Tomczyk Piotr 236676
Zalataya Anastasiya 236708

Podstawy kryptografii
Środa, 10:15

Szyfr asymetryczny RSA

1. Algorytm RSA

Asymetrycznych algorytmów kryptograficznych z kluczem publicznym, zaprojektowany w 1977 przez Rona Rivesta, Adiego Shamira oraz Leonarda Adlemana. Bezpieczeństwo szyfrowania opiera się na trudności faktoryzacji dużych liczb złożonych. Jego nazwa pochodzi od pierwszych liter nazwisk jego twórców. Proces szyfrowania polega na generacji kluczy publicznego i prywatnego, następnie udostępnia się klucz publiczny, co umożliwia każdej osobie na zaszyfrowanie wiadomości. Jedynie klucz prywatny pozwala na odszyfrowanie wiadomości.

Szyfrowanie wiadomości polega na podziale wiadomości na części długości klucza (np. 128 bitowe) następnie wykonaniu następującej operacji:

$$c \equiv m^e (mod\ n)$$

Deszyfrowanie

$$m \equiv c^d (mod\ n)$$

Gdzie:

- m - wiadomość
- d - liczba będąca sekretną częścią klucza prywatnego
- n - liczba będąca częścią kluczy prywatnego i publicznego
- c - zaszyfrowana wiadomość
- e - liczba będąca składnikiem klucza publicznego

Liczby ***e*** i ***d*** są wybrane jako liczby odwrotne wzajemnie w modulo n. Dlatego umożliwia to działanie algorytmu. A ***n*** z dwóch wielobajtowych liczb pierwszych zapewnia bezpieczeństwo, z powodu trudności faktoryzacji liczb pierwszych.

2. Implementacja algorytmu

Język: Java wersja 18
Aby zaimplementować algorytm RSA potrzebujemy mieć możliwość operacji na liczbach dowolnej długości (do 1024 bitów).

Aby to było możliwe została napisana klasa `UFatInt` (Unsigned Fast Integer). Przechowuje ona liczbę jako listę bajtów, umożliwia to operacje przesunięć bitowych jak i redukcję pamięci potrzebuje do przechowywania liczby w stosunku do przechowywania

jej pozycyjnie np. na bazie String. W celu dalszego usprawnienia algorytmu zostały zastosowane następujące algorytmy:

- Mnożenie Karatsuby, złożoność $O(n) = 2^{log(3)}$
- Dzielnie na podstawie przesunięć bitowych
- Fast Pow, usprawnia podnoszenie liczb do potęgi
- Mod Pow, podczas podnoszenia do potęgi jednocześnie wykonuje operacje modulo, zmniejszając zakres liczby na której wykonuje się operacje
- Do testowania pierwszości liczb zastosowano probabilistyczny test Millera-Rabina w 5 iteracjach.
- Stosuje się również inne usprawniające metody, np. metoda odejmowania jedności od liczby nieparzystej czy podnoszenie do 10 potęgi na przesunięciach bitowych.

Generacja kluczy:

Na początku generuje się liczba nieparzysta o zadanej długości ***p***, następnie 5 razy sprawdza się za pomocą testu Millera-Rabina czy jest ona liczbą pierwszą, jeśli nie to powtarzamy krok. Taki sam schemat jest wykorzystywany do wygenerowania drugiej liczby pierwszej ***q***. Następnie sprawdza się czy nie są one takie same, jeśli są, generuje się kolejna liczba. Po wygenerowaniu wyliczana jest wartość ***n*** przez wymnożenie przez siebie ***p*** i ***q***. Korzystając z własności funkcji Euler'a dla liczb pierwszych wyznacza się wartość funkcji $\phi(n) = (p - 1)(q - 1)$. Dalej generuje się losowe ***e*** o takiej samej długości jak $\phi(n)$, będące względnie pierwsze z liczbą $\phi(n)$. Na końcu wyznaczamy ***d***, element odwrotny do liczby ***e*** w modulo $\phi(n)$. Ostatecznie klucz prywatny definiowany jest jako para liczb (***d***, ***n***), a klucz publiczny (***e***, ***n***).

Aby wytworzyć klucz o odpowiedniej długości należy wygenerować obie liczby pierwsze o długości o połowę mniejszej niż docelowa.

Szyfrowanie:

Wpierw odczytywana jest zawartość z pliku (dla plików binarnych jest kodowana w kodowaniu `base64` w celu lepszego przedstawienia w oknie tekstowym) lub bezpośrednio wpisywana przez użytkownika. Następnie powstałą wiadomość dzieli się na fragmenty równe długości klucza. Powstałe fragmenty są konwertowane na liczby w reprezentacji klasy `UFatInt`, z odpowiedniego pola odczytywana jest wartość klucza publicznego i wykonywane są operacje matematyczne jak opisane w sekcji 1. Wynikiem jest tablica liczb klasy `UFatInt`, która jest konwertowana na jednowymiarową tablice bajtów. W celu prezentacji odbywa się konwersja na kodowanie `base64` i zaszyfrowaną wiadomość wyświetlamy w polu `CIPHER TEXT`.

Możliwy jest zapis/odczyt szyfrogramu, przed zapisem na dysk odbywa dekodowanie z systemu `base64`, a podczas odczytu następuje konwersja do systemu `base64`.

Odszyfrowanie:

Z pola `CIPHER TEXT` odczytywany jest tekst i dekodowany z kodowania `base64`, następnie otrzymaną tablice bajtów dzieli się na tablice o długości klucza. Otrzymane bajty są zamieniane na instancje klasy `UFatInt` w celu wykonania operacji matematycznych opisanych w sekcji 1. Wynikiem jest tablica liczb klasy `UFatInt`, która jest konwertowana na jednowymiarową tablice bajtów, determinowane jest metodą `try` czy oryginalna

wiadomość była binarna czy tekstowa, następnie odpowiednio zapisywana do pola `PLAIN TEXT` jako tekst lub w kodowaniu `base64`.

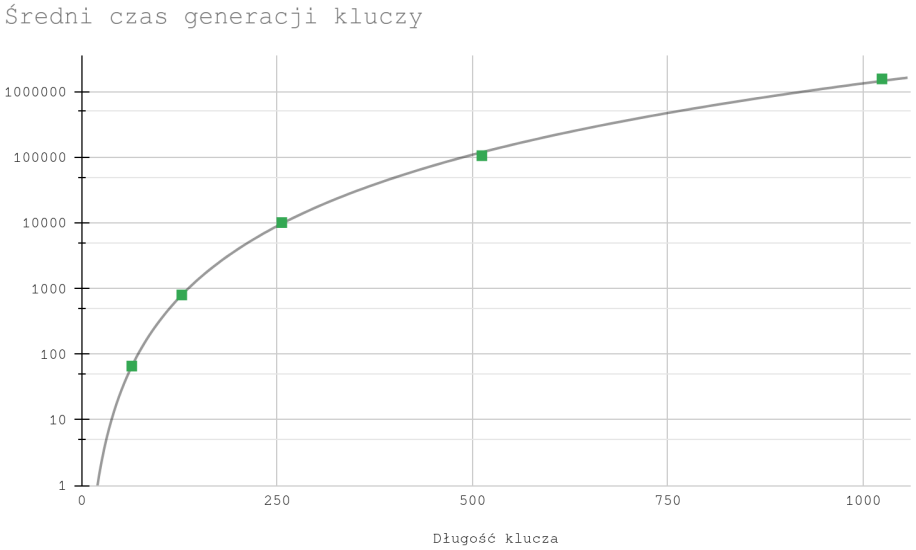
Zastosowane struktury i funkcjonalności biblioteczne:

- ByteBuffer - w celu ułatwienia konwersji typów liczbowych na bajty
- List, ArrayList - w celu łatwego przechowywania danych
- Base64 - klasa odpowiedzialna za konwersję do kodowania `base64`

3. Analiza

Tabela 1. Analiza czasu generacji kluczy w zależności od długości

Długość klucza	Minimalny czas	Średni czas	Maksymalny czas
64	29 ms	66 ms	145 ms
128	262 ms	789 ms	2082 ms
256	3137 ms	10133 ms	32674 ms
512	16.315 s	105.912 s	266.118 s
1024	400.821 s	26.05 min	57.77 min



Literatura:

- Algorytm RSA: https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf
- Kryptografia dla praktyków. Protokoły, algorytmy i programy źródłowe w języku C. Bruce Schneier