Bezpieczeństwo Systemów i Usług Informatycznych

Laboratorium 1 - Komunikator z szyfrowaniem

1. Cel zadania.

Celem zadania było wykonanie w wybranej technologii komunikatora internetowego wspierającego komunikację wielu osób oraz implementującego szyfrowanie XOR i Cezara. Do uzyskania klucza wymagana była implementacja algorytmu Diffiego-Hellmana. Komunikacja klientów z serwerem odbywać się miała przy użyciu obiektów JSON wedle zadanej struktury.

Całość projektu składa się z części serwerowej oraz klienckiej.

2. Opis wykonania.

Zadanie wykonałem używając języka C# oraz platformy .NET. Aplikacja serwerowa utworzona została w formie konsolowej, natomiast do utworzenia GUI aplikacji klienckiej posłużyła mi technologia WPF.

2.1 Gniazda.

Całość działania komunikatora opiera się na wykorzystaniu gniazd (socketów). W języku C# do implementacji tego typu rozwiązań służy przestrzeń nazw System.Net, a w szczególności System.Net.Sockets.

Utworzenie i uruchomienie gniazda serwerowego wymaga jedynie kilku linijek kodu.

```
localEndPoint = new IPEndPoint(ipAddress, serverPort);

socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

socket.Bind(localEndPoint);
socket.Listen(10);

client = socket.Accept();
```

Pierwszym krokiem jest utworzenie obiektu typu IPEndPoint, który reprezentuje punkt końcowy. Do jego utworzenia potrzebny jest adres IP, oraz port nasłuchowy serwera. Kolejnym krokiem jest utworzenie gniazda. Możemy ustawić jaka rodzina adresów nas interesuje, typ przesyłu danych oraz protokół. Następnie łączyny gniazdo z punktem końcowym i uruchamiamy nasłuch.

Wątek czeka bezczynnie do momentu przyjścia połączenia od klienta.

2.2 Łączenie z serwerem.

Klient aby połączyć się z serwerem musi wykonać kilka kroków.

```
remoteEP = new IPEndPoint(ipAddress, port);

socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
socket.Connect(remoteEP);
```

Analogicznie jak w przypadku serwera należy utworzyć obiekt typu IPEndpoint (podając przy tym IP i port serwera), oraz utworzyć gniazdo. Tym razem korzystamy z funkcji Connect() do połączenia się z serwerem.

2.2 Komunikacja.

Przesyłanie informacji również jest bardzo wygodne. Wykorzystujemy do tego dwa obiekty.

```
public StreamWriter streamWriter { get; set; }
public StreamReader streamReader { get; set; }
```

Obiekty te umożliwiają strumieniowe korzystanie z gniazda. Wystarczy podać do konstruktora obiekt Socket, a następnie korzystać z funkcji jakie one oferują: odpowiednio WriteLine() oraz ReadLine(). Funkcje te pozwalają na przesyłanie obiektów typu string, co jest przydatne z uwagi na to, że komunikacja klienta z serwerem odbywa się przy użyciu obiektów JSON.

Aby aplikacja mogła wspierać wielu użytkowników jednocześnie wymagane było użycie wielu wątków (przestrzeń nazw System.Threading). Serwer posiada główny wątek nasłuchujący. Gdy podłącza się jakiś klient, tworzony jest osobny wątek sesji, który go obsługuje (klasa Session). Główny wątek posiada kolekcję sesji

```
public List<Session> clientSessions { get; set; }
```

oraz gniazd klienckich

```
public List<Socket> clientSockets { get; set; }
```

co umożliwia wysyłanie wiadomości do wszystkich klientów.

Kiedy indywidualna sesja po stronie serwera otrzymuje wiadomość od klienta, wysyła ją do głównego wątku serwera, a on uruchamia proces rozsyłania we wszystkich sesjach. Każda z sesji otrzymuje rozkodowaną wersję komunikatu, co umożliwia komunikację między użytkownikami używającymi innych algorytmów szyfrowania.

3.3 Kodowanie.

Obie strony komunikacji posiadają enkodery i dekodery Base64 oraz szyfrów XOR i Cezar.

3.4 Diffie-Hellman.

Algorytm został zaimplementowany zgodnie z ustaleniami. Ważne było zagwarantowanie dowolnej kolejności przesyłu danych w oznaczonych miejscach. Niestety platforma .NET nie posiada funkcji generującej losowe, duże liczby pierwsze. Na potrzeby programu przyjąłem, że p = 23, g = 5. Nie są to wartości bezpieczne, ale pozwalają sprawdzić poprawność działania algorytmu.

3.5 GUI

Graficzny interfejs użytkownika wykonany został w oparciu o technologię WPF. Okno aplikacji zaprogramowane jest w dwóch plikach: MainWindow.xaml oraz MainWindow.xaml.cs. Pierwszy z nich zawiera zakodowane rozmieszczenie elementów interfejsu (język XAML), drugi implementację działania (język C#).

Podsumowanie

Powyższe zadanie okazało się być ciekawym i rozwijającym ćwiczeniem. C# oraz platforma .NET oferują bardzo wygodne rozwiązania jeżeli chodzi o budowanie interfejsów użytkownika oraz implementację połączeń z wykorzystaniem gniazd. Dużą wadą natomiast okazał się być brak funkcji do generowania dużych liczb pierwszych. Tego typu rozwiązania posiada przykładowo Java. Brak tego typu funkcjonalności wydaje mi się bardzo zastanawiający. Prawdopodobnie wynika z tego, że większość algorytmów związanych z szyfrowaniem (przykładowo Diffie-Hellman, RSA) ma swoje gotowe implementacje w bibliotekach składowych platformy .NET.