

logowei.jpg

Systemy kryptograficzne

Projekt

Kryptosystem plecakowy Merklego i Hellmana

Author: Jakub Mączka, Marcel Majcher

14 marca 2024

Spis treści

1	Problem plecakowy	3
2	Rozwój kryptosystemów plecakowych	4
3	Kryptosystem plecakowy Merklego i Hellmana	5
3.1	Generowanie kluczy	6
3.2	Szyfrowanie	6
3.3	Deszyfrowanie	7
4	Implementacja programistyczna kryptosystemu	8
4.1	Implementacja mechanizmu generowania kluczy	8
4.2	Implementacja mechanizmów szyfrowania i deszyfrowania	9
4.3	Testy poprawności działania systemu	10
5	Podsumowanie	11

1 Problem plecakowy

Problem plecakowy dotyczy w większości przypadków systemów architektonicznie wzorujących się na systemie Merklego-Hellmana. Samo nazewnictwo tego zagadnienia budzi jednak wiele kontrowersji. Niektórzy autorzy uważają, że nazwa kryptosystem plecakowy, odnosząca się do wymienionego w tytule rozdziału problemu, jest błędna. Przyczyną takiego stanowiska jest fakt, iż kryptosystem ten oparty jest na zagadnieniach dotyczących sumy podzbiorów. Temat ten porusza w swoim artykule Douglas R. Stinson. Formułuje on definicję problemu plecakowego jako zwykle odnoszącego się do wyboru obiektów o określonych wartościach parametrów wagowych i zysku, w taki sposób, aby nie przekroczyć określonej wartości pojemności i osiągnąć oczekiwany, docelowy zysk. Rozważania Stinsona można zapisać za pomocą zależności (1) i (2), będących funkcjami sumy. [2]

$$\sum_{i=1}^n p_i \cdot w_i \quad (1)$$

gdzie:

p_i jest wartością zysku dla danego obiektu

w_i jest zmienną odpowiadającą wadze

n stanowi liczbę wszystkich obiektów - np. liczb składających się na klucz publiczny

$$\sum_{i=1}^n w_i \cdot x_i \leq c \quad (2)$$

gdzie:

x_i stanowi zmienną binarną

c jest zmienną odpowiadającą pojemności

Rola x_i w procesie opisanym za pomocą wzoru (2) polega na określaniu czy dany obiekt został wybrany do dalszych przekształceń - wartość 1. Natomiast zmienna c dotyczy wartości maksymalnej, której podczas rozwiązywania problemu plecakowego nie można przekraczać. Często spotykana jest też zgoła inna definicja problemu sumy podzbiorów od przedstawionej w tym rozdziale. W niektórych opracowaniach na ten temat występuje skończony zbiór liczb naturalnych, a problem polega na znalezieniu takiego podzbioru tego zbioru, którego elementy sumują się do wartości pewnej liczby naturalnej. Zdarzają się też opinie, dotyczące postrzegania problemu sumy podzbiorów jako szczególnej formy problemu plecakowego opisanego przez Stinsona, w którym wartości zysków oraz wartości parametrów wagowych są równe dla wszystkich obiektów. Sam Merkle i Hellman w swojej pracy bardziej opowiadali się za wersją przedstawioną z użyciem wzorów (1) i (2) i to właśnie na niej oparli swój kryptosystem. Rozwiązuje on problem, w którym biorąc pod uwagę zbiór liczb całkowitych $A = \{a_1, a_2, \dots, a_n\}$ oraz liczbę całkowitą S należy zdecydować czy istnieje podzbiór A , który sumuje się dokładnie do S . [3]

Analizując różne podejścia naukowe do tego typu zagadnień nie można jednoznacznie sformułować zasady ustanawiającej konieczność opisu tych zagadnień jako problem plecakowy, czy też problem sumy podzbiorów. Jedynym wyjściem z sytuacji jest dogłębna analiza danego przypadku użycia, włącznie z danymi wejściowymi, czy też wartościami docelowymi, które powinny zostać osiągnięte i na podstawie tego procesu określenie, czy mamy do czynienia z problemem plecakowym, czy problemem sumy podzbiorów. [1]

W literaturze można znaleźć różnorodne rozwinięcia pierwotnego problemu plecakowego. Jedną z tych modyfikacji jest nazywana plecakiem kompaktowym (eng. compact knapsack). Roszerzenie dotyczy tu wektora binarnego x_i , tj. wektora zmiennych które mogą przyjmować wartości 0 lub 1. W przypadku plecaka kompaktowego, x_i może przyjmować wartości ze zbioru $\{0, 1, 2, \dots, 2b - 1\}$, gdzie b jest dodatnią liczbą całkowitą. Zastosowanie tego typu rozwiązania przy jednoczesnym zachowaniu przeznaczenia kryptosystemu wymaga wprowadzenia zmian w jego algorytmie. [4]

2 Rozwój kryptosystemów plecakowych

Historia kryptosystemów plecakowych od czasu powstania, aż do ich ostatecznego skompromitowania odznacza się wieloma wydarzeniami bez precedensu. Te najważniejsze, uszeregowano zgodnie z chronologią i przedstawiono poniżej. [1]

- **1978 r.** - opracowanie przez Merklego i Hellmana kryptosystemu plecakowego, kilka miesięcy po publikacji pierwszego kryptosystemu klucza publicznego. Merkle i Hellman opisali wersję z pojedynczą i wielokrotną iteracją.
- **1982 r.** - udany atak Shamira na wersję kryptosystemu plecakowego z pojedynczą iteracją i modyfikacją oryginalnego schematu.
- **1984 r.** - atak na kryptosystem plecakowy w wersji wielokrotnej iteracji przez Ernesta F. Brickell'a. System został złamany w 40 iteracji, w czasie około 60 minut.
- **1985 r.** - publikacja Rodneya M.F. Godmana oraz AJ'a McAuley'a modułowego kryptosystemu plecakowego, opartego na chińskim twierdzeniu o resztach.
- **1985 r.** - opracowanie przez Józefa Pieprzyka kryptosystemu plecakowego opartego na wielomianach. System ten można złamać za pomocą największego wspólnego dzielnika niektórych składników klucza publicznego.
- **1986 r.** - publikacja Haralda Niederreitera na temat kryptosystemu plecakowego wykorzystującego algebraiczną teorię kodowania.
- **1987 r.** - opracowanie przez Józefa Vyskoca schematu, w którym to kryptosystem szyfruje każdy bit wiadomości za pomocą dwóch liczb całkowitych, z których każda jest sumą losowo wybranych elementów wektora ładunku.
- **1988 r.** - opracowanie przez Masakatu Morii oraz Masao Kasaharę kryptosystemu wykorzystującego logarytm dyskretny. Jak dotąd nie odnotowano udanych prób złamania tego kryptosystemu.
- **1989 r.** - publikacja Chi-Sunga na temat ulepszenia schematu Merkle-Hellmana. Ulepszenia dotyczyły zastosowania metody przesunięcia liniowego w celu poprawy bezpieczeństwa kryptosystemu.
- **1990 r.** - opracowanie przez Valtteri Niemi'ego kryptosystemu opartego na plecakach modularnych. Kryptosystem został złamany w roku następnym.
- **1991 r.** - Hussain Ali Hussain, Jafar Wadi Abdul Sada i Saad M. Kalipha opracowali wieloetapowy system typu trapdoor knapsack, w którym dane wyjściowe (szyfrogram) każdego etapu są traktowane jako dane wejściowe (tekst jawny) następnego etapu. Nie jest znany żaden skuteczny atak na ten schemat.
- **1996 r.** - K. Kobayashi i M. Kimura opracowali zmodyfikowany kryptosystem plecakowy, oparty na nowej koncepcji, zgodnie z którą nadawcy może wybrać klucze szyfrujące posługując się danym mu zestawem kluczy szyfrujących.
- **1997 r.** - Kouichi Itoh, Eiji Okamoto i Masahiro Mambo opracowali kryptosystem klucza publicznego, w którym deszyfrowanie postrzegane było jako multiplikatywny problem plecakowy. Niedługo potem na system przeprowadzony został atak, który udowodnił, że w praktyce klucz prywatny można uzyskać z klucza publicznego w mniej niż 10 minut.
- **2001 r.** - Shinya Kiuchi, Yasuyuki Murakami i Masao Kasahara ulepszyli multiplikatywny system plecakowy przy użyciu algorytmu Schalkwijka, [1]

3 Kryptosystem plecakowy Merklego i Hellmana

Algorytm szyfrowania plecakowego to kryptosystem z kluczem asymetrycznym. Do komunikacji wymagane są dwa klucze - publiczny i prywatny. Proces szyfrowania polega na przekształceniu wiadomości w postaci tekstu jawnego do postaci zaszyfrowanej z użyciem klucza publicznego. Deszyfrowanie odbywa się przy użyciu odpowiedniego klucza prywatnego w celu odzyskania oryginalnego tekstu jawnego.

Algorytm w głównej mierze opiera się na przekształceniu wiadomości w serię wielu bitów, które następnie się mnożone przez inną sekwencję generowaną z użyciem super rosnących liczb całkowitych (eng. super-increasing integers). W ten sposób powstaje zaszyfrowany kod, który może zostać odszyfrowany jedynie poprzez odtworzenie obliczeń, do których niezbędna jest wiedza na temat czynników pierwszych lub zaznajomienie z innymi technikami kryptograficznymi. Samo odszyfrowanie nie jest możliwe bez znajomości klucza prywatnego. Pewne cechy algorytmu szyfrowania plecakowego są unikalne na tle innych algorytmów szyfrowania. Tabela 1. przedstawia porównanie w/w algorytmu z innymi popularnymi metodami o podobnym przeznaczeniu. [5]

Tabela 1: Porównanie algorytmów szyfrujących [5]

Algorytm szyfrujący	Typ szyfrowania	Bezpieczeństwo	Szybkość	Implementacja
Merkle-Hellman	asymetryczne	Podatny na algorytm LLL	Wolny	W przeszłości
RSA	asymetryczne	Dla dużych rozmiarów klucza	Wolny	SSL/TLS
AES	symetryczne	Odporny na znane ataki	Szybki	Szyfrowanie plików
DES	symetryczne	Podatny na atak brute-force	Szybki	Zastąpiony AES'em

Tabela 1. w dobitny sposób pokazuje, iż chociaż algorytm szyfrowania plecakowego był rewolucyjny w pewnym okresie historii, to jednak został on zdystansowany przez inne metody szyfrowania, głównie AES i DES w wielu płaszczyznach, dotyczących bezpieczeństwa, szybkości, implementacji. Zwłaszcza kwestia implementacji wydaje się w dzisiejszych czasach być problemem - próżno szukać miejsca, gdzie ten kryptosystem miałby zastosowanie.

Szyfrowanie plecakowe wykazuje pewne cechy, które czynią go silną metodą szyfrującą. Jednym z głównych atutów jest złożoność procesów generowania klucza - wykorzystywana jest duża sekwencja super-rosnąca, która jest trudna do odtworzenia bez znajomości prawidłowej sumy podzbiorów. Fakt, że każda wiadomość ma swój własny losowy klucz prywatny stanowi kolejny aspekt ochronny, ponieważ uniemożliwia to atakującym wykorzystanie ataków z użyciem znanego tekstu jawnego. Szyfrowanie plecakowe zapewnia także wysoki poziom poufności wiadomości. Zaszyfrowanie jednego bitu w oryginalnej wiadomości powoduje, że ponad połowa zaszyfrowanych pozostałych bitów zmienia się w sposób losowy. Atak typu brute-force na ten typ systemu kryptograficznego nie jest stosowany z uwagi na dużą przestrzeń liczbowa i wiążącą się z tym ogromną potrzebną mocą obliczeniową i szerokimi ramami czasowymi, niezbędnymi do osiągnięcia celu.

Opisując mocne strony algorytmu szyfrowania plecakowego, nie można zapominać o jego potencjalnych podatnościach. Jedną z nich dotyczy generowania klucza publicznego. Jeśli atakujący określi w prawidłowy sposób podzbiór sekwencji super-rosnącej, może złamać system szyfrowania. Dodatkowo należy pamiętać o aspektach implementacyjnych - użycie generatorów liczb losowych o wąskim zakresie, czy też słabej jakości implementacja może ułatwić złamanie kryptosystemu. Aby zaradzić tym lukom i słabościom, eksperci sugerują staranne wdrażanie kryptosystemów plecakowych i regularne ich aktualizowanie w miarę pojawiania się kolejnych, nowych metod ataku. Pomimo obaw związanych z zagrożeniami bezpieczeństwa dotyczącymi algo-

rytmów szyfrowania plecakowego, pozostają one ważnymi narzędziami dla specjalistów ds. bezpieczeństwa informacji - oczywiście, jeśli są odpowiednio zaimplementowane i zabezpieczone przed znanymi zagrożeniami. [5]

3.1 Generowanie kluczy

Proces generowania kluczy można przedstawić za pomocą poniższej listy kroków: [1]

1. Wybór wartości zmiennej n , odpowiadającej rozmiarowi bloku. Określenie wartości tej zmiennej determinuje późniejszą maksymalną liczbę bitów na które może składać się szyfrowana liczba.
2. Wybór losowej sekwencji super-rosnącej. Sprowadza się to do wylosowania n liczb, będących dodatnimi liczbami całkowitymi, składających się na wektor W . Etap ten można przedstawić za pomocą zależności (3), (4) i (5).

$$W = (w_1, w_2, \dots, w_n) \quad (3)$$

$$w_k > \sum_{i=1}^{k-1} w_i \quad (4)$$

gdzie k spełnia następujący warunek:

$$1 < k \leq n \quad (5)$$

3. Wybór wartości zmiennej q , będącej losową liczbą całkowitą, większą niż suma wszystkich wartości obiektów sekwencji super-rosnącej.

$$q > \sum_{i=1}^n w_i \quad (6)$$

4. Wybór wartości zmiennej r , będącej losową liczbą całkowitą, względnie pierwszą z q , co oznacza, że ich największy wspólny dzielnik wynosi 1.
5. Klucz prywatny stanowi listę elementów (W, q, r)
6. Klucz publiczny ma postać następującego wektora:

$$B = (b_1, b_2, \dots, b_n) \quad (7)$$

gdzie każdy element obliczany jest za pomocą wzoru (8)

$$b_i = rw_i \bmod q \quad (8)$$

3.2 Szyfrowanie

Proces szyfrowania można przedstawić w następujących etapach: [1]

1. Utworzenie zmiennej m będącej n - bitową wiadomością o strukturze:

$$m = (m_1, m_2, \dots, m_n) \quad (9)$$

gdzie m_1 reprezentuje najstarszy bit

2. Wybór wartości b_i z klucza publicznego, zgodnie z wartością indeksu iteracyjnego dla rozpatrywanego bitu m_i , a następnie moltiplicacja wartości 0-1 i wartości liczbowej.
3. Utworzenie zmiennej c reprezentującej szyfrogram i nadanie jej wartości równej sumie obliczonych w kroku 2. wyrażeń.

$$c = \sum_{i=1}^n m_i b_i \quad (10)$$

3.3 Deszyfrowanie

Proces deszyfrowania dotyczy znalezienia takiego podzbioru B , który zsumowany da wartość c . Jest to realizowane według poniższej listy kroków: [1]

1. Utworzenie zmiennej r' będącej modularną odwrotnością r modulo q , której wartość obliczana jest za pomocą rozszerzonego algorytmu Euklidesa, zgodnie z poniższą zależnością:

$$r' := r^{-1} \pmod{q} \quad (11)$$

gdzie konieczne jest spełnienie warunku $\text{NWD}(q, r) = 1$

2. Utworzenie i obliczenie wartości zmiennej c' , będącej przekształceniem szyfrogramu.

$$c' := cr' \pmod{q} \quad (12)$$

3. Rozwiązanie problemu sumy podzbiorów dla zmiennej c' z użyciem liczb wchodzących w skład wektora super-rosnącego W . Spełniona musi być zależność (13)

$$c' = \sum_{i=1}^k w_{x_i} \quad (13)$$

gdzie x_i to elementy listy X zawierającej indeksy, pod którymi znajdują się wartości z sekwencji super-rosnącej, które zsumowane są równe zmiennej c' . Utworzenie tej listy indeksów można przedstawić w pięciu krokach.

1. Inicjalizacja pustej listy X .
 2. Znalezienie największego elementu z wektora super-rosnącego W , ale jednocześnie o wartości mniejszej lub równej c' i oznaczenie go jako w_j .
 3. Wykonanie operacji odejmowania:

$$c' := c' - w_j \quad (14)$$
 4. Dodanie indeksu j do listy X
 5. Sprawdzenie, czy c' jest większe od zera, jeśli tak to powrót do kroku 2.
4. Deszyfracja wiadomości zaszyfrowanej z użyciem operacji sumy ze wzoru (15)

$$m = \sum_{i=1}^k 2^{n-x_i} \quad (15)$$

4 Implementacja programistyczna kryptosystemu

Kryptosystem został zaimplementowany przy użyciu języka Python w wersji 3.11.3. Dokonuje on szyfrowania i deszyfrowania wiadomości zapisywanych w systemie binarnym. Podczas tworzenia kryptosystemu wykorzystano dwie biblioteki:

- **random** - użyta do generowania liczb losowych
- **math** - użyta do wyznaczania wartości z użyciem operacji matematycznych, takich jak wyznaczenie NWD

4.1 Implementacja mechanizmu generowania kluczy

Ważnym aspektem generowania kluczy była metoda generująca sekwencję liczb super-rosnących. Przy tworzeniu tej struktury zdecydowano się na listę, która w języku Python odznacza się szybkim dostępem do każdego jej elementu. Co najważniejsze, w tej sytuacji każdy z kolejno dodawanych elementów musiał być większy niż suma wszystkich poprzednich. Kod metody przedstawiono na listingu 1.

```
1 def generate_random_superincreasing_sequence(n):
2     sequence = [ ]
3     sequence.append(random.randint(1,10))
4     i = len(sequence)
5     for i in range(n):
6         additionalElement = sum(sequence)+random.randint(1,10)
7         sequence.append(additionalElement)
8     return sequence
```

Listing 1: Metoda generująca wektor liczb super-rosnących

Zmienną q wygenerowano z użyciem funkcji z biblioteki random. Wynik musiał jednak spełniać jeden warunek - stanowić wartość większą niż suma elementów wektora liczb super-rosnących. Wartość zmiennej r , była generowana losowo w przedziale $< 1,10 >$, przy czym konieczne było spełnienie warunku $NWD(r,q) = 1$. Wartości NWD obliczano za pomocą funkcji gcd z biblioteki math. Opisywane metody przedstawione są na listingu 2.

```
1 def generate_q(superIncreasing_sequence):
2     q = sum(superIncreasing_sequence) + random.randint(1,10)
3     return q
4
5 def generate_r(qValue):
6     r = random.randint(1,10)
7     while(math.gcd(r, qValue) != 1):
8         r = random.randint(1,10)
9     return r
```

Listing 2: Metody generujące q i r

Klucz prywatny generowany jest z użyciem wektora liczb super-rosnących, oraz wartości zmiennych q i r . Funkcjonalność tą, realizuje metoda z listingu 3.

```
1 def generate_private_key(superIncreasing_sequence, qValue, rValue):
2     privateKey = []
3     for i in superIncreasing_sequence:
4         privateKey.append(i)
5     privateKey.append(qValue)
6     privateKey.append(rValue)
7     return privateKey
```

Listing 3: Metoda generująca klucz prywatny

Klucz publiczny jest tworzony za pomocą operacji (8), gdzie każdy element wektora liczb super-rosnących jest poddawany odpowiednim przekształceniom matematycznym. Realizującą ten proces metoda przedstawiona jest na listingu 4.

```

1 def generate_private_key(superIncreasing_sequence, qValue, rValue):
2     privateKey = []
3     for i in superIncreasing_sequence:
4         privateKey.append(i)
5     privateKey.append(qValue)
6     privateKey.append(rValue)
7     return privateKey

```

Listing 4: Metoda generująca klucz publiczny

4.2 Implementacja mechanizmów szyfrowania i deszyfrowania

Mechanizm szyfrowania oparty jest na zamianie oryginalnej wiadomości zapisanej w tekście jawnym, na liczbę zapisaną w systemie dziesiętnym. Warto wspomnieć, że wiadomości wejściowe są ciągiem zer i jedynek - algorytm, natrafiając na wartość 1 w ciągu znaków, odczytuje indeks pod którym się ta wartość znajduje i używa go do odczytania prawidłowego elementu klucza publicznego. Element ten jest sumowany wraz z innymi uzyskanymi w ten sposób, tworząc zaszyfrowaną wiadomość. Metodę realizującą tą funkcjonalność przedstawia listing 5.

```

1 def knapsack_encryption(textToEncrypt, publicKey):
2     encryptedMessage = 0
3     for i in range(len(textToEncrypt)):
4         if textToEncrypt[i]=='1':
5             encryptedMessage += publicKey[i]
6     return encryptedMessage

```

Listing 5: Metoda szyfrująca

Deszyfrowanie odbywa się z udziałem klucza prywatnego. Głównym problemem rozwiązywanym podczas deszyfrowania jest znalezienie sumy podzbiorów dla zmiennej, w tym przypadku - c' , z użyciem liczb wchodzących w skład wektora super-rosnącego W . Metoda z listingu 6. realizuje to zadanie najpierw inicjalizując listę $wElements$ wartościami elementów klucza prywatnego, z wyjątkiem dwóch ostatnich, odpowiadających zmiennym q i r , a następnie szukając każdorazowo maksimum tej listy, a po odjęciu maksimum usuwając wspomniany element z listy $wElements$. Indeksy, pod którymi znajdowały się wartości maksymalne są dodawane do listy X , a następnie, z użyciem operacji potęgowania i sumy tworzona jest na ich podstawie zdeszyfrowana wiadomość, zapisana w formacie dziesiętnym.

```

1 def knapsack_encryption(textToEncrypt, publicKey):
2     encryptedMessage = 0
3     for i in range(len(textToEncrypt)):
4         if textToEncrypt[i]=='1':
5             encryptedMessage += publicKey[i]
6     return encryptedMessage

```

Listing 6: Metoda deszyfrująca

4.3 Testy poprawności działania systemu

Aby uzyskać informacje, czy system działa w sposób prawidłowy, konieczne było stworzenie jeszcze jednej metody. Metoda realizująca funkcjonalność związaną z deszyfrowaniem zwraca wynik w systemie dziesiętnym, stąd z zamiarem umożliwienia porównania wyniku z danymi wejściowymi w postaci binarnej, opracowano metodę, będącą konwerterem binarnym, widoczną na listingu 7.

```
1 def decimal_to_binary(decryptedMessage):
2     binaryMessage = ""
3     if decryptedMessage == 0:
4         return "0"
5     while decryptedMessage > 0:
6         binaryMessage = str(decryptedMessage % 2) + binaryMessage
7         decryptedMessage //= 2
8     return binaryMessage
```

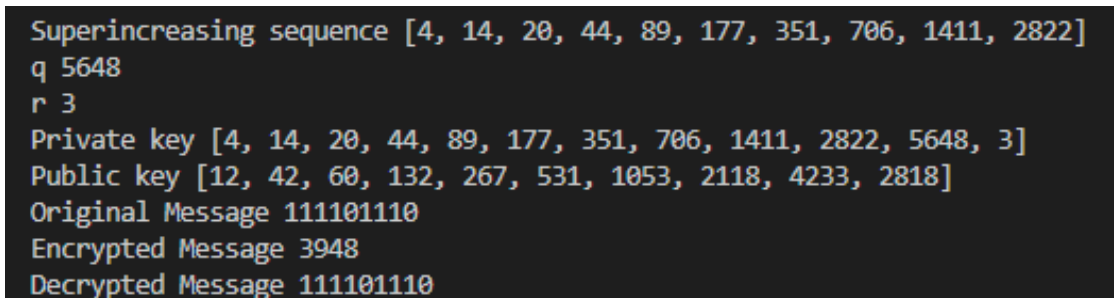
Listing 7: Metoda konwertująca na system binarny

Środowisko testowe zaprojektowano tworząc wywołania wszystkich dostępnych metod, a ich wyniki przekazując sukcesywnie do kolejnych następujących po sobie w ramach procedury działania algorytmu. Postanowiono zaimplementować wyświetlanie wartości istotnych zmiennych, tak aby lepiej przetestować działanie systemu.

```
1 superIncreasing_sequence = generate_random_superincreasing_sequence(9)
2 print("Superincreasing sequence", superIncreasing_sequence)
3 q = generate_q(superIncreasing_sequence)
4 print("q", q)
5 r = generate_r(q)
6 print("r", r)
7 privateKey = generate_private_key(superIncreasing_sequence, q, r)
8 print("Private key", privateKey)
9 publicKey = generate_public_key(superIncreasing_sequence, q, r)
10 print("Public key", publicKey)
11 originalMessage = '111101110'
12 print("Original Message", originalMessage)
13 encryptedMessage = knapsack_encryption(originalMessage, publicKey)
14 print("Encrypted Message", encryptedMessage)
15 decryptedMessage = knapsack_decryption(encryptedMessage, privateKey, q, r)
16 binaryMessage = decimal_to_binary(decryptedMessage)
17 print("Decrypted Message", binaryMessage)
```

Listing 8: Środowisko testowe

Działanie systemu zdecydowano się przetestować z użyciem wiadomości o treści '111101110'. System zwrócił wartości widoczne na rysunku 1.



```
Superincreasing sequence [4, 14, 20, 44, 89, 177, 351, 706, 1411, 2822]
q 5648
r 3
Private key [4, 14, 20, 44, 89, 177, 351, 706, 1411, 2822, 5648, 3]
Public key [12, 42, 60, 132, 267, 531, 1053, 2118, 4233, 2818]
Original Message 111101110
Encrypted Message 3948
Decrypted Message 111101110
```

Rysunek 1: Wyniki działania systemu

5 Podsumowanie

Analizując rysunek 1. można stwierdzić, iż wiadomość oryginalna została zaszyfrowana i odszyfrowana w prawidłowy sposób.

Literatura

- [1] Knapsack Cryptosystems: The Past and the Future, Ming Kin Lai, Department of Information and Computer Science, University of California Irvine, CA 92717-3425, USA, Email: mingl@ics.uci.edu, March 2001
- [2] Douglas R. Stinson. Cryptography Theory and Practice. CRC Press, Boca Raton, 1995.
- [3] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, MA, 1974.
- [4] Ralph C. Merkle, Martin E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks. IEEE Transactions on Information Theory, vol. IT-24, 1978, pp. 525-530.
- [5] Knapsack Encryption Algorithm in Cryptography, <https://www.tutorialspoint.com/knapsack-encryption-algorithm-in-cryptography>, dostęp 14.03.2024