Bezpieczeństwo Systemów Komputerowych

Zadanie 4: Współdzielenie tajemnicy

Firma, która ma n-osobowy zarząd chciałaby pozwolić na dostęp do największych firmowych tajemnic tylko wtedy, gdy zbierze się cały zarząd. Tobie zostało powierzone zadanie zaimplementowanie systemu o strukturze Klient-Serwer, który pozwoli na takie zabezpieczenie firmowych tajemnic. System ma wykorzystywać system haseł jednorazowych Lamporta oraz schemat dzielenia sekretu zdefiniowany przez Shamira (opartego o wielomiany w ciele skończonym modulo p) i działać zgodnie z protokołem:

1. Klient posiada:

   - sekret zerowy h_0
   - indeks i (początkowo ustawiony na 1)
   - ograniczenie K
   - punkt P, który jest punktem należącym do wielomianu, znanego serwerowi

   Klient wysyła do serwera krotkę[1]:

   $$(\{0\}, \{1\})$$

   gdzie:

   $\{0\}$ jest identyfikatorem klienta

   $\{1\}$ jest napisem obliczonym ze wzoru:

   $$E_{K_{KS}}(T)$$

   gdzie

   $E_{K_{KS}}(M)$ oznacza napis, który jest wynikiem szyfrowania symetrycznego napisu M przy użyciu klucza K_{KS}, który jest znany tylko klientowi i serwerowi

   T jest krotką:

   $$(\{2\} : \{3\} : \{4\})$$

   gdzie:

   $\{2\}$ identyfikatorem klienta

   $\{3\}$ wartość obliczoną ze wzoru:

   $$H^{K-i}(h_0)$$

   gdzie $H^x(y)$ oznacza, x-krotne złożenie funkcji haszującej H na elemencie y

2. Serwer zna:

   - liczbę pierwszą p, która definiuje ciało skończone reszt z dzielenia przez liczbę p
   - wielomian W w ciele zadanym przez p, który może zostać odtworzony z punktów od wszystkich klientów. Stopień wielomianu to $n-1$

   oraz dla każdego klienta zna:

   - jego aktualny indeks i (początkowo ustawiony na 1)
   - ograniczenie K
   - wartość

     $$Secret = H^K(h_0)$$

     gdzie $H^x(y)$ oznacza, x-krotne złożenie funkcji haszującej H na elemencie y

[1] Przez krotkę mamy rozumieć `tuple`

Serwer sprawdza, czy wartość przesłana przez klienta jest prawidłowa. Między innymi istotne jest, żeby wartość: $H\left(H^{K-i}(h_0)\right)$ była równa wartości $Secret$. Jeżeli cokolwiek się nie zgadza, to odsyła klientowi jawnym tekstem słowo 'ERROR', jeżeli wszystko się zgadza to wysyła do klienta zaszyfrowany kluczem K_{KS} napis 'OK'. Gdy wszystko się zgadza, serwer aktualizuje indeks i oraz wartość hasha, który przechowuje dla danego klienta. **Uwaga:** Serwer nie może pozwolić się zalogować użytkownikowi o nazwie, która jest wśród nazw wcześniej zalogowanych użytkowników.

3. Klient odbiera wiadomość od serwera, jeżeli logowanie się powiodło, zwiększa swoją wartość indeksu i i przechodzi do następnego kroku. Jeżeli, się nie powiodło ma przerwać swoje działanie.

4. Klient wysyła do serwera napis:
$$E_{K_{KS}}(M)$$
gdzie:

   $E_{K_{KS}}(M)$ oznacza napis, który jest wynikiem szyfrowania symetrycznego napisu M przy użyciu klucza K_{KS}, który jest znany tylko klientowi i serwerowi

   M jest napisem w postaci:
$$\{4\} : \{5\}$$
   gdzie

   {4} współrzędna x-owa punktu klienta

   {5} współrzędna y-owa punktu klienta

5. Serwer czeka, aż wszyscy spośród n klientów (n jest zadane przez stopień wielomianu W) prześlą mu swój sekret. Gdy wszyscy klienci prześlą mu swój sekret, sprawdza czy punkty zadają oczekiwany wielomian W. Jeżeli punkty nie definiują tego samego wielomianu, to wszyscy klienci dostają jawnym tekstem napis 'ERROR', w przeciwnym wypadku, każdy klient dostaje zaszyfrowany właściwym kluczem napis 'OK'

W ramach rozwiązania, mają państwo wysłać na system BaCa archiwum ZIP, które będzie zawierało 3 pliki (archiwum nie może zawierać katalogu, który będzie zawierał te pliki):

- Server.py
- Client.py
- Utils.py

I żadnego pliku więcej!

1 Zawartość pliku Server.py

W pliku Server.py ma być zdefiniowana klasa Server (dziedzicząca po klasie threading.Thread), która będzie implementowała funkcjonalność serwera z wyżej opisanego protokołu. Klasa ma posiadać metody:

__**init**__ , która ma przyjmować argumenty:

input_queue - kolejka służąca do wysyłania komunikatów przeznaczonych dla serweraa[2]

output_queue - kolejka służąca do odbierania komunikatów od serwera

p - rozmiar ciała[3], można założyć, że ten argument ma poprawną wartość

passwords - słownik[4], który dla każdego identyfikatora klienta będzie przechowywał krotkę:
$$(\{6\}, \{7\}, \{8\})$$
gdzie:

{6} aktualna wartość $Secret$ (początkowo: $H^K(h_0)$)

{7} ograniczenie K

{8} aktualna wartość indeksu i (początkowo 1)

[2] przez kolejkę mamy rozumieć obiekt typu Queue.Queue

[3] Oznaczenie zgodne z opisem protokołu

[4] obiekt dict

keys - słownik, który dla każdego identyfikatora klienta, będzie przechowywał klucz do symetrycznego szyfrowania komunikatów wymienianych z klientem

polynomial - wielomian zapisany jako lista współczynników. Na przykład, wielomian

$$W(x) = x^2 + 4 \cdot x + 6 \mod p$$

będzie zapisany jako lista:

$$[1, 4, 6]$$

run - metoda implementująca komunikację z przychodzącymi klientami. Dla każdego przychodzącego klienta, tworzony jest nowy wątek, który będzie implementował komunikację z jednym konkretnym klientem (używając konkretnych kolejek). Czyli każdy przychodzący klient wysyła jakąś wiadomość do serwera (protokół nie definiuje jaka to jest wiadomość) w odpowiedzi dostaje krotkę:

$$(q_1, q_2)$$

gdzie

q_1 służy do wysyłania komunikatów do nowotworzonego wątku

q_2 służy do wysyłania komunikatów do klienta

finish - metoda, która kończy działanie wątka głównego serwera (wątki odpowiedzialne za komunikację powinny same się zakończyć, po niepowodzonym logowaniu klienta albo po zweryfikowaniu sekretów)

2 Zawartość pliku Client.py

W pliku Client.py ma być zdefiniowana klasa Client (dziedzicząca po klasie threading.Thread), która będzie implementowała funkcjonalność klienta z powyżej opisanego protokołu. Klasa ma posiadać pole (oczywiście klasa powinna mieć więcej pól, jednak poniżej wymienione jest częścią API i inne klasy będą korzystały z poniższej listy):

finished_successfully - flaga boolowska, która być ustawiona na True, tylko jeżeli Klient będzie uważał, że ma właściwy klucz dla komunikacji z Serwerem. Początkowa wartość to None

Klasa ma posiadać metody:

__init__ , która ma przyjmować argumenty:

name - napis identyfikujący klient

h_0 - sekret zerowy dla protokołu Lamporta[5]

K - ograniczenie na ilość logowań przy użyciu protokołu Lamporta[6]

key - klucz do szyfrowanej komunikacji z serwerem

serwer_input_queue - kolejka, którą klient wykorzysta do nawiązania połączenia z serwerem

server_output_queue - kolejka, którą klient wykorzysta do odebrania od serwera krotki z kolejkami do komunikacji z dedykowanym dla niego wątkiem serwera

secret - punkt (x, y), który jest częścią dzielonego sekretu należącą do tego klienta

run - metoda implementująca zachowanie jednego klienta w powyższym protokole. Metoda ma się skończyć sama po niepowodzonym logowaniu albo po zweryfikowaniu sekretów. Przy niepowodzeniu, pole finished_successfully powinno przyjąć wartość False, przy sukcesie pole finished_successfully powinno przyjąć wartość True

3 Zawartość pliku Utils.py

W pliku Utils.py ma się znaleźć jedna klasa Cipher, która będzie posiadała metody

__init__ , która ma przyjmować argumenty:

key - Klucz dla szyfrowania symetrycznego, który może być napisem lub liczbą

encrypt - metoda która przyjmuje jeden argument :

[5] Oznaczenie zgodne z opisem protokołu

[6] Oznaczenie zgodne z opisem protokołu

m który jest napisem. Wynikiem funkcji ma być napis, który będzie wynikiem szyfrowania napisu m przy użyciu szyfrowania symetrycznego z key jako kluczem

decrypt - metoda która przyjmuje jeden argument:

c który jest napisem. Wynikiem funkcji ma być napis, który będzie wynikiem odszyfrowania napisu c przy użyciu szyfrowania symetrycznego z key jako kluczem

Klasa Cipher ma implementować dowolne szyfrowanie smetryczne, dla którego prawdą będzie:

- m == encrypt(decrypt(m))
- m == decrypt(encrypt(m))
- m != encrypt(m)

Plik Utils.py powinien dodatkowo zawierać funkcję:

hash_function która przyjmuje 1 argument m, który jest napisem. Wynikiem ma być wynik funkcji skrótu obliczonej dla napisu m

4 Uwagi implementacyjne

- Zadanie będzie uruchamiane w interpreterze pythona w wersji 2.6
- Każdy komunikat może być stringiem "ERROR". Państwa kod, nie powinien wypisywać żadnych komunikatów na standardowe wyjście.
- Zwracam uwagę, że Państwa kod może tworzyć dodatkowe wątki. Ważne jest, aby wszystkie wątki zakończyły się samoistnie (oprócz wątka Server, który ma się zakończyć po wywołaniu metody finish). Jeżeli jakiś wątek będzie żył po zakończeniu protokołu, to test zakończy się z komunikatem błąd wykonania.
- Ważne jest, że w razie jakichś błędów, czekanie na akcję innego wątku może się nie zakończyć. Ważne, żeby wszędzie gdzie Państwa program czeka na inny wątek był ustawiony maksymalny czas przez jaki program będzie czekał. Musicie Państwo go tak dobrać, aby wątki się zakończyły w czasie jaki jest przewidziany przez metodę join. We wszystkich testach timeout metody join jest taki jak w teście jawnym.

5 Uwagi odnośnie testowania

- Państwa klasy będą w różnych konfiguracjach testowane z klasami napisanymi przez autora zadania.
- Żaden z Państwa plików nie będzie nadpisywany lub też edytowany. Więc wszędzie tam gdzie Państwo napiszecie linijkę:
  from Utils import hash_function
  możecie mieć pewność, że wywołanie funkcji hash_function wywoła funkcję z Państwa pliku hash_function.
- Żaden z państwa plików nie może zawierać słowa "pylint" - rozwiązania, które takie słowo będą posiadały, będą automatycznie odrzucane.
- W czasie testowania Państwa kod będzie analizowany za pomocą narzędzie `pylint`[7] z dodatkowymi opcjami:

```
pylint --disable=C0111,R0922,C0103 --max-args=10 --max-locals=20 \
       --max-line-length=120 --max-attributes=12 Server.py Client.py Utils.py
```

6 Termin i punktacja

- Termin na oddanie zadania mija 3 tygodnie po ogłoszeniu zadania na stronie BaCa (jednocześnie zostanie umieszczony post na forum wykładu na platformie pegaz)
- Za zadanie można uzyskać 50 punktów, z czego 10 punktów będzie wynikiem działania programu `pylint` na przesłanym kodzie źródłowym.

[7] w wersji 0.21.1

7 Test jawny

```
from Queue import Queue
from Client import Client
from Server import Server
import sys
from threading import active_count, enumerate, current_thread

# jest wielomian:x^2+4x+6 mod 31
# i sa trzy punkty:
# (1,11)
# (3,27)
# (7,21)

passwords = {'alice':   ('2a414846467aadb9872f029787224bdb', 50, 1),
             'bob':     ('b1cf97c60932ab006be914b668ae8f46', 50, 1),
             'charlie': ('11250c7c4e996c15c32fb9cb43695c5d', 50, 1)
            }
keys = {'alice': '1234', 'bob': '4321', 'charlie': '5678'}

server_input_queue = Queue()
server_output_queue = Queue()

server = Server(input_queue=server_input_queue,
                output_queue=server_output_queue,
                p=31,
                passwords=passwords,
                polynomial=[1,4,6],
                keys=keys)
alice = Client(server_input_queue=server_input_queue,
               server_output_queue=server_output_queue,
               name='alice',
               h_0='9944',
               K=50,
               key='1234',
               secret=(1, 11))
bob = Client(server_input_queue=server_input_queue,
             server_output_queue=server_output_queue,
             name='bob',
             h_0='6850',
             K=50,
             key='4321',
             secret=(3, 27))
charlie = Client(server_input_queue=server_input_queue,
                 server_output_queue=server_output_queue,
                 name='charlie',
                 h_0='5478',
                 K=50,
                 key='5678',
                 secret=(7, 21))

assert alice.successfully_finished is None
assert bob.successfully_finished is None
assert charlie.successfully_finished is None

server.start()

alice.start()
bob.start()
charlie.start()

killed = False
```

```
alice.join(timeout=30)
bob.join(timeout=30)
charlie.join(timeout=30)
server.finish()
server.join(timeout=30)

if active_count() > 1:
    for t in enumerate():
        if t != current_thread():
            print t
            t._Thread__stop()
    sys.exit(1)

assert alice.successfully_finished is True
assert bob.successfully_finished is True
assert charlie.successfully_finished is True
```