Komunikator z szyfrowaniem

TL;DR

Należy przygotować komunikator (chat) klient-serwer wspierający bezpieczną wymianę sekretu (protokół Diffie-Hellman) oraz obsługujący zadany format komunikacji. Im lepiej zrobione tym lepiej ocenione.

Wersja dłuższa

Komunikator musi wspierać protokół Diffiego-Hellmana oraz szyfrowanie wiadomości zgodnie z następującym formatem danych opartym o JSON:

Stage	A (client)	B (server)
1	{ “request”: “keys” } →	
2		← { “p”: 123, “g”: 123 }
3	{ “a”: 123 } →	← { “b”: 123 }
4	{ “encryption”: “none” } →	
5	{ “msg”: “...” } →	← { “msg”: “...” }

1. Po połączeniu do serwera klient prosi o liczby p oraz g.
2. Serwer wysyła do klienta liczby p oraz g.
3. Serwer i klient wymieniają się publicznymi wartościami A oraz B:
    a. Klient wysyła do serwera obliczoną wartość A.
    b. Serwer wysyła do klient obliczoną wartość B.
    UWAGA: a) oraz b) mogą nastąpić w dowolnej kolejności
4. [OPCJONALNIE] Klient wysyła do serwera informację o żądanym sposobie szyfrowania wiadomości.
    UWAGA: Krok ten jest opcjonalny. Jeżeli klient nie wyśle tej informacji, to strony przyjmują domyśle szyfrowanie ustawione na “none”.
5. Klient oraz serwer wymieniają się szyfrowanymi wiadomościami.

Treść wiadomości powinna być zakodowana za pomocą base64 przed umieszczeniem jej w strukturze JSON:
`base64(encrypt(user_message))`

Wspierane metody szyfrowania:

- none - brak szyfrowania (domyślne)
- xor - szyfrowanie OTP xor jednobajtowe (należy użyć najmłodszego bajtu sekretu)
- cezar - szyfr cezara

Kryteria oceny

1. Kompletność rozwiązania
    a. Czy w ogóle działa?
    b. Czy format komunikacji jest poprawny?
    c. Czy protokół DH został zaimplementowany poprawnie?
2. Jakość pomysłu
    a. Czy serwer wspiera wielu klientów?
    b. Czy klient/serwer wspiera zmianę sposobu szyfrowania w locie?
    c. Czy parametry DH są stałe, konfigurowalne czy też dynamicznie generowane?
        i. Czy są różne dla każdego klienta?
    d. Czy dozwolona jest dowolna kolejność wymiany wartości A i B?
3. Jakość kodu
    a. Czy są testy jednostkowe?
    b. Czy ogólna jakość kodu jest dobra? (vide *code guidelines* dla wybranego języka/technologii)
    c. Czy poprawnie użyto OOP/FP?
4. Dokumentacja
    a. Czy jest README.md?
        i. Czy opisuje wymagania systemowe?
        ii. Czy opisuje sposób uruchomienia i/lub kompilacji?
        iii. Czy zawiera dokumentację użytkową projektu?
        iv. Czy opisuje cel/przeznaczenie projektu?
    b. Czy kod jest udokumentowany?
        i. Czy są komentarze dla funkcji/metod/klas?
        ii. Czy komentarze są zgodne z uznanymi standardami (javadoc, doxygen, yard, clojure.spec itd.)?
        iii. Czy język komentarzy jest zgody z językiem nazewnictwa zmiennych/metod/funkcji/klas?
5. Praca z gitem
    a. Czy praca została wykonana przyrostowo?
    b. Czy *commit messages* są zrozumiałe i opisowe?
    c. Czy praca została wykonana w nowym *branchu*?
6. Raport
    a. Czy raport jest w wymaganym formacie (PDF)?
    b. Czy raport zawiera informacje o celu zadania?
    c. Czy raport streszcza sposób wykonania zadania?
    d. Czy w raporcie zawarto argumentację za konkretnymi rozwiązaniami?

e. Czy w raporcie są przemyślenia/wnioski?