Zastosowanie liczb pierwszych w algorytmie RSA

Julia Wieleba, Daria Popławska

Spis treści

1 Wstęp

W dobie dynamicznego rozwoju technologii informacyjnych oraz powszechnej cyfryzacji życia codziennego zapewnienie bezpieczeństwa przesyłanych i przechowywanych danych stało się jednym z ważniejszych problemów współczesnego świata. Każdego dnia użytkownicy korzystają z bankowości elektronicznej, komunikatorów, sklepów internetowych oraz wielu innych usług cyfrowych. W takich sytuacjach konieczne jest zabezpieczenie informacji przed dostępem osób nieuprawnionych.

Jedną z dziedzin zajmujących się ochroną informacji jest kryptografia. Jej zadaniem jest takie przekształcenie wiadomości, aby mogła zostać odczytana tylko przez osobę posiadającą odpowiedni klucz. Szczególne znaczenie we współczesnej kryptografii ma kryptografia asymetryczna, w której wykorzystuje się dwa różne klucze: publiczny i prywatny. Jednym z najbardziej znanych algorytmów tego typu jest algorytm RSA, opisany w 1978 roku przez Ronalda Rivesta, Adiego Shamira oraz Leonarda Adlemana.[1]

Algorytm RSA znajduje zastosowanie między innymi w mechanizmach podpisu cyfrowego, certyfikatach oraz wybranych elementach protokołów zabezpieczających komunikację internetową. Jego bezpieczeństwo opiera się na własnościach matematycznych liczb pierwszych oraz na trudności rozkładu dużych liczb złożonych na czynniki pierwsze.

Głównym celem niniejszej pracy jest przedstawienie roli liczb pierwszych w algorytmie RSA oraz pokazanie, że bezpieczeństwo tego algorytmu wynika z trudności problemu faktoryzacji. W szczególności zostanie pokazane, że znajomość rozkładu liczby będącej iloczynem dwóch dużych liczb pierwszych umożliwia odtworzenie klucza prywatnego, a więc złamanie systemu RSA.

W pracy omówione zostaną podstawowe pojęcia z teorii liczb, takie jak liczby pierwsze, rozkład na czynniki pierwsze, największy wspólny dzielnik, arytmetyka modularna oraz funkcja Eulera. Następnie przedstawione zostaną podstawy kryptografii asymetrycznej oraz opis działania algorytmu RSA, w tym generowanie kluczy, szyfrowanie i odszyfrowywanie wiadomości. Kolejna część pracy będzie dotyczyć ataku faktoryzacyjnego dla małych liczb oraz wyjaśnienia, dlaczego podobny atak nie jest praktycznie wykonalny dla dużych modułów RSA. Omówione zostaną również kwestie czasu, pamięci oraz komputerów kwantowych.

[1] R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, 1978, `https://people.csail.mit.edu/rivest/Rsapaper.pdf`.

2 Podstawowe pojęcia z teorii liczb

Teoria liczb jest działem matematyki, który zajmuje się własnościami liczb całkowitych. W kryptografii, a szczególnie w algorytmie RSA, odgrywa ona bardzo ważną rolę. Aby zrozumieć działanie RSA, należy najpierw omówić kilka podstawowych pojęć: liczby pierwsze, liczby złożone, rozkład na czynniki pierwsze, największy wspólny dzielnik, arytmetykę modularną, funkcję Eulera oraz wybrane twierdzenia teorii liczb.[2]

2.1 Liczby pierwsze i liczby złożone

Liczbą pierwszą nazywa się liczbę naturalną większą od 1, która ma dokładnie dwa dzielniki naturalne: 1 oraz samą siebie. Przykładami liczb pierwszych są:

$$2, 3, 5, 7, 11, 13.$$

Liczbą złożoną nazywa się natomiast liczbę naturalną większą od 1, która ma więcej niż dwa dzielniki naturalne. Przykładami liczb złożonych są:

$$4, 6, 8, 9, 10.$$

Liczba 1 nie jest ani liczbą pierwszą, ani liczbą złożoną.

Liczby pierwsze mają bardzo duże znaczenie w matematyce, ponieważ można traktować je jako podstawowe elementy budowy liczb naturalnych. W algorytmie RSA wykorzystuje się dwie duże liczby pierwsze do utworzenia modułu, który jest częścią klucza publicznego.

2.2 Rozkład liczb na czynniki pierwsze

Rozkład liczby na czynniki pierwsze polega na zapisaniu jej w postaci iloczynu liczb pierwszych. Na przykład:

$$21 = 3 \cdot 7,$$

$$60 = 2^2 \cdot 3 \cdot 5.$$

Zgodnie z podstawowym twierdzeniem arytmetyki każdą liczbę naturalną większą od 1 można przedstawić w postaci iloczynu liczb pierwszych i taki rozkład jest jednoznaczny z dokładnością do kolejności czynników.

Dla małych liczb rozkład na czynniki pierwsze jest prosty. Dla bardzo dużych liczb staje się jednak trudny obliczeniowo. Właśnie ta trudność ma podstawowe znaczenie dla bezpieczeństwa RSA, ponieważ moduł n jest iloczynem dwóch dużych liczb pierwszych.

[2] W. Krysicki, L. Włodarski, *Wstęp do teorii liczb i kryptografii.*

2.3 Największy wspólny dzielnik i algorytm Euklidesa

Największym wspólnym dzielnikiem dwóch liczb całkowitych a i b nazywa się największą liczbę naturalną, która dzieli obie te liczby. Oznacza się go symbolem:

$$\gcd(a, b).$$

Na przykład:

$$\gcd(12, 18) = 6.$$

Jeżeli:

$$\gcd(a, b) = 1,$$

to liczby a i b nazywa się względnie pierwszymi. Pojęcie to jest ważne w algorytmie RSA, ponieważ wykładnik publiczny e musi być względnie pierwszy z wartością funkcji Eulera $\varphi(n)$.

Do obliczania największego wspólnego dzielnika wykorzystuje się algorytm Euklidesa. Polega on na wykonywaniu kolejnych dzieleń z resztą. Jest to szybka i skuteczna metoda, która ma również znaczenie praktyczne przy wyznaczaniu odwrotności modularnej.

2.4 Arytmetyka modularna

Arytmetyka modularna opiera się na działaniach wykonywanych na resztach z dzielenia. Mówimy, że liczby a i b są przystające modulo n, jeżeli przy dzieleniu przez n dają tę samą resztę. Zapisuje się to jako:

$$a \equiv b \pmod{n}.$$

Na przykład:

$$17 \equiv 5 \pmod{12},$$

ponieważ zarówno 17, jak i 5 dają resztę 5 przy dzieleniu przez 12.

Arytmetyka modularna jest podstawowym narzędziem wykorzystywanym w RSA. Zarówno szyfrowanie, jak i odszyfrowywanie wiadomości polega na potęgowaniu modulo liczba n.

2.5 Funkcja Eulera

Funkcja Eulera, oznaczana przez $\varphi(n)$, określa liczbę dodatnich liczb naturalnych nie większych od n, które są względnie pierwsze z n.

Przykładowo:

$$\varphi(8) = 4,$$

ponieważ z liczbą 8 względnie pierwsze są liczby:

$$1, 3, 5, 7.$$

Jeżeli p jest liczbą pierwszą, to:

$$\varphi(p) = p - 1.$$

Szczególnie ważny dla RSA jest przypadek, gdy:

$$n = pq,$$

gdzie p i q są różnymi liczbami pierwszymi. Wtedy:

$$\varphi(n) = (p-1)(q-1).$$

Wzór ten jest używany podczas generowania klucza prywatnego w algorytmie RSA.

2.6 Małe twierdzenie Fermata i twierdzenie Eulera

Małe twierdzenie Fermata mówi, że jeżeli p jest liczbą pierwszą oraz liczba a nie dzieli się przez p, to:

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

Na przykład dla $a = 2$ oraz $p = 7$:

$$2^6 = 64 \equiv 1 \pmod{7}.$$

Twierdzenie Eulera jest uogólnieniem tego wyniku. Jeżeli liczby a i n są względnie pierwsze, to:

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

Przykładowo, dla $a = 3$ oraz $n = 10$:

$$\varphi(10) = 4,$$

$$3^4 = 81 \equiv 1 \pmod{10}.$$

Twierdzenie Eulera ma duże znaczenie w RSA, ponieważ pozwala uzasadnić, dlaczego odszyfrowanie wiadomości odtwarza jej pierwotną postać.

3 Kryptografia i miejsce RSA w kryptografii

Kryptografia jest dziedziną zajmującą się zabezpieczaniem informacji przed dostępem osób nieuprawnionych. Jej zadaniem jest takie przekształcenie wiadomości, aby jej treść mogła zostać odczytana tylko przez osobę posiadającą odpowiedni klucz. Współcześnie kryptografia jest wykorzystywana między innymi w bankowości elektronicznej, komunikacji internetowej, podpisach cyfrowych oraz systemach logowania.

Algorytm RSA jest jednym z najbardziej znanych przykładów kryptografii asymetrycznej. Został opisany przez Ronalda Rivesta, Adiego Shamira i Leonarda Adlemana, a jego nazwa pochodzi od pierwszych liter nazwisk twórców. Jego bezpieczeństwo opiera się na trudności rozkładu dużej liczby złożonej na czynniki pierwsze.[3]

3.1 Pojęcie kryptografii

Wiadomość przed zaszyfrowaniem nazywa się tekstem jawnym. Po zastosowaniu algorytmu szyfrującego powstaje szyfrogram, czyli wiadomość zapisana w postaci niezrozumiałej dla osób, które nie mają odpowiedniego klucza.

Proces zamiany tekstu jawnego na szyfrogram nazywa się szyfrowaniem. Proces odwrotny, czyli odtworzenie pierwotnej wiadomości z szyfrogramu, nazywa się deszyfrowaniem lub odszyfrowaniem.

Kryptografia służy przede wszystkim do zapewnienia poufności informacji. W praktyce pozwala również potwierdzić tożsamość użytkownika, sprawdzić integralność danych oraz tworzyć podpisy cyfrowe.

3.2 Kryptografia symetryczna i asymetryczna

W kryptografii wyróżnia się dwa podstawowe rodzaje szyfrowania: symetryczne oraz asymetryczne.

Kryptografia symetryczna polega na tym, że ten sam klucz służy zarówno do szyfrowania, jak i do odszyfrowywania wiadomości. Nadawca i odbiorca muszą więc wcześniej znać wspólny tajny klucz. Jest to rozwiązanie szybkie, ale ma jedną istotną wadę: klucz trzeba bezpiecznie przekazać drugiej osobie.

Schemat kryptografii symetrycznej można zapisać następująco:

$$\text{tekst jawny} \xrightarrow{\text{klucz tajny}} \text{szyfrogram},$$

[3] R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, 1978, `https://people.csail.mit.edu/rivest/Rsapaper.pdf`.

$$\text{szyfrogram} \xrightarrow{\text{ten sam klucz tajny}} \text{tekst jawny}.$$

Kryptografia asymetryczna wykorzystuje dwa różne, ale matematycznie powiązane klucze: publiczny i prywatny. Klucz publiczny może być udostępniony innym osobom, natomiast klucz prywatny powinien znać tylko jego właściciel. Wiadomość zaszyfrowana kluczem publicznym może zostać odszyfrowana za pomocą odpowiadającego mu klucza prywatnego.

Schemat kryptografii asymetrycznej można zapisać w uproszczeniu:

$$\text{tekst jawny} \xrightarrow{\text{klucz publiczny}} \text{szyfrogram},$$

$$\text{szyfrogram} \xrightarrow{\text{klucz prywatny}} \text{tekst jawny}.$$

RSA należy do kryptografii asymetrycznej. Oznacza to, że korzysta z pary kluczy: publicznego i prywatnego.

3.3 Idea klucza publicznego

Idea klucza publicznego rozwiązuje problem bezpiecznego przekazywania tajnego klucza. Użytkownik może udostępnić swój klucz publiczny, a klucz prywatny zachować tylko dla siebie. Dzięki temu inne osoby mogą szyfrować wiadomości przeznaczone dla niego, ale nie mogą ich samodzielnie odszyfrować.

Przykładowo, jeżeli osoba A chce wysłać poufną wiadomość do osoby B, używa klucza publicznego osoby B. Po zaszyfrowaniu wiadomość może zostać przesłana nawet przez niezabezpieczony kanał komunikacji. Odczytać ją może tylko osoba B, ponieważ posiada odpowiedni klucz prywatny.

W algorytmie RSA klucz publiczny zawiera między innymi liczbę n, która jest iloczynem dwóch dużych liczb pierwszych. Klucz prywatny jest związany z tymi liczbami oraz z funkcją Eulera. Bezpieczeństwo RSA wynika z faktu, że rozłożenie bardzo dużej liczby n na czynniki pierwsze jest trudne obliczeniowo.

4 Budowa i działanie algorytmu RSA

Algorytm RSA jest algorytmem kryptografii asymetrycznej. Do jego działania wykorzystuje się dwa klucze: publiczny oraz prywatny. Klucz publiczny może być udostępniany innym osobom i służy do szyfrowania wiadomości. Klucz prywatny powinien pozostać tajny, ponieważ umożliwia odszyfrowanie danych.

RSA opiera się na własnościach liczb pierwszych, funkcji Eulera oraz arytmetyki modularnej. W uproszczeniu bezpieczeństwo tego algorytmu wynika z faktu, że łatwo jest pomnożyć dwie duże liczby pierwsze, ale trudno jest odtworzyć te liczby, znając tylko ich iloczyn.

4.1 Generowanie kluczy

Pierwszym etapem działania algorytmu RSA jest wygenerowanie pary kluczy. W tym celu wybiera się dwie różne liczby pierwsze:

$$p \quad \text{oraz} \quad q.$$

Następnie oblicza się:

$$n = pq.$$

Liczba n jest nazywana modułem RSA i występuje zarówno w kluczu publicznym, jak i prywatnym. Jeżeli liczby p i q są odpowiednio duże, to znajomość samego n nie wystarcza do łatwego odtworzenia tych liczb.

Kolejnym krokiem jest obliczenie wartości funkcji Eulera:

$$\varphi(n) = (p-1)(q-1).$$

Następnie wybiera się liczbę e, która spełnia warunki:

$$1 < e < \varphi(n)$$

oraz:

$$\gcd(e, \varphi(n)) = 1.$$

Liczba e jest nazywana wykładnikiem publicznym. Musi być względnie pierwsza z $\varphi(n)$, ponieważ tylko wtedy można wyznaczyć jej odwrotność modularną.

Ostatnim krokiem jest wyznaczenie liczby d, czyli wykładnika prywatnego. Liczba ta spełnia warunek:

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Oznacza to, że d jest odwrotnością modularną liczby e modulo $\varphi(n)$. W efekcie otrzymujemy:

$$\text{klucz publiczny} = (n, e),$$

$$\text{klucz prywatny} = (n, d).$$

Poniższy przykład pokazuje wygenerowanie prostych kluczy RSA w języku Python. Wartości są małe, aby obliczenia były czytelne.

```
import math

p = 11
q = 17

n = p * q
phi = (p - 1) * (q - 1)

e = 7

print("NWD(e, phi) =", math.gcd(e, phi))

d = pow(e, -1, phi)

print("p =", p)
print("q =", q)
print("n =", n)
print("phi =", phi)
print("Klucz publiczny =", (n, e))
print("Klucz prywatny =", (n, d))
```

Listing 1: Generowanie prostych kluczy RSA w Pythonie

Po uruchomieniu programu otrzymujemy:

```
NWD(e, phi) = 1
p = 11
q = 17
n = 187
phi = 160
Klucz publiczny = (187, 7)
Klucz prywatny = (187, 23)
```

Listing 2: Wynik generowania kluczy

Dla $p = 11$ oraz $q = 17$ otrzymujemy:

$$n = 187,$$

$$\varphi(n) = 160.$$

Klucz publiczny ma postać:

$$(187, 7),$$

a klucz prywatny:

$$(187, 23).$$

4.2 Klucz publiczny i klucz prywatny

Klucz publiczny w algorytmie RSA składa się z modułu n oraz wykładnika publicznego e. Może on być znany innym osobom. Każdy, kto zna klucz publiczny, może zaszyfrować wiadomość przeznaczoną dla właściciela klucza.

Klucz prywatny składa się z modułu n oraz wykładnika prywatnego d. Ten klucz musi być przechowywany w tajemnicy, ponieważ umożliwia odszyfrowanie wiadomości. W praktycznych implementacjach klucz prywatny może zawierać również dodatkowe dane, na przykład liczby p i q, które przyspieszają obliczenia.[4]

Związek między kluczem publicznym a prywatnym wynika z arytmetyki modularnej. Liczby e i d są dobrane tak, aby:

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Dzięki temu proces szyfrowania można odwrócić.

4.3 Szyfrowanie i odszyfrowanie wiadomości

Aby zaszyfrować wiadomość za pomocą RSA, trzeba najpierw przedstawić ją w postaci liczby. W praktyce tekst jest zamieniany na liczby za pomocą odpowiedniego kodowania oraz dodatkowych schematów zabezpieczających. W prostym przykładzie można założyć, że wiadomość jest już liczbą m, gdzie:

$$0 \leq m < n.$$

Szyfrowanie polega na obliczeniu szyfrogramu c według wzoru:

$$c \equiv m^e \pmod{n}.$$

Odszyfrowanie polega na obliczeniu:

$$m \equiv c^d \pmod{n}.$$

[4]K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, *PKCS #1: RSA Cryptography Specifications Version 2.2*, RFC 8017, 2016, `https://www.rfc-editor.org/rfc/rfc8017.html`.

Przyjmijmy, że wiadomość jawna ma postać:

$$m = 42.$$

```
n = 187
e = 7
d = 23

m = 42

c = pow(m, e, n)
m_odczytane = pow(c, d, n)

print("Wiadomosc␣jawna␣=", m)
print("Szyfrogram␣=", c)
print("Wiadomosc␣po␣odszyfrowaniu␣=", m_odczytane)
```

Listing 3: Szyfrowanie i odszyfrowanie wiadomości w RSA

Wynik programu jest następujący:

```
Wiadomosc jawna = 42
Szyfrogram = 15
Wiadomosc po odszyfrowaniu = 42
```

Listing 4: Wynik szyfrowania i odszyfrowania

Oznacza to, że wiadomość 42 została zaszyfrowana do postaci 15, a następnie poprawnie odszyfrowana z powrotem do wartości 42.

W Pythonie funkcja `pow(m, e, n)` oblicza wartość:

$$m^e \bmod n.$$

Jest to ważne, ponieważ w RSA operuje się na bardzo dużych liczbach. Potęgowanie modularne pozwala uniknąć bezpośredniego obliczania ogromnych potęg.

4.4 Matematyczne uzasadnienie poprawności RSA

Poprawność algorytmu RSA oznacza, że wiadomość po zaszyfrowaniu i odszyfrowaniu wraca do swojej pierwotnej postaci. Jeżeli:

$$c \equiv m^e \pmod{n},$$

to po użyciu klucza prywatnego powinno zachodzić:

$$c^d \equiv m \pmod{n}.$$

Po podstawieniu wzoru na c otrzymujemy:

$$c^d \equiv (m^e)^d \pmod{n},$$

czyli:

$$c^d \equiv m^{ed} \pmod{n}.$$

Liczby e oraz d zostały dobrane tak, że:

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Oznacza to, że istnieje liczba całkowita k, dla której:

$$ed = 1 + k\varphi(n).$$

Wtedy:

$$m^{ed} = m^{1+k\varphi(n)} = m \cdot \left(m^{\varphi(n)}\right)^k.$$

Z twierdzenia Eulera wynika, że jeżeli m i n są względnie pierwsze, to:

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

Po podstawieniu:

$$m^{ed} \equiv m \cdot 1^k \pmod{n}.$$

Ostatecznie:

$$m^{ed} \equiv m \pmod{n}.$$

Oznacza to, że proces odszyfrowania odtwarza pierwotną wiadomość. Powyższe uzasadnienie przedstawia najprostszy przypadek, gdy wiadomość m jest względnie pierwsza z modułem n. Pełne uzasadnienie poprawności RSA można przeprowadzić również z wykorzystaniem rozumowania modulo p i modulo q.

4.5 Przykład działania algorytmu RSA

Cały przykład można zebrać w następujących obliczeniach:

$$p = 11, \quad q = 17,$$

$$n = pq = 11 \cdot 17 = 187,$$

$$\varphi(n) = (p-1)(q-1) = 10 \cdot 16 = 160,$$

$$e = 7, \quad d = 23.$$

Klucz publiczny ma postać:

$$(187, 7),$$

a klucz prywatny:

$$(187, 23).$$

Dla wiadomości:

$$m = 42$$

szyfrowanie wygląda następująco:

$$c \equiv 42^7 \pmod{187},$$

$$c = 15.$$

Następnie odszyfrowanie:

$$m \equiv 15^{23} \pmod{187},$$

$$m = 42.$$

Przykład pokazuje wszystkie najważniejsze etapy działania RSA: wybór liczb pierwszych, obliczenie modułu n, wyznaczenie funkcji Eulera, utworzenie klucza publicznego i prywatnego, szyfrowanie oraz odszyfrowanie wiadomości.

4.6 Przykład ataku faktoryzacyjnego na RSA dla małych liczb

W poprzednim podrozdziale pokazano poprawne działanie algorytmu RSA dla małych liczb. Teraz można pokazać, co stanie się w sytuacji, gdy osoba nieuprawniona spróbuje odtworzyć klucz prywatny na podstawie klucza publicznego.

W RSA klucz publiczny ma postać:

$$(n, e).$$

Liczba n jest iloczynem dwóch liczb pierwszych:

$$n = pq.$$

Jeżeli uda się rozłożyć n na czynniki pierwsze, czyli znaleźć liczby p oraz q, to można obliczyć:

$$\varphi(n) = (p - 1)(q - 1).$$

Następnie można wyznaczyć wykładnik prywatny d, który spełnia warunek:

$$ed \equiv 1 \pmod{\varphi(n)}.$$

W praktyce oznacza to, że znajomość rozkładu liczby n pozwala odtworzyć klucz prywatny i odszyfrować wiadomość.

Załóżmy, że osoba atakująca zna klucz publiczny:

$$(n, e) = (187, 7)$$

oraz szyfrogram:

$$c = 15.$$

Nie zna natomiast klucza prywatnego d. Celem jest odtworzenie wiadomości jawnej.

```
import math

n = 187
e = 7
c = 15

def factorize(n):
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return i, n // i
    return None, None

p, q = factorize(n)

phi = (p - 1) * (q - 1)

d = pow(e, -1, phi)

m = pow(c, d, n)

print("Znalezione p =", p)
print("Znalezione q =", q)
print("phi =", phi)
print("Odtworzony klucz prywatny d =", d)
print("Odszyfrowana wiadomosc =", m)
```

Listing 5: Atak faktoryzacyjny na RSA dla małych liczb

Po uruchomieniu programu otrzymujemy:

```
Znalezione p = 11
Znalezione q = 17
phi = 160
Odtworzony klucz prywatny d = 23
Odszyfrowana wiadomosc = 42
```

Listing 6: Wynik ataku faktoryzacyjnego

Program znalazł rozkład:

$$187 = 11 \cdot 17.$$

Dzięki temu możliwe było obliczenie:

$$\varphi(187) = (11 - 1)(17 - 1) = 160.$$

Następnie odtworzono wykładnik prywatny:

$$d = 23.$$

Po użyciu tego klucza szyfrogram:

$$c = 15$$

został odszyfrowany do wiadomości:

$$m = 42.$$

Przykład pokazuje, że RSA można złamać, jeżeli uda się rozłożyć moduł n na czynniki pierwsze. Dla małych liczb, takich jak 187, jest to bardzo proste. Dla dużych modułów używanych w praktyce liczba możliwych prób jest jednak ogromna.

5 Zastosowanie liczb pierwszych w algorytmie RSA

W poprzednim rozdziale pokazano, że algorytm RSA można złamać, jeżeli uda się rozłożyć moduł n na czynniki pierwsze. Dla małych liczb jest to proste, co pokazuje przykład w Pythonie. W praktyce jednak RSA wykorzystuje bardzo duże liczby pierwsze, dlatego rozkład liczby n staje się zadaniem bardzo trudnym obliczeniowo.

Celem tego rozdziału jest pokazanie, jaką rolę pełnią liczby pierwsze w RSA oraz dlaczego bezpieczeństwo algorytmu zależy od trudności faktoryzacji dużych liczb.

5.1 Rola liczb pierwszych w konstrukcji RSA

Podstawą działania RSA są dwie duże liczby pierwsze oznaczane jako p oraz q. Na ich podstawie oblicza się moduł:

$$n = pq.$$

Liczba n jest częścią klucza publicznego, natomiast liczby p oraz q powinny pozostać tajne. Jeżeli ktoś pozna p i q, może obliczyć funkcję Eulera:

$$\varphi(n) = (p-1)(q-1),$$

a następnie wyznaczyć klucz prywatny d.

Właśnie dlatego bezpieczeństwo RSA zależy od tego, czy osoba atakująca potrafi odtworzyć liczby p i q, znając jedynie ich iloczyn n. Pomnożenie dwóch dużych liczb pierwszych jest łatwe, ale wykonanie działania odwrotnego, czyli rozłożenie n na czynniki pierwsze, jest znacznie trudniejsze.

5.2 Funkcja Eulera i odtworzenie klucza prywatnego

W RSA klucz publiczny ma postać:

$$(n, e),$$

a klucz prywatny jest związany z liczbą d, która spełnia warunek:

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Aby obliczyć d, trzeba znać wartość $\varphi(n)$. Dla modułu RSA, gdzie:

$$n = pq,$$

wartość funkcji Eulera wynosi:

$$\varphi(n) = (p-1)(q-1).$$

Oznacza to, że znajomość liczb p i q pozwala obliczyć $\varphi(n)$, a następnie odtworzyć klucz prywatny. Dlatego w RSA liczby pierwsze nie mogą być małe ani łatwe do odgadnięcia.

Dla przykładu:

$$n = 187 = 11 \cdot 17.$$

Wtedy:

$$\varphi(187) = (11-1)(17-1) = 160.$$

Znając tę wartość, można wyznaczyć klucz prywatny, tak jak pokazano w poprzednim rozdziale.

5.3 Trudność faktoryzacji dużych liczb

Faktoryzacja oznacza rozkład liczby na czynniki pierwsze. W przypadku algorytmu RSA chodzi o rozłożenie modułu

$$n = pq,$$

gdzie p oraz q są dużymi liczbami pierwszymi. Moduł n jest częścią klucza publicznego, natomiast liczby p i q muszą pozostać tajne. Jeżeli osoba atakująca pozna rozkład n, może obliczyć

$$\varphi(n) = (p-1)(q-1),$$

a następnie wyznaczyć wykładnik prywatny d. Oznacza to, że skuteczna faktoryzacja modułu RSA prowadzi do odtworzenia klucza prywatnego.

W tym miejscu trzeba doprecyzować, co oznacza określenie „duże liczby". W kryptografii nie chodzi o liczby duże w sensie potocznym, na przykład kilkanaście albo kilkadziesiąt cyfr. Długość modułu RSA podaje się zwykle w bitach. Moduł 512-bitowy ma około 155 cyfr dziesiętnych, moduł 1024--bitowy około 309 cyfr dziesiętnych, a moduł 2048-bitowy około 617 cyfr dziesiętnych. W praktycznych zastosowaniach liczby pierwsze p i q mają więc po kilkaset cyfr dziesiętnych.

Najprostsza metoda faktoryzacji polega na sprawdzaniu kolejnych możliwych dzielników. Nie trzeba sprawdzać wszystkich liczb aż do n, ponieważ wystarczy sprawdzać dzielniki do $\sqrt{n}$. Dla małego przykładu

$$187 = 11 \cdot 17$$

taka metoda działa natychmiast. Jednak przy dużych modułach RSA sprawdzanie kolejnych dzielników jest zbyt wolne i nie opisuje rzeczywistych metod ataku na RSA. Tę metodę stosuje się głównie wtedy, gdy liczy się na znalezienie małego dzielnika.

Znacznie skuteczniejszą metodą jest metoda rho Pollarda. Jest to algorytm probabilistyczny, prosty w implementacji i dużo szybszy od sprawdzania kolejnych dzielników. Jeżeli najmniejszy czynnik pierwszy liczby n oznaczymy przez p, to metoda rho Pollarda znajduje dzielnik w czasie rzędu

$$O(\sqrt{p}).$$

W typowym module RSA czynniki p i q mają podobną wielkość, więc można przyjąć w przybliżeniu, że

$$p \approx \sqrt{n}.$$

Wtedy czas działania metody rho Pollarda jest rzędu

$$O(n^{1/4}).$$

Jest to bardzo duża poprawa względem sprawdzania dzielników, którego czas działania jest rzędu

$$O(n^{1/2}).$$

Z tego powodu samo stwierdzenie, że „dla dużych liczb trzeba sprawdzić bardzo wiele dzielników", jest zbyt dużym uproszczeniem. W praktyce liczby mające rząd kilkudziesięciu cyfr dziesiętnych nie dają bezpieczeństwa kryptograficznego, ponieważ można je często rozłożyć metodą rho Pollarda albo innymi metodami faktoryzacji.

5.4 Bardziej zaawansowane metody faktoryzacji

Dla większych liczb stosuje się metody skuteczniejsze niż sprawdzanie dzielników i metoda rho Pollarda. Jedną z nich jest metoda krzywych eliptycznych, oznaczana skrótem ECM. Metoda ta jest szczególnie przydatna wtedy, gdy liczba złożona ma czynnik pierwszy o umiarkowanej wielkości. Jej skuteczność zależy głównie od rozmiaru znajdowanego czynnika, a nie tylko od rozmiaru całej liczby n.

Do faktoryzacji bardzo dużych liczb ogólnej postaci, takich jak moduły RSA, stosuje się jeszcze bardziej zaawansowane algorytmy. Najważniejszą klasyczną metodą jest sito w ciele liczbowym, czyli Number Field Sieve. Dla dużych liczb ogólnej postaci jego wersja ogólna, General Number Field Sieve, jest obecnie najlepszą znaną klasyczną metodą faktoryzacji.

Metody tego typu są znacznie szybsze od prostego sprawdzania dzielników, ale wymagają bardzo dużych zasobów. Potrzebna jest nie tylko duża liczba operacji, lecz także duża ilość pamięci oraz możliwość wykonywania obliczeń równolegle. Dlatego łamanie dużych kluczy RSA nie jest zwykle zadaniem dla pojedynczego zwykłego komputera. Może być natomiast realnym celem dla państw, dużych instytucji albo dobrze finansowanych organizacji, które dysponują kosztownym sprzętem i odpowiednim oprogramowaniem.

Historia RSA pokazuje, że granica tego, co uznaje się za bezpieczne, zmienia się wraz z rozwojem algorytmów i sprzętu. Moduły RSA o długości 512 bitów zostały już rozłożone na czynniki, dlatego obecnie są uznawane za całkowicie niebezpieczne. Również klucze 1024-bitowe nie są obecnie zalecane do nowych zastosowań. Za minimalny praktyczny rozmiar klucza RSA przyjmuje się obecnie 2048 bitów, natomiast dla wyższego poziomu bezpieczeństwa stosuje się 3072 bity lub więcej.

5.5 Znaczenie długości klucza i doboru liczb pierwszych

Długość klucza ma bardzo duże znaczenie dla bezpieczeństwa RSA. Im większy jest moduł n, tym trudniej rozłożyć go na czynniki pierwsze. Nie wystarczy jednak powiedzieć, że liczby p i q mają być „duże”. Trzeba określić ich rozmiar w bitach i porównać go z aktualnym stanem wiedzy o faktoryzacji.

W przykładach edukacyjnych często używa się małych liczb, takich jak

$$p = 11, \qquad q = 17.$$

Wtedy

$$n = 187.$$

Taki przykład jest dobry do zrozumienia działania algorytmu, ale nie daje żadnego praktycznego bezpieczeństwa. Liczbę 187 można rozłożyć na czynniki natychmiast.

W rzeczywistych zastosowaniach liczby p i q muszą być generowane losowo, muszą być odpowiednio duże i nie mogą być przewidywalne. Powinny mieć podobną długość bitową, ale nie powinny być zbyt blisko siebie, ponieważ wtedy mogą pojawić się dodatkowe ataki. Ważne jest również, aby generator liczb losowych był bezpieczny. Jeżeli liczby pierwsze zostaną źle dobrane, bezpieczeństwo RSA może zostać osłabione nawet wtedy, gdy sam moduł ma dużą długość.

Wniosek jest więc następujący: bezpieczeństwo RSA nie wynika z tego, że faktoryzacja jest matematycznie niemożliwa. Wynika ono z tego, że dla odpowiednio dużych i poprawnie wygenerowanych modułów najlepsze znane

klasyczne metody faktoryzacji wymagają zbyt dużej ilości czasu, pamięci i mocy obliczeniowej.

5.6 Znaczenie długości klucza i doboru liczb pierwszych

Długość klucza ma bardzo duże znaczenie dla bezpieczeństwa RSA. Im większy jest moduł n, tym trudniej rozłożyć go na czynniki pierwsze. Jeżeli n jest zbyt małe, możliwe jest odtworzenie liczb p oraz q, a następnie wyznaczenie klucza prywatnego.W praktyce nie wystarczy powiedzieć, że liczby pierwsze mają być „duże". Trzeba określić ich rozmiar w bitach. Historycznie używano krótszych modułów RSA, na przykład 512-bitowych i 1024-bitowych, ale wraz z rozwojem metod faktoryzacji przestały one zapewniać odpowiedni poziom bezpieczeństwa. Obecnie 512-bitowy RSA należy uznać za całkowicie niebezpieczny, a 1024-bitowy za niewystarczający dla nowych zastosowań. Minimalnym rozsądnym rozmiarem we współczesnych zastosowaniach jest 2048 bitów, a dla dłuższej perspektywy często rozważa się 3072 bity lub więcej.

Należy także pamiętać, że liczby (p) i (q) powinny być generowane losowo, powinny mieć podobną długość, ale nie mogą być zbyt blisko siebie. Jeżeli są źle dobrane, na przykład zbyt małe, przewidywalne albo zbyt podobne, mogą pojawić się dodatkowe ataki, które osłabiają bezpieczeństwo RSA niezależnie od samej długości modułu.

W przykładach edukacyjnych często używa się małych liczb, takich jak:

$$p = 11, \quad q = 17.$$

Wtedy:

$$n = 187.$$

Taki przykład jest dobry do zrozumienia algorytmu, ale nie zapewnia żadnego praktycznego bezpieczeństwa.

W rzeczywistych zastosowaniach liczby p oraz q muszą być bardzo duże, losowe i trudne do przewidzenia. Nie powinny być zbyt blisko siebie ani generowane w sposób powtarzalny. Jeżeli liczby pierwsze zostaną źle dobrane, bezpieczeństwo RSA może zostać osłabione.

Współczesne specyfikacje RSA opisują nie tylko same działania matematyczne, ale także praktyczne schematy szyfrowania, podpisu cyfrowego i reprezentacji kluczy. Pokazuje to, że w zastosowaniach praktycznych RSA nie jest używane wyłącznie jako proste działanie $m^e \bmod n$, lecz wymaga dodatkowych zasad implementacyjnych i zabezpieczeń.[5]

[5]K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, *PKCS #1: RSA Cryptography Specifica-*

5.7 Komputery kwantowe a bezpieczeństwo RSA

W kontekście bezpieczeństwa RSA ważne są również komputery kwantowe. Istnieje algorytm Shora, który teoretycznie pozwala rozkładać duże liczby na czynniki pierwsze dużo szybciej niż klasyczne algorytmy. Oznacza to, że odpowiednio duży i stabilny komputer kwantowy mógłby w przyszłości zagrozić bezpieczeństwu RSA.

Nie oznacza to jednak, że obecne komputery kwantowe potrafią praktycznie łamać duże klucze RSA. Do takiego ataku potrzeba bardzo wielu stabilnych kubitów oraz skutecznej korekcji błędów. Kubity są wrażliwe na zakłócenia, dlatego do uzyskania jednego niezawodnego kubitu logicznego potrzeba wielu kubitów fizycznych.

Według oszacowań Gidneya i Ekerå faktoryzacja 2048-bitowej liczby RSA w ciągu około 8 godzin mogłaby wymagać około 20 milionów zaszumionych kubitów fizycznych przy określonych założeniach technologicznych.[6]

Z tego powodu komputery kwantowe są ważnym zagrożeniem dla RSA w przyszłości, ale obecnie nie są jeszcze praktycznym narzędziem do masowego łamania dużych kluczy. Instytucje zajmujące się bezpieczeństwem rozwijają kryptografię postkwantową, ponieważ odpowiednio silne komputery kwantowe mogłyby w przyszłości osłabić wiele obecnie stosowanych systemów kryptograficznych.[7]

Podsumowując, liczby pierwsze są fundamentem algorytmu RSA. To z nich buduje się moduł n, a trudność odtworzenia tych liczb na podstawie samego iloczynu stanowi podstawę bezpieczeństwa całego algorytmu. Dla małych liczb RSA można złamać łatwo, ale dla dużych modułów problem faktoryzacji wymaga tak dużego czasu i zasobów, że klasyczny atak staje się praktycznie niewykonalny.

tions Version 2.2, RFC 8017, 2016, `https://www.rfc-editor.org/rfc/rfc8017.html`.

[6] C. Gidney, M. Ekerå, *How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits*, Quantum 5, 433, 2021, `https://quantum-journal.org/papers/q-2021-04-15-433/`.

[7] National Institute of Standards and Technology, *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*, 2024, `https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards`.

6 Zastosowania RSA w praktyce i ograniczenia algorytmu

Algorytm RSA przez wiele lat był jednym z najważniejszych algorytmów kryptografii asymetrycznej. Jego znaczenie wynika z tego, że pozwala na bezpieczne wykorzystanie klucza publicznego oraz prywatnego. Dzięki temu RSA może być używane między innymi do szyfrowania danych, wymiany kluczy oraz tworzenia podpisów cyfrowych.

6.1 Praktyczne zastosowania RSA

Jednym z ważnych zastosowań RSA są podpisy cyfrowe. Podpis cyfrowy pozwala potwierdzić, że dana wiadomość lub dokument pochodzi od określonego nadawcy oraz że nie został zmieniony. W takim zastosowaniu klucz prywatny służy do utworzenia podpisu, a klucz publiczny do jego weryfikacji.

RSA może być również wykorzystywane w certyfikatach cyfrowych. Certyfikaty pomagają potwierdzić tożsamość stron komunikacji, na przykład serwera internetowego. Dzięki temu użytkownik może upewnić się, że łączy się z właściwą stroną, a nie z podszywającym się pod nią atakującym.

W praktyce RSA często nie służy do szyfrowania dużych wiadomości bezpośrednio. Wynika to z tego, że operacje RSA są wolniejsze niż szyfrowanie symetryczne. Często stosuje się więc rozwiązanie hybrydowe: RSA służy do zabezpieczenia lub wymiany klucza, a właściwe dane szyfrowane są szybszym algorytmem symetrycznym.

6.2 Zalety i ograniczenia algorytmu

Główną zaletą RSA jest możliwość używania dwóch różnych kluczy: publicznego i prywatnego. Dzięki temu nie trzeba wcześniej przekazywać tajnego klucza każdej osobie, z którą chce się bezpiecznie komunikować. Klucz publiczny może być udostępniony, a klucz prywatny pozostaje tajny.

Kolejną zaletą RSA jest jego silne oparcie na matematyce. Algorytm wykorzystuje dobrze znane pojęcia teorii liczb, takie jak liczby pierwsze, funkcja Eulera oraz arytmetyka modularna. Dzięki temu można dokładnie opisać jego działanie i uzasadnić poprawność matematycznie.

RSA ma jednak także ograniczenia. Przede wszystkim wymaga stosowania odpowiednio długich kluczy. Jeżeli klucz jest za krótki, moduł n może zostać rozłożony na czynniki pierwsze. Wtedy możliwe jest odtworzenie klucza prywatnego.

Drugim ograniczeniem jest wydajność. Operacje wykonywane w RSA są bardziej kosztowne obliczeniowo niż operacje w wielu algorytmach symetrycznych. Dlatego RSA nie jest najlepszym rozwiązaniem do bezpośredniego szyfrowania dużych ilości danych.

Trzecim ograniczeniem są błędy implementacyjne. Sam algorytm matematyczny nie wystarcza do zapewnienia bezpieczeństwa. W praktyce konieczne jest stosowanie odpowiednich schematów dopełniania, poprawnego generowania liczb pierwszych oraz bezpiecznego przechowywania klucza prywatnego.[8]

6.3 Współczesne znaczenie RSA

Mimo rozwoju nowych metod kryptograficznych RSA nadal ma duże znaczenie historyczne, teoretyczne i praktyczne. Jest jednym z najważniejszych przykładów wykorzystania teorii liczb w ochronie informacji. Pokazuje również, jak abstrakcyjne pojęcia matematyczne, takie jak liczby pierwsze i arytmetyka modularna, mogą mieć bezpośrednie zastosowanie w informatyce.

Współcześnie coraz większą uwagę zwraca się jednak na kryptografię postkwantową. Wynika to z faktu, że odpowiednio duże komputery kwantowe mogłyby w przyszłości zagrozić algorytmom opartym na faktoryzacji, w tym RSA. Z tego powodu rozwijane są nowe algorytmy, które mają być odporne na ataki z wykorzystaniem komputerów kwantowych.

RSA pozostaje więc bardzo ważnym algorytmem do zrozumienia podstaw kryptografii asymetrycznej, ale jego przyszłość zależy od rozwoju technologii obliczeniowych oraz od przejścia na rozwiązania odporne na ataki kwantowe.

[8] K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, *PKCS #1: RSA Cryptography Specifications Version 2.2*, RFC 8017, 2016, `https://www.rfc-editor.org/rfc/rfc8017.html`.

7 Zakończenie

Celem pracy było przedstawienie zastosowania liczb pierwszych w algorytmie RSA oraz wyjaśnienie, dlaczego są one podstawą bezpieczeństwa tego systemu. W pracy omówiono najważniejsze pojęcia z teorii liczb, takie jak liczby pierwsze, rozkład na czynniki pierwsze, największy wspólny dzielnik, arytmetyka modularna oraz funkcja Eulera. Następnie przedstawiono ideę kryptografii asymetrycznej oraz sposób działania algorytmu RSA.

Pokazano, że w algorytmie RSA wybiera się dwie duże liczby pierwsze p oraz q, a następnie tworzy się ich iloczyn:

$$n = pq.$$

Liczba n jest publiczna, natomiast liczby p oraz q muszą pozostać tajne. Jeżeli osoba atakująca potrafi rozłożyć n na czynniki pierwsze, może obliczyć:

$$\varphi(n) = (p-1)(q-1),$$

a następnie odtworzyć klucz prywatny.

W pracy pokazano również prosty przykład ataku faktoryzacyjnego w języku Python. Dla małego modułu $n = 187$ program szybko znalazł rozkład:

$$187 = 11 \cdot 17.$$

Dzięki temu możliwe było obliczenie funkcji Eulera, odtworzenie klucza prywatnego i odszyfrowanie wiadomości. Przykład ten pokazuje, że bezpieczeństwo RSA zależy bezpośrednio od trudności faktoryzacji.

Najważniejszy wniosek jest taki, że RSA nie jest bezpieczne dlatego, że rozkład liczby na czynniki pierwsze jest matematycznie niemożliwy. Jest bezpieczne dlatego, że dla odpowiednio dużych liczb wykonanie takiego rozkładu jest praktycznie niewykonalne przy użyciu dostępnych klasycznych metod obliczeniowych. W przypadku dużych modułów RSA problemem staje się ogromna liczba operacji, czas obliczeń oraz zasoby potrzebne do przeprowadzenia ataku.

Omówiono także znaczenie komputerów kwantowych. Algorytm Shora pokazuje, że odpowiednio silny komputer kwantowy mógłby w przyszłości zagrozić RSA. Obecnie jednak komputery kwantowe nie posiadają jeszcze wystarczających zasobów, aby praktycznie łamać duże klucze RSA na masową skalę. Z tego powodu RSA nadal ma duże znaczenie, ale równocześnie rozwijana jest kryptografia postkwantowa.

Podsumowując, liczby pierwsze są fundamentem algorytmu RSA. To dzięki nim możliwe jest utworzenie kluczy, a jednocześnie trudność odtworzenia

tych liczb na podstawie ich iloczynu chroni cały system. Algorytm RSA jest więc dobrym przykładem praktycznego zastosowania teorii liczb w bezpieczeństwie informacji.

8 Bibliografia

1. W. Krysicki, L. Włodarski, *Wstęp do teorii liczb i kryptografii*.

2. R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, 1978, `https://people.csail.mit.edu/rivest/Rsapaper.pdf`.

3. K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, *PKCS #1: RSA Cryptography Specifications Version 2.2*, RFC 8017, 2016, `https://www.rfc-editor.org/rfc/rfc8017.html`.

4. C. Gidney, M. Ekerå, *How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits*, Quantum 5, 433, 2021, `https://quantum-journal.org/papers/q-2021-04-15-433/`.

5. National Institute of Standards and Technology, *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*, 2024, `https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quan`