Programozói dokumentáció

A NHZ-hoz

1. *A projekt felépítése*

   A projekt alatt összesen 6 db C fájl és 5 db header készült. Ezek a következők:

   - main.c
   - rsa.c – rsa.h
   - account.c – account.h
   - card.c – card.h
   - menu.c – menu.h
   - transfer.c – transfer.h

2. *Funkcióik, dokumentációjuk*

   a. **main.c**: Vezérli a programot. Ide van include-olva az összes header fájl. A vezérlés switch-en keresztül történik, illetve két while ciklus gondoskodik arról, hogy ne csak egyszer fusson le a program, hanem amíg a felhasználó nem lép ki, addig ne álljon le a program. Két boolean típusú változó addig hamis, amíg a felhasználó nem jelentkezik ki és nem lép ki.

   b. **rsa.c – rsa.h**: Itt történik a titkosítás. Ez az RSA algoritmuson alapszik.

   Az első feladat a két prím deklarálása. Ezért a *p()* és a *q()* függvények felelnek, hogy a többi C fájlban is meg lehessen hívni őket.

   Ezután fejeztem ki az *e* számot az *e()* függvényben. Ez az a szám, amit ha az üzenet (másik szám) hatványául veszek és veszem a két prím szorzatának modulusával, akkor a titkosított üzenetet kapom meg. Gyakran használják a 65537-es számot, így a programban is ezt használtam.

   A két prím szorzatával az *N()* függvény tér vissza. Ez gyakran van használva a modulusoknál.Fontos még az Euler-féle fí függvény használata N-re. Ezért a *fiN()* függvény felel.

   Az egyik legfontosabb függvény a *RepSqr()* (Repeated Squaring). Az ismételt négyzetre emelés segítségével nagy hatványokat is könnyen és gyorsan ki tudunk számolni úgy, hogy kevesebb, mint 64 bites integerben is elfér.

   A *Creating_d()* függvény a dekódolásnál játszik fontos szerepet,ugyanis a kódolt üzenetet d-edik hatványra emelve visszakapjuk az eredeti üzenetet.

   A kódolást a *C()* függvény végzi. felhasználja az ismételt négyzetre emeléses függvényt is, hogy gyorsan tudjon számolni.

   A dekódolt üzenettel a *Decode()* tér vissza.

   c. **account.c – account.h**: Ez a fájl a számlanyitásért, beolvasásért felel.

   A lines() függvény azt mondja meg, hogy van-e számla. Ha nincs, akkor 0, ha van, akkor 1. Megszámolja a sortöréseket.

   Az *accCreate(int N, int e)* végzi a számla elkészítését. Négy elemből áll: számlaszám (N), ePIN, Énazonosító és az e szám.

   Az *accRead()* beolvassa a számlaszámot, és visszatér az értékével.

Az *accLogin()* meghatározza, hogy helyes-e a beírt Énazonosító, vagy sem. True, ha helyes, false, ha nem.

A *changeID()* az énazonosító megváltoztatását végzi.

d. **card.c – card.h**: A kártyával kapcsolatos funkciókat látja el.

A createCard() a kártya létrehozásáért felel. Ez 3 számot tartalmaz: CVC, lejárati dátum, kártyaszám. A lejárati dátum változhat, attól függően is, hogy mikor futtatjuk a programot. Decemberben már 28/12 lesz ráírva.

A *readCard()* a kártyaadatok kiírásánál játszik szerepet. Körbecsillagozva megkapjuk a „kártyát”.

e. **menu.c – menu.h**: A menü kiírásáért felelős fájl.

A *login_menu()* kiírja a bejelentkező menü pontjait.

A *main_menu()* a már bejelentkezett felhasználó számláján és kártyáján végezhető utasítások listáját írja ki.

d. **transfer.c – transfer.h**: Az utalásért felelős fájl.

A *transferlines()* felel a fájlban lévő sorok számlálásáért.

A *transfering()* ír a fájlba minden utalás után.

A *readTransfering()* kiolvassa és kiírja az utalási előzményeket.

3. *Adatszerkezetek*
   a. **card***:* cvc: a kártya cvc számának tárolása; lejaratev, lejaratho: a lejárati dátum; cardnum: a kártyaszám
   b. **account***:* ID: az énazonosító; num: számlaszám; pin: az ePIN; e: az e szám tárolása
   c. **transfer***:* numfrom: a saját számlaszám; numto: címzett számlaszám; amount: utalni kívánt pénz mennyisége

Töttös Balázs

F0V56I