HÁZI FELADAT

Programozás alapjai 2.

Töttös Balázs

F0V56I

2024.04.14.

Tartalom

1. Feladat

A választott feladatom egy banki alkalmazás elkészítése. Manapság egyre fontosabbá válik, hogy adataink biztonságban legyenek. Ehhez titkosítást alkalmazunk, ami segít a privát adataink elrejtésében úgy, hogy azt csak a megfelelő kulccsal lehessen megismerni. A program egy egyszerű bemutatása a mobilbank/e-bankolás menetének. A program használata során a felhasználó megismerheti a titkosítás fontosságát a modern pénzügy világában.

2. Feladatspecifikáció

A program megvalósítja az RSA-algoritmust, amely egy titkosítási módszer. Az RSA három alapvető lépésből áll: kulcsok generálása, titkosítás és visszafejtés. A felhasználó által megadott szöveg vagy fájl a program által titkosításra kerül, és az eredmény egy fájlban kerül tárolásra. Csak a program használatával és a megfelelő kulcsokkal lehet visszafejteni az eredeti tartalmat. A felhasználó így biztonságosan kommunikálhat másokkal, anélkül hogy aggódnia kellene az információk illetéktelen hozzáférésétől.

3. Pontosított feladatspecifikáció

A feladatom egy komplex program elkészítése, amely képes banki ügyintézések szimulálására. Ilyenek például:

- Utalás,
- Számlák létrehozása,
- Kártya igénylése,
- Jelszóváltoztatás,
- Számla- és kártyaadatok megtekintése.

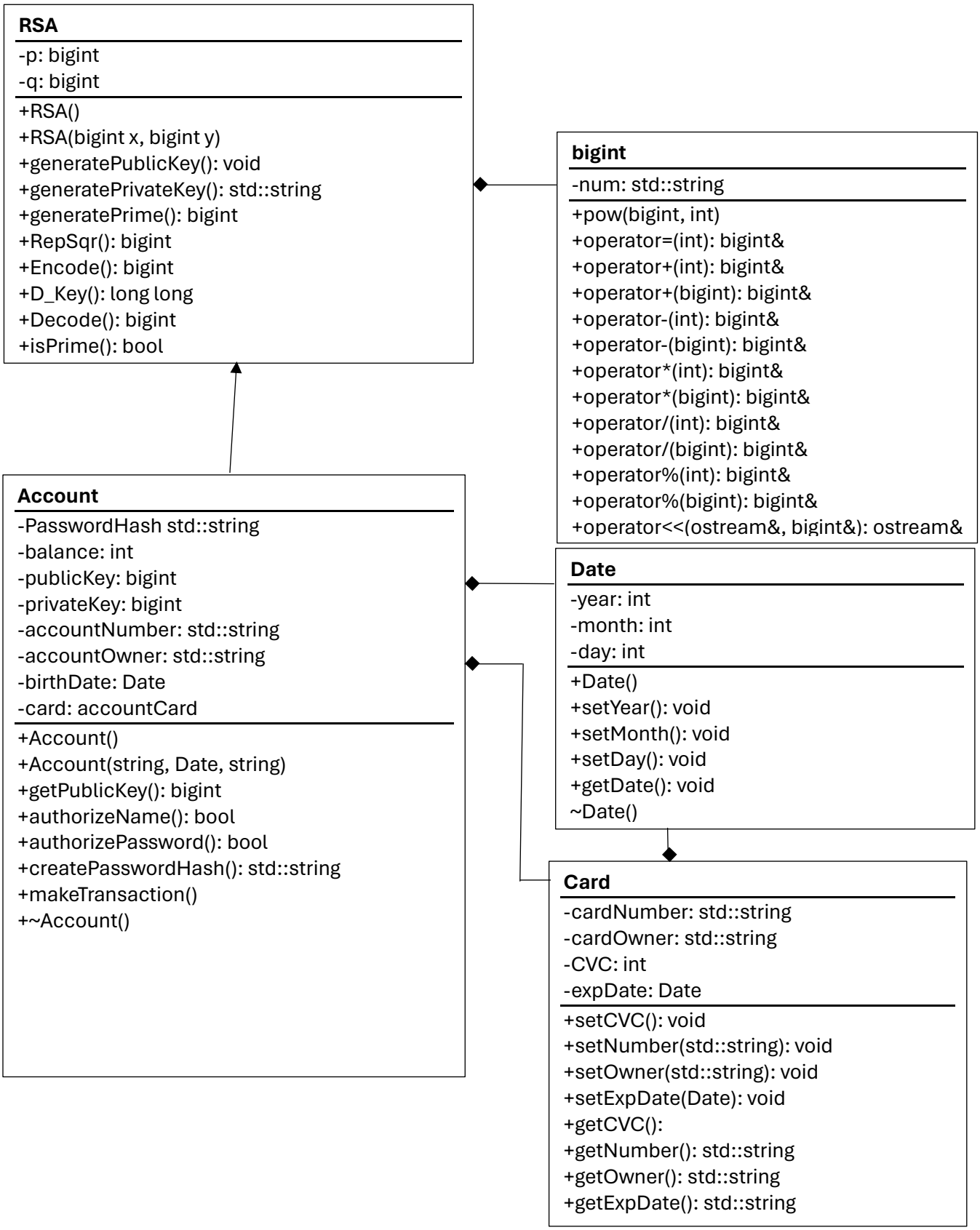
Ehhez létre kell hozni egy olyan szám típust, amely képes akár 1024 bites számműveletek végrehajtására kevesebb, mint 1 másodperc alatt, valamint ezt kell felhasználni az RSA algoritmus megvalósításánál. Ennek a teszteléséhez kellően nagy számokon végrehajtott szorzást, hatványozást, és osztást fogok alkalmazni. A másik nagyobb algoritmus a hashelés. A jelszót hashelve fogom tárolni, hogy illetéktelenül ne léphessen be senki.

4. Terv

A szám típus egy std::string típusból áll, és az operátorok segítségével valósítom meg a funkciókat. Ezek a normál írásbeli összeadás, kivonás, szorzás, osztás algoritmusait alkalmazzák. Valamint hozzátartoznak az istream, ostream operátorai is. Ezek azért kellenek, hogy kellően nagy számokat tudjunk generálni (512 bites számokat), ezáltal a titkosított adat emberi időn belül nem visszanyerhető. Ezeket a normál integerek nem tudják megvalósítani, hiszen egy unsigned long long integer is csak 64 bites számot tud tárolni.

A hashelés vödrös hash függvény lesz. Ez nem lesz túl nagy, de nem is kell akkora, amekkora az SHA-2 szabvány szerinti minimális méret (224).

Az osztálydiagram a következőképpen épül fel:


RSA
-p: bigint
-q: bigint
+RSA()
+RSA(bigint x, bigint y)
+generatePublicKey(): void
+generatePrivateKey(): std::string
+generatePrime(): bigint
+RepSqr(): bigint
+Encode(): bigint
+D_Key(): long long
+Decode(): bigint
+isPrime(): bool
bigint
-num: std::string
+pow(bigint, int)
+operator=(int): bigint&
+operator+(int): bigint&
+operator+(bigint): bigint&
+operator-(int): bigint&
+operator-(bigint): bigint&
+operator*(int): bigint&
+operator*(bigint): bigint&
+operator/(int): bigint&
+operator/(bigint): bigint&
+operator%(int): bigint&
+operator%(bigint): bigint&
+operator<<(ostream&, bigint&): ostream&
Account
-PasswordHash std::string
-balance: int
-publicKey: bigint
-privateKey: bigint
-accountNumber: std::string
-accountOwner: std::string
-birthDate: Date
-card: accountCard
+Account()
+Account(string, Date, string)
+getPublicKey(): bigint
+authorizeName(): bool
+authorizePassword(): bool
+createPasswordHash(): std::string
+makeTransaction()
+~Account()
Date
-year: int
-month: int
-day: int
+Date()
+setYear(): void
+setMonth(): void
+setDay(): void
+getDate(): void
~Date()
Card
-cardNumber: std::string
-cardOwner: std::string
-CVC: int
-expDate: Date
+setCVC(): void
+setNumber(std::string): void
+setOwner(std::string): void
+setExpDate(Date): void
+getCVC():
+getNumber(): std::string
+getOwner(): std::string
+getExpDate(): std::string

Talán az egyik legfontosabb algoritmus a program szempontjából az RSA-algoritmus. Ez a következőképpen épül fel:

- generatePrime(): Arra szolgál, hogy prímet adjon vissza. Ez kell p és q értékeknek, ami aztán a privát és a publikus kulcsot adja.
- RepSqr(a, b): Ismételt négyzetre emelés módszere (a^b). Ez a nagyobb hatványozások szempontjából hasznos algoritmus, ugyanis nem konkrét értéket számol, hanem csak egy adott modulussal kapott maradékát. Habár a szám típusunk képes kezelni nagy számok hatványozását kellően gyorsan, nagyobb számok esetén már exponenciálisan nő a végrehajtás ideje.
- Encode(a, b): RepSqr() eredményét veszi N modulussal, és azt adja vissza. Ez a kódolási függvény.
- D_Key(): Elkészít egy d visszafejtő kulcsot, amivel visszakapjuk a kódolt eredményből az eredeti adatot.
- Decode(C, d): C kódolt üzenetet dekódolja d segítségével. Itt újfent az ismételt négyzetre emelés módszerét alkalmazzuk.
- generatePublicKey(): A nyilvános kulcsot írja be az objektumba (N, e).
- generatePrivateKey(): A titkos kulcsot írja be az objektumba (N, d).

5. Kezelés

A kezelés nagyon egyszerű, felhasználóbarát. A programablakban kell dolgozni, számok begépelésével (pl.:1. Főoldal; 2. Beállítások…). A felhasználó a titkosítással nem kell foglalkozzon, ugyanis ez a programon belül történik. Természetesen a felhasználó be kell jelentkezzen, amihez először létre kell hozzon egy profilt. Itt be kell állítani a jelszót, a nevet, születési dátumot, majd ha ezek megvannak, bejelentkezhet. A kijelentkezést is meg lehet valósítani, de automatikusan is kijelentkeztet, ha nem csinálunk semmit.

6. Funkciók

A titkosításhoz kellően nagy számok kellenek, és kellően gyorsan kell végrehajtani az utasításokat rajta. Ehhez az kell, hogy a program képes legyen bármekkora számokkal dolgozni, ezeken emberi idő alatt hatványozást végezni és később pedig ezeket a számokat felhasználni a titkosításhoz. A titkosítás lépéseit a program nem jelzi, csak azt, hogy sikerült-e, vagy sem. Ezt a titkosítást több helyen is felhasználja a program: profil létrehozásánál, utalásnál, kártya adatainak tárolásánál, felhasználói adatok tárolásánál, stb. Ezek a funkciók kulcsfontosságúak a program működése szempontjából.