Bezp. syst. i usług inform. 2

Komunikator z szyfrowaniem

Mateusz Mańka 209895  
22 października 2016

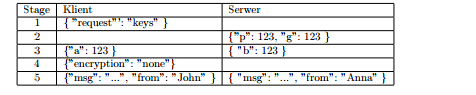
# Cel projektu

Cele m projektu jest przygotowanie komunikatora w architekturze klient-serwer wspierającego bezpieczną wymianę sekretu wg protokołu Diffiego-Hellmana oraz obsługujący zadany format komunikacji.

# Wymagania

## Protokół komunikacji

W celu zapewnienia kompatybilności z innymi realizacjami tego projektu, ustalony został protokół komunikacji, według którego powinna odbywać się wymiana wiadomości w aplikacji. Wiadomości przesyłane pomiędzy klientem a serwerem powinny być zgodne z formatem danych opartych o JSON przedstawiony:



Podczas wymiany wiadomości należy uwzględnić podane wymagania:

* W kroku 3. wiadomości od serwera i klienta mogą nastąpić w dowolnej kolejności.
* Krok 4. tabeli 1 jest opcjonalny. W przypadku braku wysłania wiadomości o metodzie szyfrowania należy przyjąć wartość domyślną none.

## Szyfrowanie

Wiadomości przesyłane między klientami a serwerem powinny być szyfrowane według metody wybranej przez użytkownika. Komunikator powinien wspirać następujące metody szyfrowania:

• none - brak szyfrowania (domyślne)

• xor - szyfrowanie OTP xor jednobajtowe (należy użyć najmłodszego bajtu sekretu)

• cezar - szyfr cezara

Komunikator powinien zapewniać możliwość zmiany metody szyfrowania w dowolnym momencie. Treść wiadomości powinna być zakodowana za pomocą base64 przed umieszczeniem jej w strukturze JSON:

b a se 6 4 ( e n c r y p t ( u s e r m e s s a g e ) )

# Projekt i implementacja

Aplikacja wykonana została w języku Java(wersja 1.8) z wykorzystaniem Swinga do stworzenia interfejsów a także technologii Maven w celu łatwiejszego zarządzania pluginami, pluginem Gson do tworzenia pluginu. Projekt został zaimplementowany w środowisku JnteliJ IDEA.

## Koncepcja Rozwiązania

Część serwerowa i kliencka aplikacji zostaną wykonane w ramach jednego projektu lecz różnych pakietów (Client , Server). Aby wybrać funkcje klienta należy odpalić plik Client.java , aby odpalić funkcje Serwera należy odpalić plik Server.java. Po skonfigurowaniu parametrów połączenia aplikacja rozpocznie działanie zależnie od wybranej funkcji według jednego ze schematów:

### Klient :

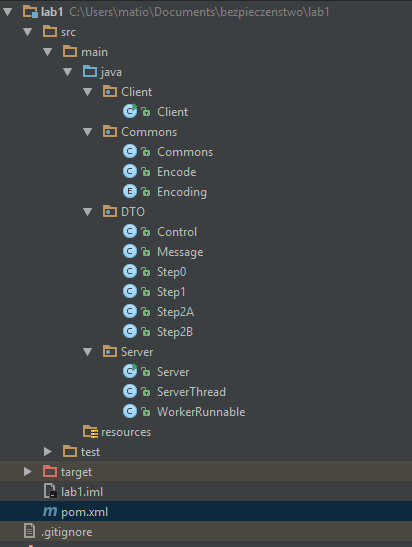
* Aplikacja tworzy połączenie z zadanym adresem na wskazanym porcie(W pliku Commons.java) .
* Po utworzeniu połączenia rozpoczyna się wymiana kluczy według protokołu opisanego w punkcie 2.1.
* Po ustaleniu kluczy szyfrowania aplikacja oczekuje na interakcję użytkownika.
* Użytkownik może wybrać

Serwer :

* Aplikacja tworzy nowy wątek oczekujący na połączenia.
* Po ustanowieniu połączenia z klientem serwer tworzy dla niego wątek nasłuchujący i odpowiada na zapytania ze strony klienta według ustalonego protokołu.
* Po otrzymaniu wiadomości z jednego z aktywnych wątków klienckich, serwer odszyfrowuje wiadomość według metody wybranej przez nadawcę.
* Serwer wyświetla wiadomość w konsoli a następnie zwraca ją, ponownie szyfrując ją.

## Struktura projektu

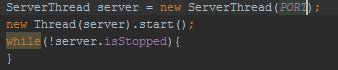
Projekt składa się z 4 pakietów 13 klas i pliku pom.xml do konfiguracji mavana i pluginów.

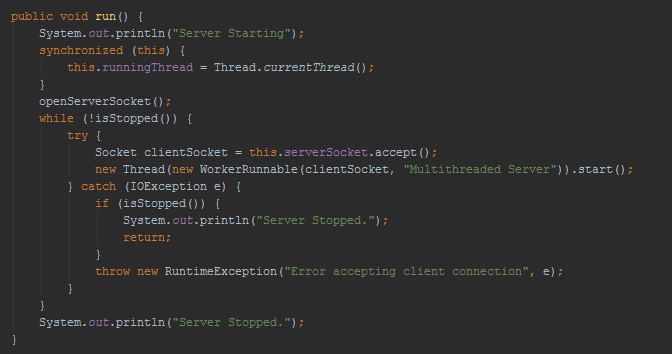
Pakiety :

1. Client zawiera Klase:
   1. Client która zawiera wszystkie metody do komunikacji z serwerem wraz z GUI w Swingu
2. Commons klasy wspólne :
   1. Commons wpolne wartości odnośnie adresu serwera a także wspólne metody dla soketów
   2. Encode wpsólne metody kodujące
   3. Encoding Enum metod kodowania
3. DTO Data transwear Objects wykorzystywane do budowania Jsonów opisanych w punkcie 2.1
4. Server zawiera klasy :
   1. Server klasa z metodą main do uruchomienia serwera
   2. ServerThread : Nasłuchiwanie na nowy socket
   3. WorkerRunnable : Oblugiwanie konkretnego klienta (Wątek)

## Nawiązywanie połączenia i wymiana kluczy :

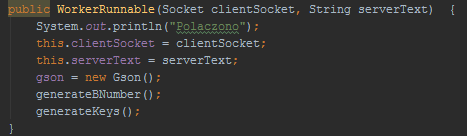
### Uruchomienie serwera:

Utworzenie wątku ServerThread , uruchomienie go (metoda start wywołująca metode run) i oczekiwanie na zamknięcie.

Metoda run w klasie ServerThread :   


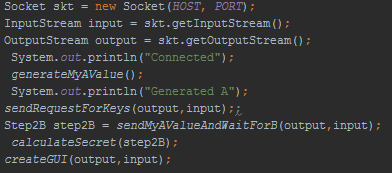
Metoda ta za pomocą metody klasy Socket accept() oczekuje na połączenie z klienta , w przypadku połączenia tworzy nowy wątek WorkerRunnable i uruchamia go , a następnie oczekuje na kolejne połączenie do momentu zatrzymania (pętla while)

### WorkerRunnable :

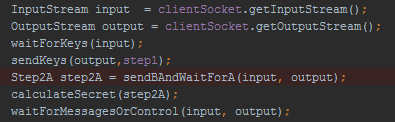


WorkerRunnable generuje swąją liczbę b dla tego klienta a także generuje klucze p i g.

### Klient :

Tworząc obiekt Socket tworzymy połączenie z serwerem.  
następnie generujemy wartość a (prywatną), wysyłamy prośbe o klucze i czekamy.   
w sendMyValueAndWaitForB obliczamy wartość A wysyłamy i oczekujemy na wartość B i obliczamy secret.

### Serwer :



Oczekujemy na prośbę na klucze i wysyłamy je, następnie oczekujemy na wartość A i wysyłamy swoją obliczoną wartość B i obliczamy Secret . I oczekujemy na wiadomość lub zmiane kodowania.

## Obsługa wiadomości :

### Serwer :

Oczekuje wiadomości następnie stara się ją sparsowac na Message jeśli się nie uda oznacza to że jest to wiadomość kontrolna.

### Klient:

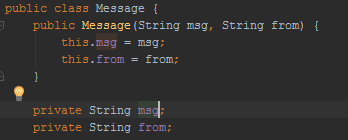
Za pomocą gui może zmienć kodowanie za pomocą ComboBox Encoding   
Lub napisać wiadomość w miejscu Status, odpisać swoje imię w miejscu FROM i wysłać wiadomość za pomocą klawisza Send, wiadomośc zwrotna zostanie wyświetlona w miejscu status.

## Tworzenie Jsona

Json zostaje tworzony za pomocą biblioteki GSON z wcześniej zdefiniowanych klas DTO

### Przykład :

#### Wiadomość :

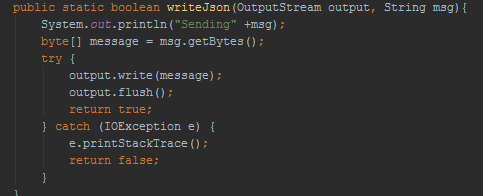
Posiada pola msg i from który póżniej uzupełniamy i parsujemy na JSON.   
To rozwiązanie ułatwia zmianę formy jsona w przyszłości.

#### Zamiana obiektu Message na Json:

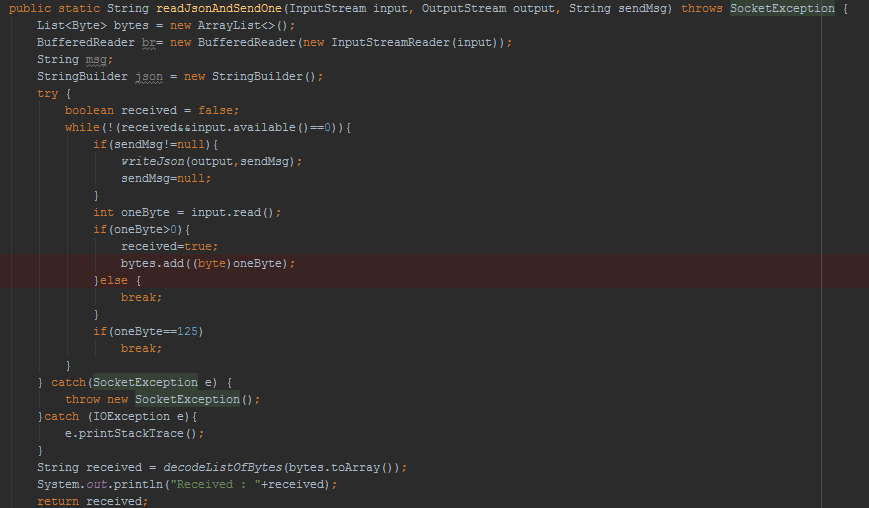


## Wysyłanie i odbieranie obiektu JSON

### Wysyłanie

  
Wysyłąnie polega na zamianie obiektu String na ciąg bitów i wypisanie go na output strem(+ obsługa wyjątków)

### Odbieranie



Jak widać metoda nie tylko odbiera ale także wysyła przed odebraniem wiadomośc jeśli do metody zostanie jakaś przekazana. Metoda została tak utworzona aby nie powtarzać kodu i była używalna w taki sposób w wielu miejscach (np. przy wysyłaniu i odbieraniu A i B)

Metoda w pętli while czyta tak długo dopóki nie otrzymała chociaż jednego bitu i nie ma już więcej dostępnych bitów lub otrzymała znak zamknięcia JSON-a „}” . Opcja z „}” została dodana ponieważ podczas testów udało mi się wytworzyć sytuacje w której w szybkim odstępie czasu dostałem dwa JSONy jeden z kluczem p i g a drugi z wyliczonym przez serwer B.

## Testy

Zostało przeprowadzonych wiele testów manualnych w których wykryłem i poprawiłem wiele problemów aplikacji , dodatkowo w trakcje zajęć wraz z kolega Adamem Zimnym przeprowadziliśmy test połączenia naszych dwóch aplikacji który odniósł sukces potwierdzająć że nasze aplikacje działają zgodnei z protokołem ustalonym na zajęciach.

# Wnioski :

* Używanie integera zamiast BigIntegera (long) powodowało wiele problemów związancch z niedokładnością obliczeń (np. przy wyliczaniu wielkich potęg)
* Klasa BigInteger pozwala na szybsze obliczanie sekretu za pomocą wbudowanej fnkcji powMod (potęga z modulo)
* Klasa Javy AlgorithmParameterGenerator pozwala w łatwy sposób wygenerować p i g

# Podsumowanie :

Przygotowana aplikacja spełnia wymagania projektu:

* Aplikacja poprawnie łączy się przez sieć z wykorzystaniem gniazd,
* Zaimplmenetowany został bezpieczny protokół wymiany kluczy Diffiego-Hellmana,
* Zaimplementowane zostały dwie metody szyfrowania (xor, cezar)
* Poprawnie obsługiwany jest zadany protokół komunikacyjny. Aplikacja poprawnie komunikuje się z innymi implementacjami przygotowanymi przez uczestników laboratorium,
* Metody szyfrowania można zmienić przed wysłaniem każdej wiadomości.