PODSTAWY KRYPTOGRAFII

Zadanie 2, zestaw V

Wykonali:

Michał Ferdzyn 242383

Artur Grzybek 242399

Cel zadania:

Napisać program szyfrujący/deszyfrujący dane wprowadzone przez użytkownika lub z pliku wykorzystując algorytm ElGamala.

Szyfrowanie/Deszyfrowanie ElGamala

Szyfrowanie i deszyfrowanie są nieodłącznymi elementami dziedziny bezpieczeństwa komunikacji i przechowywania danych. Jednym z popularnych algorytmów szyfrowania asymetrycznego jest szyfrowanie ElGamala. Algorytm ten, nazwany na cześć swojego twórcy Tahera ElGamala, jest oparty na matematycznym problemie dyskretnego logarytmu i jest szeroko stosowany do bezpiecznej wymiany kluczy oraz poufnego przesyłania danych przez niezabezpieczone kanały komunikacyjne. Szyfrowanie asymetryczne różni się od symetrycznego, ponieważ wykorzystuje dwa różne klucze: publiczny i prywatny. Klucz publiczny jest dostępny publicznie i może być używany do szyfrowania wiadomości, podczas gdy klucz prywatny jest utrzymywany w tajemnicy i służy do deszyfrowania wiadomości. Algorytm ElGamala jest przykładem takiego systemu szyfrowania asymetrycznego.

Opis algorytmu

1. Ustawienie wartości p,g,h, które są kluczami publicznymi:
    - Metoda *setP(BigInteger p)* ustawia wartość zmiennej *p*.
    - Metoda *setH(BigInteger h*) ustawia wartość zmiennej *h*.
    - Metoda *setG(BigInteger g)* ustawia wartość zmiennej *g*.
2. Generowanie kluczy:
    - Metoda *generateKeys()* generuje klucze publiczne i prywatne.
    - Generowana jest liczba pierwsza *p* o długości bitowej 512.
    - Generowane są losowe liczby *g* i *k* o długości bitowej 510.
    - Obliczane są wartości *h* i *Nm1*.
    - Klucze publiczne i prywatne są zapisywane jako wartości szesnastkowe w zmiennych *pKey*, *gKey, hKey i privateKey*.
3. Generowanie prawdopodobnej liczby pierwszej:
    - Prywatna metoda *generateProbablePrime(int bitLength)* generuje losową liczbę pierwszą o określonej długości bitowej.
    - Wykorzystuje algorytm testu pierwszości Millera-Rabina zaimplementowany w klasie MillerRabin.

4. Generowanie losowej liczby BigInteger:
    - Prywatna metoda *generateRandomBigInteger(int bitLength)* generuje losową liczbę BigInteger o określonej długości bitowej.
5. Szyfrowanie wiadomości:
    - Metoda *encryptMessage(byte[] toEncrypt)* szyfruje wiadomość.
    - Generowana jest losowa liczba *b* o długości bitowej 500.
    - Obliczane są wartości *c1* i *c2* na podstawie kluczy publicznych g i h, oraz wiadomości *toEncrypt*.
    - Zwracane są zaszyfrowane dane w postaci tablicy dwóch ciągów znaków *c1* i *c2*.
6. Weryfikacja kluczy:
    - Metoda *verifyKeys()* sprawdza poprawność kluczy publicznych.
    - Sprawdzane jest, czy *h* jest równe *g* podniesionemu do potęgi *k* modulo *p*.
7. Deszyfrowanie wiadomości:
    - Metoda *decryptMessage()* deszyfruje zaszyfrowaną wiadomość.
    - Obliczane są tymczasowe wartości *temp* i *temp2* na podstawie zaszyfrowanych danych *c1* i *c2*.
    - Zwracana jest odszyfrowana wiadomość w postaci tablicy bajtów.
8. Pobieranie kluczy publicznych:
    - Metoda *getpKey()* zwraca klucz publiczny *p* jako ciąg znaków w formacie szesnastkowym.
    - Metoda *getgKey()* zwraca klucz publiczny *g* jako ciąg znaków w formacie szesnastkowym.
    - Metoda *gethKey()* zwraca klucz publiczny *h* jako ciąg znaków w formacie szesnastkowym.
9. Ustawianie i pobieranie klucza prywatnego:
    - Metoda *setPrivateKey(BigInteger privateKey)* ustawia klucz prywatny *k*.
    - Metoda *getPrivateKey()* zwraca klucz prywatny *k* jako ciąg znaków w formacie szesnastkowym.

Implementacja

Program został przygotowany w języku Java, natomiast interfejs graficzny został przygotowany przy pomocy JavaFX. Kod został podzielony na pięć plików, z których dwa służą do implementacji GUI, pozwalają nam na komunikacje z użytkownikiem. Klasa ElGamal kolejno przedstawia funkcjonalności opisane przez nas powyżej, zawiera właściwą implementacje szyfrowania oraz deszyfrowania algorytmu. Wyodrębniona została klasa MillerRabin, która zawiera implementacje testu pierwszości. W celach przetestowania owej klasy dodaliśmy dodatkowo klasę testową MillerRabinTest, która sprawdza poprawność funkcji *isProbablePrime(BigInteger n).* Użytkownik poprzez stworzoną przez nas aplikacje okienkową może samodzielnie wpisać klucze, z których chce skorzystać lub losowo je wygenerować. Umożliwione zostało samodzielne wprowadzanie tekstu jawnego, który ma zostać zaszyfrowany lub też wczytanie owego tekstu z wybranego pliku (analogicznie można wpisać szyfr, który chcemy odszyfrować bądź też wczytać go z pliku). Zarówno szyfr jak i odszyfrowany tekst wyświetlany jest w polach tekstowych, ale jest też możliwość zapisania owej zawartości do pliku.

Zapewniona została również funkcjonalność szyfrowania oraz odszyfrowywania plików binarnych, takich jak .pdf

Wnioski

- Ważnym aspektem algorytmu ElGamala jest klucz prywatny, który jest trzymany w tajemnicy i służy do deszyfrowania wiadomości.
- Algorytm ElGamal jest odporanym na ataki kryptograficznym, jeśli używane parametry są odpowiednio dobrane i klucze są odpowiednio generowane.
- Algorytm ElGamal opiera się na problemie dyskretnego logarytmu, który jest trudny do rozwiązania, co zapewnia bezpieczeństwo procesu szyfrowania i deszyfrowania wiadomości.

Bibliografia

1. http://zon8.physd.amu.edu.pl/~tanas/krypt07.pdf
2. https://ftims.edu.p.lodz.pl/course/view.php?id=1801 ,,WYKŁAD KRYPTOGRAFIA ASYMETRYCZNA CZĘŚĆ II”
3. https://pl.wikipedia.org/wiki/ElGamal