazuje, że dany element ma pierwiastki kwadratowe, ale nie pokazuje, jak je obliczyć.

Obliczanie pierwiastków kwadratowych modulo p

Załóżmy $p = 3 \pmod 4$

Tw. pom.: jeśli $c \in (Z_p)^*$ jest resztą kwadratową wtedy

$\sqrt{c} = c^{(p+1)/4}$ w Z_p

Dowód: $\left(c^{(p+1)/4}\right)^2 = c^{(p+1)/2} = c^{\frac{p-1}{2}} \cdot c = 1 \cdot c = c$ w Z_p

Kiedy $p = 1 \pmod 4$, także może zostać policzone, ale z dłuższym czasem obliczeń $\approx O(\log^3 p)$

algorytm losowy ...

21

Rozważmy dwa przypadki. W pierwszym p = 3 (mod 4). Wtedy można wyprowadzić gotowy wzór pokazany na slajdzie. Dowód na poprawność formuły również pokazano na slajdzie. Drugi przypadek dotyczy sytuacji, gdy dna liczba p = 1 (mod 4). Wtedy można opracować efektywny algorytm wyznaczania pierwiastków. Stosuje on losowanie.
Na marginesie, jeśli ktoś byłby w stanie udowodnić rozszerzoną hipotezę Riemann'a, to równocześnie sformułowałby deterministyczny algorytm dla obliczania pierwiastków dla przypadku gdy p = 1 mod 4.
Próby obliczenia pierwiastków kwadratowych z liczb x modulo p wymaga zastosowania rezultatów z zakresu matematyki, z których część nie jest udowodniona. Jak dotąd nie istnieje deterministyczny algorytm znajdowania pierwiastków dla p=1 (mod 4), ale algorytm losowy radzi sobie z tym problemem wystarczająco dobrze.

Rozwiązywanie równań kwadratowych mod p

Rozwiąż: $a \cdot x^2 + b \cdot x + c = 0$ w Z_p

Rozwiązanie: $x = (-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}) / 2a$ w Z_p

- Znajdź $(2a)^{-1}$ w Z_p z zast. rozszerzonego alg. Euklidesa
- Znajdź pierwiastek kwadratowy $b^2 - 4 \cdot a \cdot c$ w Z_p (jeśli istnieje) używając algorytm obliczania pierwiastków kwadratowych

22

Gdybyśmy się zastanawiali, jak rozwiązać równanie kwadratowe w arytmetyce modularnej, to okazuje się, że dysponujemy już wszystkimi algorytmami do tego potrzebnymi. Działa tu wzór do rozwiązywania równania kwadratowego znany od szkoły średniej. Wartość 1/2a możemy wyznaczyć z rozszerzonego alg. Euklidesa, zaś pierwiastek kwadratowy z zastosowaniem jednego z omówionych wcześniej algorytmów. Oczywiście wzory będą działać, jeśli taki pierwiastek kwadratowy istnieje w Zp.

Obliczanie n-tego pierwiastka mod N

Niech N liczbą złożoną (nie pierwszą) i e>1

Kiedy $c^{1/e}$ **w** Z_N istnieje? Czy możemy go efektywnie obliczyć?

Odpowiedź na to pytanie wymaga rozkładu na czynniki pierwsze wartości N

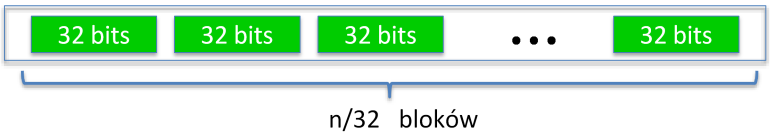
(jak dotąd tyle wiemy)

23

A co, jeśli chcemy wyznaczyć pierwiastki n-tego stopnia w arytmetyce mod N, gdzie N nie jest liczbą pierwszą. Ponownie trzeba sformułować pytanie, kiedy taki pierwiastek istniej i czy się go da efektywnie obliczyć. Tutaj poszukiwanie rozwiązania jest znacznie trudniejsze obliczeniowo i jego złożoność jest porównywalna do poszukiwania rozkładu na liczby pierwsze wartości N.

Reprezentowanie dużych liczb

Reprezentacja n-bitowej wartości całowitej (np. n=2048) na 64 -bitowej maszynie



n/32 bloków

Uwaga: niektóre procesory mają rejestry 128-bitowe (lub dłuższe) i wspierają mnożenie na takich rejestrach

24

Duże liczby, przekraczające normalny rozmiar rejestrów procesora są dzielone na bloki, i każdy z bloków reprezentuje „porcję” długiej liczby. Współczesne procesory dysponują dłuższymi rejestrami i pozwalają na tych dłuższych rejestrach przeprowadzać operacje arytmetyczne. Liczby dzieli się na 32-bitowe pola, aby umożliwić efektywne mnożenie (aby po wymnożeniu dwóch 32-bitowych pól wynik zmieścił się na 64 bitach).

Arytmetyka

Mając: dwie n-bitowe liczby całkowite

- **Dodawanie i odejmowanie**: liniowy czas $O(n)$
- **Mnożenie** : naiwnie $O(n^2)$. Karatsuba (1960): $O(n^{1.585})$

  Podstawowy pomysł: $(2^b x_2 + x_1) \times (2^b y_2 + y_1)$ z 3 mnożeniami.

  Najlepszy (asymptotyczny) algorytm: około $O(n \cdot \log n)$.
- **Dzielenie z resztą**: $O(n^2)$.

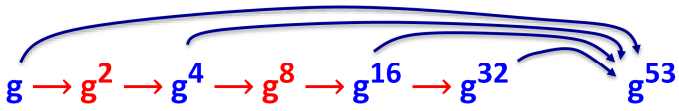
Podnoszenie do potęgi

Skończona grupa cykliczna G (np. $G = \mathbb{Z}_p^*$)

Cel: mając g w G i x oblicz g^x

Przykład: niech $x = 53 = (110101)_2 = 32+16+4+1$

Wtedy: $g^{53} = g^{32+16+4+1} = g^{32} \cdot g^{16} \cdot g^4 \cdot g^1$



26

Jeśli rozważamy operacje podnoszenia do potęgi, to najpierw założymy, że zbiór, w którym
się będziemy poruszać jest skończona grupa cykliczna, np. (Zp)*. Normalnie myśląc o podnoszeniu do potęgi wyobrażamy sobie wielokrotne mnożenie tej samej liczby. Niestety taka operacja na komputerze ma również potęgową złożoność obliczeniową. Dla małych liczb to może działać, ale jeśli wyobrazimy sobie, że taką operację mamy wykonać dla dużych x (np. 500 bitów), to czas przeprowadzenia takich obliczeń przekracza możliwości współczesnych komputerów, lub jest nieakceptowalnie długi wraz ze wzrostem wykładnika.
Powstaje pytanie, czy istnieje jakiś inny „sprytniejszy" sposób przeprowadzenia operacji podnoszenia do potęgi? Okazuje się, że jest.
Algorytm nazywa się „powtarzane podnoszenie do kwadratu" (ang. Repeated squaring algorithm). Rozważmy przykład ilustrujący to rozwiązanie.
Załóżmy, że potęga, do której chcemy podnieść pewną liczbą wynosi 53 i nie chcemy 53 razy mnożyć naszej liczby, aby uzyskać wynik. Wartość naszego wykładnika zapisujemy w kodzie binarnym. Automatycznie uzyskujemy rozkład tej liczby na potęgi dwójki. Łatwo zauważyć, że podnoszenie do potęgi 53 można rozbić na iloczyn liczb podniesionych odpowiednio do 23, 16, 4 i 1. Aby przyspieszyć nasze obliczenia bierzemy liczbę g i podnosimy ją do kwadratu, a potem kwadrat do kwadratu itd. Po uzyskaniu tych wyników możemy odpowiednie z nich ze sobą pomnożyć, aby uzyskać interesujący nas wynik. Wystarczyło wykonać tylko 5 podnoszeń do kwadratu i cztery mnożenia. Czyli wykonując 9 mnożeń otrzymujemy g do 53. Okazuje się, że jest to ogólne podejście pozwalające szybko uzyskiwać potęgi liczb o dużym wykładniku.

Efektywny algorytm obliczania potęg

Wejście: g w G i x>0 ; **Wyjście**: g^x

zapisz $x = (x_n\, x_{n-1} \ldots x_2\, x_1\, x_0)_2$

$\log_2 x$

```
y ⟵ g   ,   z ⟵ 1
for i = 0 to n do:
        if  (x[i] == 1):     z ⟵ z·y
        y ⟵ y²
zwróć  z
```

example: g^{53}

y	z
g^2	g
g^4	g
g^8	g^5
g^{16}	g^5
g^{32}	g^{21}
g^{64}	$\mathbf{g^{53}}$

27

Podsumowanie czasów wykonywania obliczeń

Mając n-bitową liczbę całkowitą N:

- **Dodawanie i odejmowanie w Z_N**: czas liniowy $T_+ = O(n)$
- **Mnożenie modulo w Z_N**: naiwnie $T_\times = O(n^2)$
- **Podnoszenie do potęgi w Z_N (g^x)**:

$$O\big((\log x)\cdot T_\times\big) \leq O\big((\log x)\cdot n^2\big) \leq O(n^3)$$

28

Łatwe problemy

- Mając liczbę złożoną N i x w Z_N znaleźć x^{-1} w Z_N

- Mając liczbę pierwszą p i funkcje wielomianową f(x) w $Z_p[x]$

  znaleźć x w Z_p takie, że f(x) = 0 w Z_p (jeśli istnieje)

  Czas wykonywania jest liniowy w zależności od stopnia (f)

… ale wiele problemów jest trudnych

Trudne problemy w dziedzinie liczb pierwszych

Ustal liczbę pierwszą p>2 i g in $(Z_p)^*$ rzędu q.

Rozważ funkcję: $x \longmapsto g^x$ w Z_p

teraz rozważ funkcję odwracającą:

$\mathbf{Dlog_g (g^x) = x}$ **gdzie x w {0, …, q-2}**

Przykład:

in $\mathbb{Z}_{11}$:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10
$Dlog_2(\cdot)$:	0,	1,	8,	2,	4,	9,	7,	3,	6,	5

30

Ustalmy p>2 jako duża liczbę (600 cyfr) i wybierzmy jakiś element g należący do Zp. Załóżmy, że rząd tej grupy wynosi q. Rozważamy funkcję przekształcającą wartość x na g^x. Pokazaliśmy wcześniej, że obliczenie takiej funkcji jest proste z zastosowaniem odpowiedniego algorytmu. Ale rozważmy pytanie o odwrócenie tej operacji. Mamy wartość g^x, a chcemy ją zlogarytmować, żeby odzyskać x. W dziedzinie liczb rzeczywistych jest to definicja logarytmu, ale w arytmetyce modularnej jest to definicja problemu dyskretnego logarytmu (Dlog).
W tabeli pokazano wartości dyskretnych logarytmów o podstawie 2 w arytmetyce modulo 11. Przykładowo $Dlog_2(8)=3$, bo $2^3=8$.
Okazuje się, że obliczenie dyskretnych logarytmów jest zadaniem trudnym obliczeniowo. Dla małych liczb pierwszych możemy siąść z kartką papieru i sprawdzić sobie wyniki, ale dla liczb 2000-bitowych nie ma dobrego algorytmu znajdowania logarytmów.

DLOG: bardziej ogólnie

Niech **G** będzie skończoną grupą cykliczną i **g** jej generatorem

$G = \{ 1 , g , g^2 , g^3 , \ldots , g^{q-1} \}$ (q jest nazywany rzędem G)

Def: Mówimy, że **DLOG is trudny w G** if dla wszystkich efektywnych algorytmów A

$\Pr_{g \leftarrow G,\, x \leftarrow Z_q} [\ A(G, q, g, g^x) = x\ ] < \text{negligible}$

Przykładowi kandydaci:
(1) $(Z_p)^*$ dla dużych p, (2) Grupy krzywych eliptycznych mod p

31

Można uogólnić problem poszukiwania dyskretnego algorytmu. Niech G będzie skończoną grupą cykliczną a g jej generatorem (Grupa zawiera wszystkie potęgi g i taka grupa jest opisywana przez duże G). Mówimy, że problem obliczania DLOG jest trudny w grupie G, jeśli nie istnieje efektywny algorytm obliczający wartość dyskretnego algorytmu. Co przez to rozumiemy? Jeśli weźmiemy grupę G, jej rząd q, losową wartość x z Zq i losową wartość g z G, to prawdopodobieństwo, że znajdziemy algorytm, który policzy dyskretny logarytm jest pomijalne (bardzo, bardzo małe). Jeśli to jest prawda dla wszstkich efektywnych algorytmów, wtedy mówimy że problem DLOG jest trudny.
Są dwie grupy, w których ten problem jest trudny:
(Zp)* dla dużych P (podstawa dla protokołu Diffie-Hellman'a), oraz Grupy krzywych eliptycznych. Przy okazji, warto przypomnieć, że rozwiązanie problemu poszukiwania DLOG jest „trudniejszy" w domenie krzywych eliptycznych iż w (Zp)*. Wystarczy więc zastosować krótsze liczby, jako parametry do rozwiązania problemu, aby był tak samo trudny.

Poszukiwanie Dlog w $(Z_p)^*$ (n-bitowa liczba pierwsza)

Najlepszy znany algorytm (GNFS): czas wyk. exp($\tilde{O}(\sqrt[3]{n})$)

rozm. klucza szyfr.	rozm. modułu	Rozmiar grupy krzyw. eliptycznych
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

Rezultat: powolne przejście z arytmetyki (mod p) do krzywych eliptycznych.

32

Zastosowanie: odporność na kolizje

Wybierz grupę G gdzie Dlog jest trudny (np. $(Z_p)^*$ dla dużych p)

Niech q = |G| będzie liczbą pierwszą.

Wybierz generatory g, h należące do G

Dla x,y ∈ {1,…,q} zdefiniuj **$H(x,y) = g^x \cdot h^y$** **w G**

Tw. pom.: Znalezienie kolizji H(.,.) jest tak samo trudne jak obliczenie $Dlog_g(h)$

Dowód: Załóżmy, że mamy kolizję $H(x_0,y_0) = H(x_1,y_1)$

wtedy $g^{x0} \cdot h^{y0} = g^{x1} \cdot h^{y1} \Rightarrow g^{x0-x1} = h^{y1-y0} \Rightarrow h = g^{x0-x1/y1-y0}$

33

Pokażmy bezpośrednie zastosowanie trudności rozwiązania DLOG. Możemy zbudować funkcję HASH odporną na kolizje na bazie problemu dyskretnego logarytmu. Wybieramy grupę G, dla której problem DLOG jest trudny. Możemy o niej myśleć, jako (Zp)* dla dużych p. Mamy też rząd tej grupy =q, który jest liczbą pierwszą. Aby zdefiniować funkcję HASH wybieramy dwa elementy tej grupy i nazywamy je g i h. Funkcja HASH dla wejść x i y wynosi g^x*h^y.
Znalezienie kolizji w takiej funkcji jest tak samo trudne jak wyznaczenie DLOG.
Na slajdzie znajduje się prosty dowód sprowadzający pojawienie się kolizji do rozwiązania problemu DLOG.
Zaproponowana funkcja skrótu jest odporna na kolizje, ale nie jest stosowana w praktyce, ponieważ obliczanie za każdym razem dwa razy potęg jest czasochłonne. SHA256 działa szybciej.

Problemy trudne modulo liczby złożone

Rozważamy zbiór liczb całkowitych: (np. dla n=1024)

$\mathbb{Z}_{(2)}(n)$:= { N = p·q gdzie p,q są n-biowymi l. pierwszymi }

Problem 1: Rozłóż N na czynniki pierwsze w $\mathbb{Z}_{(2)}(n)$ (np. dla n=1024)

Problem 2: Mając daną funkcję wielomianową **f(x)** gdzie stopień(f) > 1 i losową wartość N w $\mathbb{Z}_{(2)}(n)$

znajdź x w $\mathbb{Z}_N$ takie, że f(x) = 0 w $\mathbb{Z}_N$

34

Jakie są znane trudne problemy modulo liczby złożone (w odróżnieniu do wcześniej omawianych problemów modulo liczby pierwsze)? Załóżmy, że dysponujemy liczbami 1024 bitowymi. Definiujemy zbiór $Z_{(2)}(n)$, który definiujemy jako zestaw liczb N, które oblicza się jako iloczyn dwóch liczb pierwszych o długości n-bitów. (2) w oznaczeniu informuje, że liczbę można rozłożyć na 2 liczby pierwsze o podobnej długości równej około n-bitów. Kiedy mamy zbiór tak skonstruowanych liczb, to możemy sformułować 2 następujące trudne problemy.

1. Jak rozłożyć taką liczbę na czynniki pierwsze? Dla liczb o długości 1024 nie znaleziono rozwiązania, ale można się spodziewać, że wkrótce taka faktoryzacja zostanie opracowana. Stąd lepiej stawiać ten problem dla liczb o długości 2048 bitów.
2. Jeśli weźmiemy nieliniowy wielomian rzędu większego niż jeden, potem wylosujemy N ze zbioru $Z_{(2)}(n)$ i postawimy sobie zadanie znalezienia x, który jest pierwiastkiem tego wielomianu w Z_N. Tego problemu nie możemy rozwiązać bez wcześniejszego rozłożenia na czynniki pierwsze N, a potem obliczania pierwiastków. Są systemy RSA, które bazują na trudności rozwiązania tego poroblemu.

Problem rozkładu na czynniki pierwsze

Gauss (1805): *W swobodnym przekładzie:*
"Problem rozróżniania liczb pierwszych od liczb złożonych i rozkładnia tych ostatnich na czynniki pierwsze jest uważany za jednen z najważniejszych i użytecznych w arytmetyce"

Najlepszy znany alg. (NFS): run time exp($\tilde{O}(\sqrt[3]{n})$) dla n-bitowych l. całkowitych

Bieżący rekord świata: **RSA-768** (232 cyfr)

- Praca: dwa lata na setkach komputerów
- Rozkład 1024-bitowych liczb: około 1000 razy trudniejszy

⇒ możliwy do rozwiązania w tej dekadzie

35

Problem sprawdzania, czy dana liczba jest pierwsza oraz problem rozkładnia liczby na czynniki pierwsze był uznawany już od starożytności za ważny. Jego rolę podkreślił Gauss w swojej słynnej wypowiedzi.
Obecnie problem sprawdzenia, czy dana liczba jest pierwsza czy nie jest na drodze informatycznej dobrze „rozpracowany". Istnieją efektywne algorytmy losowe i deterministyczne wykazujące, że dana liczba jest pierwsza.
Problem rozkładu na czynniki pierwsze wciąż uznawany jest za trudny. Najlepszy algorytm (sito ciała liczbowego) NFS ma złożoność obliczeniową wykładniczą o wykładniku trzeci pierwiastek z n (dla liczb n-bitowych). Bieżący rekord faktoryzacji dotyczy liczby 768-bitowej (232 cyfr) i wymagało to 2 lat pracy setek komputerów komputerów. Uważa się, że rozwiązanie problemu faktoryzacji liczb 1024 bitowych jest 1000 razy trudniejsze, więc przy zastosowaniu tej samej platformy obliczeniowej trwałoby 2000 lat. Mając na uwadze rozwój komputerów można się spodziewać skutecznego rozwiązania tego problemu w tej dekadzie.

Literatura uzupełniająca

- A Computational Introduction to Number Theory and Algebra,
  V. Shoup, 2008 (V2), Chapter 1-4, 11, 12

Dostępne na: **//shoup.net/ntb/ntb-v2.pdf**

36

Wiadomości z teorii liczb można uzupełnić z książki w wolnym dostępie. W szczególności czytając rozdziały 1-4, 11 i 12.