Wojciech Kolendo 210038 23/10/2016

Repozytorium: https://github.com/kolendo/diffie_hellman_chat

Bezpieczeństwo systemów i usług informatycznych 2

Raport z laboratorium 1 & 2 – Komunikator z szyfrowaniem

Prowadzący: Mgr inż. Przemysław Świercz

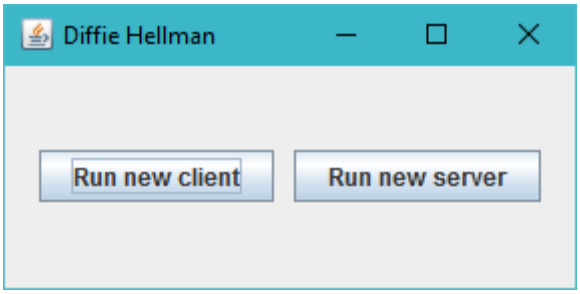
Termin zajęć: środa 15:15

1. Cel zadania.

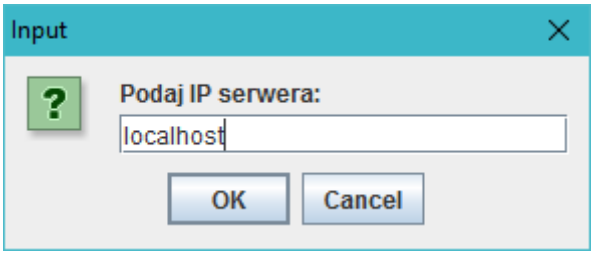
Zadanie zostało rozłożone na dwa laboratoria, a jego celem było przygotowanie komunikatora (czatu) klient – serwer ze wsparciem bezpiecznej wymiany danych z protokołem Diffie – Hellman.

2. Zrealizowane zadanie.

- Zaimplementowany został komunikator z interfejsem graficznym oparty na socketach w języku Java. Serwer ustawiany jest na porcie 8080.



- Klient przy uruchomieniu pyta użytkownika o adres IP serwera (podanie „localhost” ustawia w sockecie domyślne 127.0.0.1).



- Serwer obsługuje wielu klientów w tym samym czasie w postaci osobnych wątków trzymanych w kolekcji. Ustawiona jest maksymalna ilość 10 klientów naraz.

```
/**
 * Stała maksymalna ilość obsługiwanych klientów przez serwer.
 */
private static final int MAX_CLIENTS_COUNT = 10;
private static final ClientThread[] THREADS = new ClientThread[MAX_CLIENTS_COUNT];
private static ServerSocket sServerSocket = null;
private static Socket sClientSocket = null;
```

- Wszystkie dane między serwerem a klientem są wysyłane w strukture Json (z użyciem biblioteki json.org).

- Przy inicjalizacji klienta na serwerze, serwer wysyła do klienta request { ''init'' : ''start'' } co rozpoczyna rozpoczęcie protokołu Diffie-Hellman. Następuje wymiana kluczy, obliczenie wartości sekretu w kliencie i na serwerze oraz po obliczeniu sekretu ustawienie flagi gotowości.

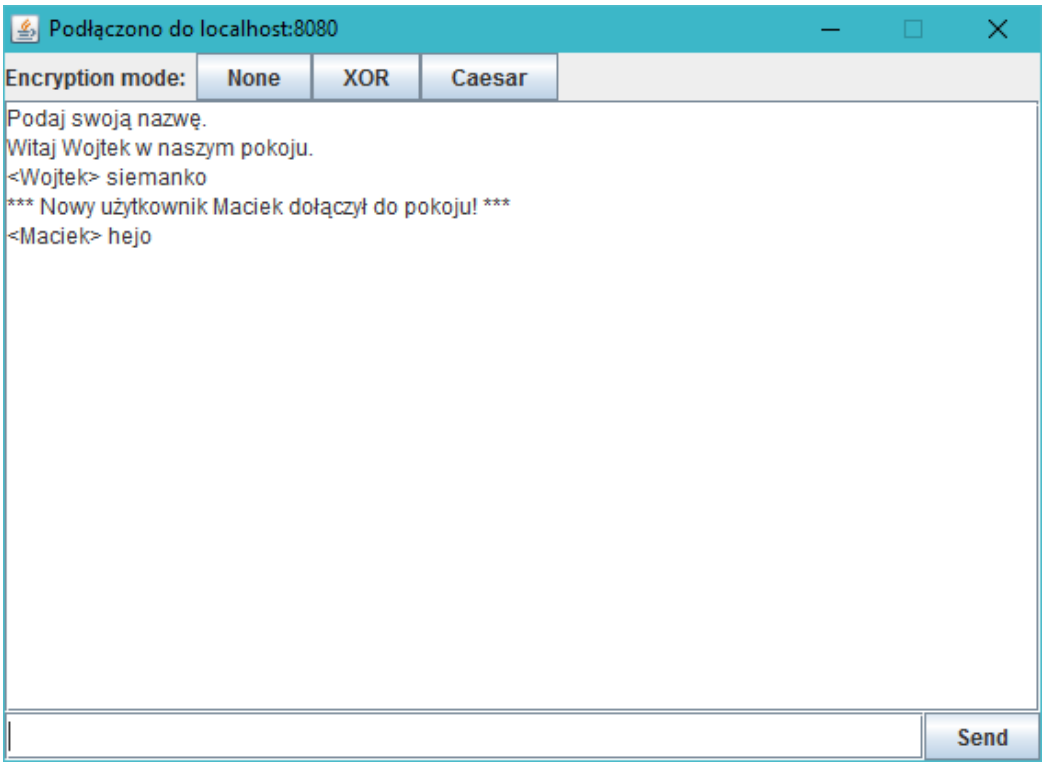
```
received: {"msg":"Podaj swoją nazwę."}
received: {"msg":"Witaj Wojtek w naszym pokoju.","init":"start"}
received: {"msg":"Witaj Wojtek w naszym pokoju.","init":"start"}
start auth
received: {"p":32,"B":15.0,"g":47}
s: 1
received: {"p":32,"B":15.0,"g":47}
s: 1
received: {"msg":"c2llbWFua28=","from":"Wojtek"}
```

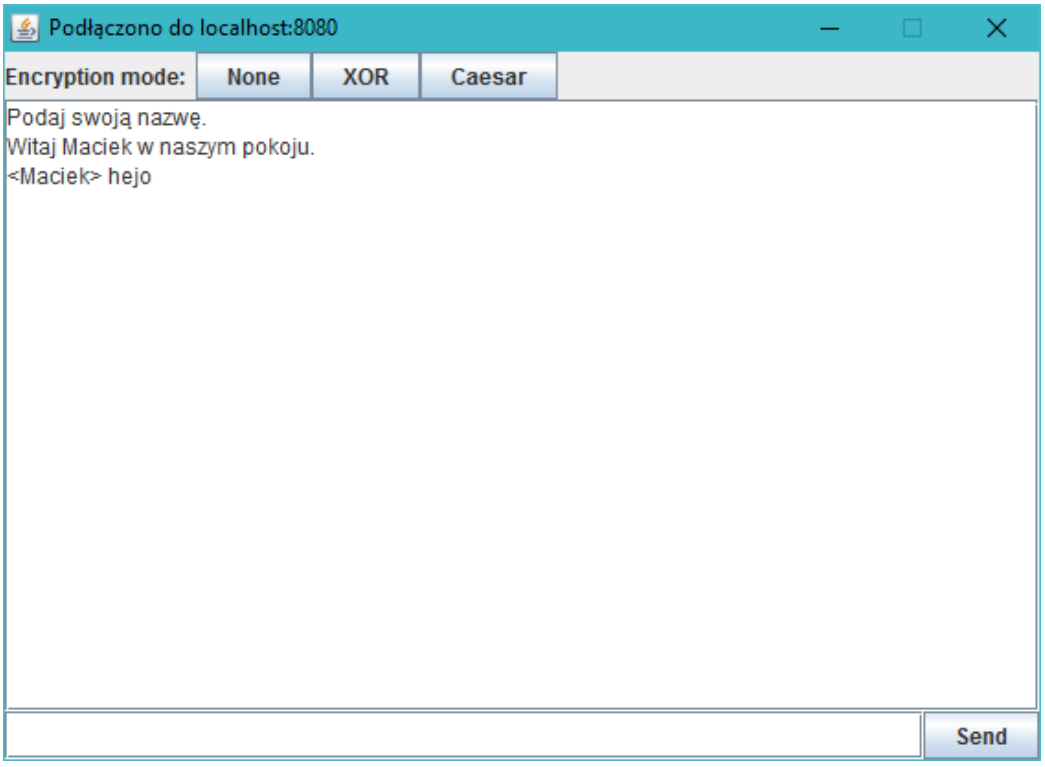
- Parametry protokołu Diffie-Hellman są generowane losowo i są inne dla każdego klienta.

```
/**
 * Konstruktor wątku klienta. Losuje wartościom P, G oraz b losową, unikalną dla klienta liczbę.
 *
 * @param clientSocket socket klienta
 * @param threads kolekcja pozostałych wątków (innych klientów) na serwerze
 */
public ClientThread(Socket clientSocket, ClientThread[] threads) {
    mClientSocket = clientSocket;
    mClientThreads = threads;
    mMaxClientsCount = threads.length;
    Random generator = new Random();
    VALUE_P = generator.nextInt(100) + 1;
    VALUE_G = generator.nextInt(100) + 1;
    value_b = generator.nextInt(10) + 1;
    value_B = (Math.pow(VALUE_G, value_b) % VALUE_P);
}
```

- Każda z wiadomości jest szyfrowana do formatu Base64.
- Wpisanie /quit przez użytkownika kończy wątek klienta na serwerze.

- Okno klienta posiada 3 przyciski do wyboru metody szyfrowania wiadomości:
  - None: jedynie Base64
  - Xor: Base64 + xorowanie każdego znaku ASCII wiadomości z najmłodszym bajtem sekretu
  - Caesar: Base64 + szyfr cezara z wartością sekretu jako klucz





3. Szczegóły implementacji.

- Udokumentowany kod źródłowy dostępny jest na repozytorium https://github.com/kolendo/diffie_hellman_chat

- Kodowanie szyfrem cezara:

```
private String encodeCaesar(String enc, int offset) {
    offset = offset % 26 + 26;
    StringBuilder encoded = new StringBuilder();
    for (char i : enc.toCharArray()) {
        if (Character.isLetter(i)) {
            if (Character.isUpperCase(i)) {
                encoded.append((char) ('A' + (i - 'A' + offset) % 26 ));
            } else {
                encoded.append((char) ('a' + (i - 'a' + offset) % 26 ));
            }
        } else {
            encoded.append(i);
        }
    }
    return encoded.toString();
}
```

- Dekodowanie szyfru cezara:

```
private String decodeCaesar(String text, int offset) {
    return encodeCaesar(text, 26-offset);
}
```

- Kodowanie/dekodowanie szyfru Xor:

```
private String codeXor(String string, int secret){
    byte b = (byte) (secret & 0xFF);
    StringBuilder encrypted = new StringBuilder();
    for (byte c : string.getBytes(StandardCharsets.UTF_8)){
        encrypted.append((char)(c ^ b));
    }
    return encrypted.toString();
}
```

- Wysyłanie wiadomości na serwer:

```
/**
 * Wysłanie tekstu na serwer
 * @param text wysyłany tekst
 */
private void sendMessage(String text) {
    try {
        if (INIT_MSG || text.startsWith("/quit")) {
            mJSONObject = new JSONObject();
            mJSONObject.put("msg", text);
            mOutputStream.write((mJSONObject.toString() + CRLF).getBytes());
            mOutputStream.flush();
            if (INIT_MSG) {
                INIT_MSG = false;
            }
        } else if (DIFFIE_READY) {
            switch (mEncryptionType) {
                case XOR: {
                    text = codeXor(Base64.getEncoder().encodeToString(text.getBytes("utf-8")), value_s);
                    mJSONObject = new JSONObject();
                    mJSONObject.put("msg", text);
                    mOutputStream.write((mJSONObject.toString() + CRLF).getBytes());
                    mOutputStream.flush();
                    break;
                }
                case CAESAR: {
                    text = encodeCaesar(Base64.getEncoder().encodeToString(text.getBytes("utf-8")), value_s);
                    mJSONObject = new JSONObject();
                    mJSONObject.put("msg", text);
                    mOutputStream.write((mJSONObject.toString() + CRLF).getBytes());
                    mOutputStream.flush();
                    break;
                }
                default: {
                    mJSONObject = new JSONObject();
                    mJSONObject.put("msg", Base64.getEncoder().encodeToString(text.getBytes("utf-8")));
                    mOutputStream.write((mJSONObject.toString() + CRLF).getBytes());
                    mOutputStream.flush();
                    break;
                }
            }
        }
    } catch (IOException e) {
        System.out.println(e);
        notifyObservers(e);
    }
}
```

4. Wnioski, przemyślenia.

- Szyfrowanie przez xorowanie kolejnych znaków ASCII wiadomości wydaje się najlepszym sposobem szyfrowania z zaimplementowanych rozwiązań. Szansa na odgadnięcie wartości xorującego bajtu z przedziału 0 – 255 jest o wiele mniejsza, niż w szyfrze cezara, gdzie wystarczy prześledzić wszystkie kombinacje jakie daje alfabet.