POLITECHNIKA WARSZAWSKA

WYDZIAŁ ELEKTRONIKI I TECHNIK INFROMACYJNYCH

**Protokoły Kryptograficzne**

**Dokumentacja projektu**

**Temat: Mobilny komunikator wykorzystujący**

**protokół Diffiego-Hellmana**

*Zespół:*

Tomasz Najda

Michał Hawryszko

*Prowadzący:*

mgr inż. Maciej Olewiński

*Data wykonania:*

01.12.2014r.

(Semestr 14Z)

# Wprowadzenie

Przedmiotem naszego projektu jest mobilny komunikator wykorzystujący uzgodnienia klucza prywatnego za pomocą protokołu Diffiego-Hellmana. Aplikacja ma umożliwiać komunikację 2 użytkowników. Po nawiązaniu połączenia powinno nastąpić automatyczna wymiana komunikatów w celu uzgodnienia prywatnego klucza. Następnie użytkownicy mogą wysyłać między sobą wiadomości. Wiadomość przed wysłaniem musi być zaszyfrowana dowolnym algorytmem szyfrującym z wykorzystaniem uzgodnionego klucza, a po odebraniu – odszyfrowana. Aplikacja powinna być wyposażona w odpowiednie logi umożliwiające śledzenie działania aplikacji bezpośrednio z urządzenia.

# Opracowanie teoretyczne

Założeniem protokołu Diffiego-Hellmana jest wygenerowanie tajnego klucza korzystając

z publicznego kanału łączności.

Alicja ========= Bob

Schemat:

1. Alicja i Bob ustalają ciało, w którym prowadzić będą obliczenia, czyli wybierają duży moduł pierwszy ***p*** i generator ***g*** grupy multiplikatywnej w GF(p) dużego rzędu modulo p.
2. Alicja wybiera dużą liczbę ***a*** z GF(p). Następnie oblicza **A** = ga mod p.
3. Bob wybiera dużą liczbę ***b*** z GF(p). Następnie oblicza ***B*** = gb mod p.
4. Alicja wysyła ***A*** do Boba.
5. Bob wysyła ***B*** do Alicji.
6. Alicja oblicza tajny klucz ***K*** = Ba mod p.
7. Bob oblicza tajny klucz ***K*** = Ab mod p

Łatwo zauważyć, że wartości podkreślone są prywatne (nie przechodzą przez publiczny kanał łączności), więc nikt nie może ich poznać. Pozostałe wartości są publicznie znane. Siłą algorytmu jest trudność obliczenia logarytmów dyskretnych w ciałach skończonych. Stosowanie takiego protokołu do uzgadniania tajnego klucza nie rozwiązuje jednak problemu „man in the middle”.

Alicja ========= Podsłuchiwacz ========= Bob

Problem polega na przechwyceniu całej komunikacji uzgodnienia klucza przez Podsłuchiwacza i podpięcia się pod wymianę. W wyniku tego komunikacja Alicji z Bobem rozdziela się na 2 komunikacje: Alicji z Podsłuchiwaczem i Podsłuchiwacza z Bobem.

Alicja ========= Podsłuchiwacz Podsłuchiwacz ========= Bob

Rozwiązaniem tego problemu jest zastosowanie algorytmów uwierzytelniania użytkowników, która jest zawarta w protokole Station-to-Station.

*Bibliografia artykuł 10,11.*

# Koncepcja rozwiązania problemu

Tworzymy aplikację na Androida z wykorzystaniem API 9. Wykorzystujemy prostą komunikację klient-serwer dla 2 urządzeń podłączonych do jednego routera. Jedno urządzenie stanowi serwer, drugie – klienta.

Działanie aplikacji:

1. Wybieramy czy aplikacja ma działać jako klient czy serwer,
2. Podajemy port połączenia oraz, jeśli to klient, podajemy ip serwera,
3. Uruchamiamy serwer,
4. Łączymy klienta do serwera,
5. Serwer, widząc dołączonego klienta, generuje p i g, a następnie automatycznie je wysyła do klienta.
6. Klient odbiera wartości. Następnie generuje a, oblicza A i wysyła A do serwera.
7. Serwer po odebraniu A, generuje swoje b, oblicza B i wysyła B do klienta.
8. Zarówno serwer jak i klient obliczają tajny klucz K.
9. Uzyskany klucz skracamy korzystając z SHA-2 (256 bitów).
10. Rozpoczynamy szyfrowaną komunikację z użyciem szyfru blokowego AES z wykorzystaniem 256-bitowego klucza.

Bezpośrednio w oknie czatu umieszczamy odpowiedni przycisk, który wyświetla lub chowa logi. Do logów zaliczamy wiadomości wysłane w celu uzgodnienia klucza oraz zaszyfrowane wiadomości.

*Bibliografia artykuły 4,5,6,9,11.*

Dobór parametrów szyfrowania:

Aby zapewnić bardzo duże bezpieczeństwo szyfrowania długości parametrów P i G powinny być odpowiednio 2048 bitów i 256 bitów. Jednak te parametry są stanowczo zbyt duże, aby aplikacja działała szybko i płynnie na smartphone’ie. Jest to spowodowane różnicą sprzętową pomiędzy Smartphonem, a komputerem stacjonarnym, bądź laptopem. Czas ustalania klucza na obydwu platformach różnił się co najmniej 10 krotnie, a testy były wykonywane na najnowszych modelach telefonów (Samsung Galaxy S3 i HTC One M8).

Przyczyny różnic:

* Długość życia baterii. Jest to najważniejsza przyczyna ograniczonej pracy procesorów w smartphone’ach. Zużycie energii jakiej potrzebowałby procesor, który dorównałby efektywności procesora PC, jest stanowczo za duże w porównaniu do możliwości baterii. Procesor w telefonie wykorzystuje od 4 do 11 wattów, gdzie procesory w komputerach wykorzystują średnio 80-90 wattów.
* Potrzeby rynku. Jeżeli zmusilibyśmy smartphone’a do wysokiego obciążenia zacząłby się nagrzewać co nie jest efektem pożądanym..
* Chłodzenie procesora. W telefonach nie jest dostępne takie chłodzenie, które umożliwiłoby pracę procesora „na wysokich obrotach”, dlatego też ich efektywność jest ograniczana do możliwości chłodzenia.

Uznaliśmy zatem, że poświęcimy czas wyliczania kluczy na rzecz szybkiego działania aplikacji. Wartości P i G są odpowiednio 512 i 32 bitowe, natomiast wygenerowany klucz jest długości 256 bitowej(SHA-2), co pozwala uzyskać 7 poziom odporności na ataki (z artykułu 3 w bibliografii).

*Bibliografia artykuły 1, 2 i 3.*

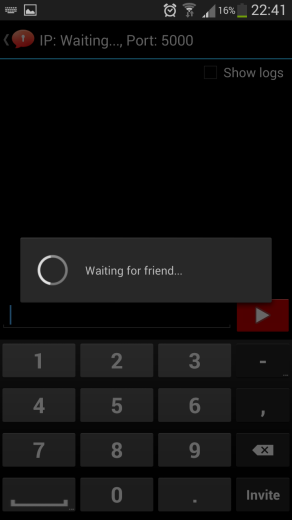
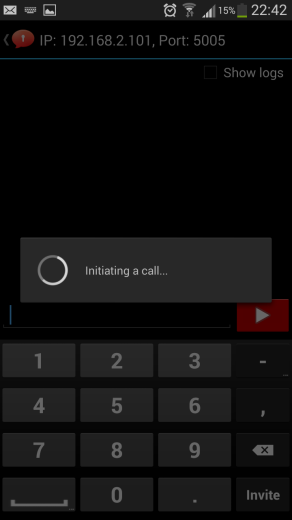
# Instrukcja użytkowania programu

## Okno startowe

## 

1. Twój adres IP,
2. Wybór czy aplikacja ma działać jako serwer (YES) czy jako klient (NO),
3. Adres IP drugiego użytkownika,
4. Port, na którym zestawione będzie połączenie (zakres: 0 – 65535),
5. Przycisk „Invite” rozpoczynający pracę serwera lub uruchamiający klienta oraz automatyczne połączenie z serwerem.

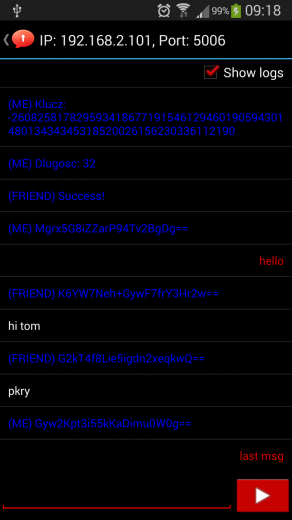
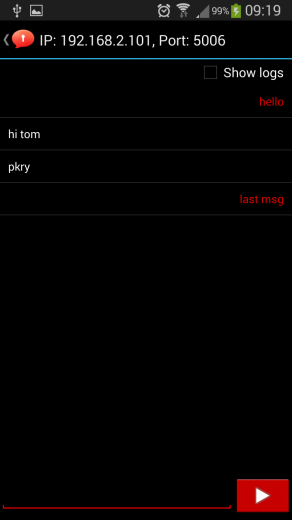
## Połączenie i wymiana klucza

**„Waiting for friend…”:** Komunikat wyświetlany po uruchomieniu serwera lub po wystartowaniu klienta, kiedy połączenie pomiędzy nimi nie zostało jeszcze zestawione. Po zestawieniu połączenia komunikat ulegnie zmianie oraz w aplikacji serwera nagłówek „IP: Waiting…” zostanie zamieniony na „IP: ADRES\_KLIENTA”

**„Initating a call…”:** Komunikat wyświetlany podczas automatycznego ustalania tajnego klucza zaraz po zestawieniu połączenia. Rozmowa będzie możliwa po wyłączeniu komunikatu.

## Główne okno czatu

Po ustaleniu klucza możliwa jest rozmowa tekstowa. Klawiatura wirtualna pokaże się po kliknięciu na pole tekstowe na dole ekranu. Tutaj wpisujemy wiadomość. Następnie, aby wysłać wiadomość, klikamy na czerwony prostokąt z białym trójkątem.

W głównym oknie czatu możemy włączyć wyświetlanie logów. Należy to zrobić zaznaczając w prawym górnym rogu pole „Show logs”. Wyłączenie logów następuje po odznaczeniu pola.

Kolory wiadomości:

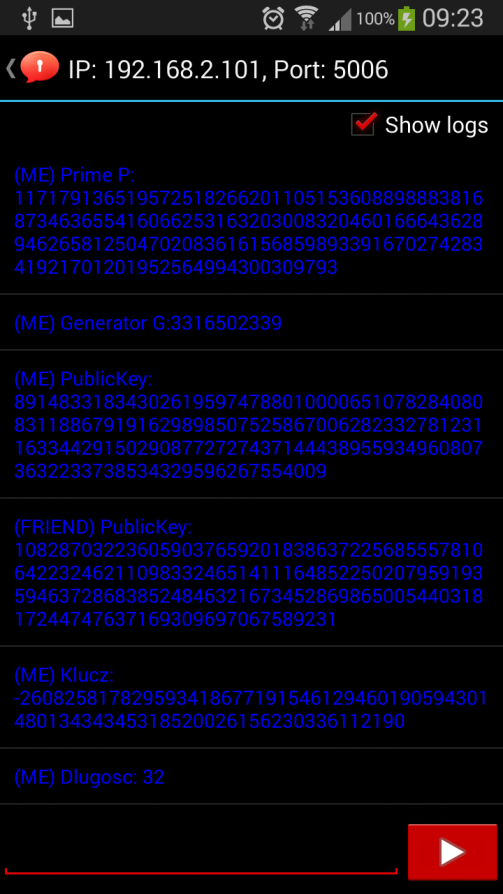
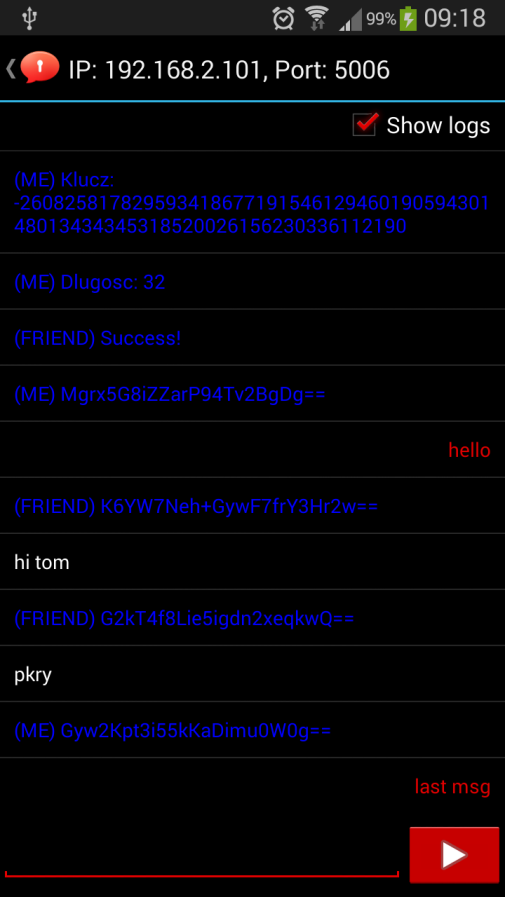
Niebieski: Logi

Czerwony: Wiadomości wysłane

Białe: Wiadomości odebrane

# Raport z testów aplikacji

Pierwszym testem jaki wykonaliśmy był test poprawności ustalania kluczy i szyfrowania/deszyfrowania wiadomości. W tym celu wprowadziliśmy dodatkową opcję w naszej aplikacji o nazwie „logs”. Gdy jest aktywna na ekranie czatu oprócz zwykłych wiadomości wyświetlane są ustalone klucze, ich długość oraz zaszyfrowane wiadomości. Testy przebiegły zgodnie z naszymi oczekiwaniami, długości kluczy są poprawne, a wiadomości są poprawnie szyfrowane i deszyfrowane.

Drugim testem jaki wykonaliśmy był test wydajnościowy. Porównywaliśmy czas w jakim wartości kluczy obliczy smartphone, a czas w jakim obliczy to komputer. Wyniki tego testu poparły naszą wcześniejszą wypowiedź na temat różnicy sprzętowej i wydajnościowej pomiędzy tymi urządzeniami. Poniżej znajdują się zrzuty ekranu zawierające czas wykonania zadania.

Wynik otrzymany na komputerze:

Screenshot at 19:12:59

Wynik otrzymany na telefonie:

Wykonując kilka takich testów wydajnościowych doszliśmy do najbardziej optymalnego rozwiązania. Czas w jakim zestawiane są klucze na telefonie jest na tyle krótki, że nie powoduje to negatywnych emocji użytkownika, ale na tyle bezpieczny, że złamanie go zajmie dużo czasu. Poniżej zamieszczony jest zrzut ekranu, który pokazuje czas zestawienia kluczy.

# Podsumowanie

W trakcie projektowania aplikacji napotkaliśmy wiele problemów i komplikacji, wielokrotnie musieliśmy „pójść na kompromis” pomiędzy bezpieczeństwem, a szybkością działania aplikacji. Jednak po zgłębieniu wiedzy na temat algorytmu Diffie-Hellman’a, bezpieczeństwa algorytmów oraz metod łamania szyfrów doszliśmy do ostatecznego rozwiązania. Nasza aplikacja nie jest w 100% odporna na ataki, ale jest na tyle odporna, aby napastnik poświęcił sporo czasu na złamanie szyfru.

Zaprojektowaliśmy również sposób na powstrzymanie ataku Man In the Middle, który jest niezwykle groźny jeżeli chodzi o ten protokół, jednak nie jest on zaimplementowany w aplikacji.

# Bibliografia

1. <http://www.digitaltrends.com/computing/why-your-smartphone-wont-be-your-next-pc/>
2. <http://www.cnet.com/news/smartphones-unlocked-understanding-processors/>
3. <http://www.keylength.com/en/3/>
4. <http://searchsecurity.techtarget.com/definition/Diffie-Hellman-key-exchange/>
5. <http://security.stackexchange.com/questions/45963/diffie-hellman-key-exchange-in-plain-english>
6. <http://security.stackexchange.com/questions/47204/dh-parameters-recommended-size>
7. <http://www.java2s.com/Tutorial/Java/0490__Security/ImplementingtheDiffieHellmankeyexchange.htm>
8. <http://stackoverflow.com/questions/14612615/what-size-primes-are-needed-for-a-good-diffie-hellman-key-generation>
9. <http://www.cryptopp.com/wiki/Diffie-Hellman>
10. <http://www.veracode.com/security/man-middle-attack>
11. Wykłady z przedmiotu PKRY