Szimmetrikus titkosítás és használata

Javaban

Fülöp Márk, 10.D

2016. május 17.

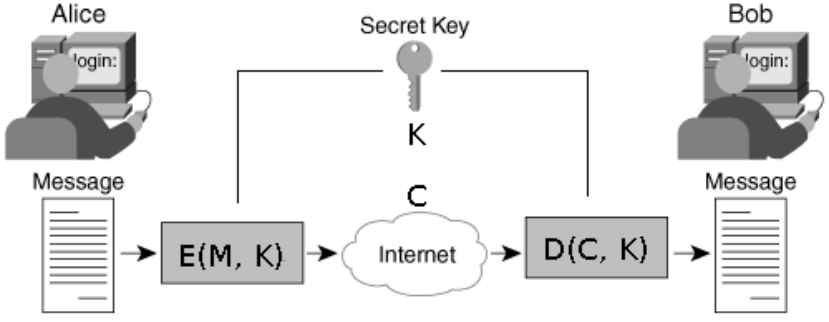
Tartalomjegyzék

Adatok elrejtése a kommunikáció során

- Adott két szovjet kém, akik távol élnek egymástól az USA-ban. Az egyikük Alice, a másikuk Bob.
  1. Alice egy fontos üzenetet szeretne elküldeni Bobnak
  2. Mivel szovjet kémek, nagyon nem lenne jó, ha bárki is hozzáférne rajtuk kívül a küldött üzenethez, ezért valamilyen olyan megoldásra lenne szükségük, amely segítségével csak Ők férnek hozzá az eredeti adathoz
- Kérdés: Hogyan tudndánk kivitelezni, hogy az adat nyílt csatornán kézbesítése során senki illetéktelen ne tudjon hozzáférni az üzenet valódi tartalmához?
- Válasz: az adatok látszólagos elrejtésével $\Longrightarrow$ ehhez van szükség a titkosításra
- Két fajtája van, amit a gyakorlatban is használnak
  1. szimmetrikus
  2. aszimetrikus

Szimmetrikus titkosítás

- P: az üzenet (plaintext)
- K: titkos kulcs (secret key)
- E(K, P): titkosító algoritmus (cipher)
- C: titkosított szöveg (ciphertext)
- D(K, C): visszafejtő algoritmus (decipher)



ábra: A szimmetrikus titkosítás működése kommunikáció során *(M=P)*

Definíció

Szimmetrikus titkosításról beszélünk, ha $E_K = D_K$

$$C = E(K, P)$$
$$P = D(K, C)$$

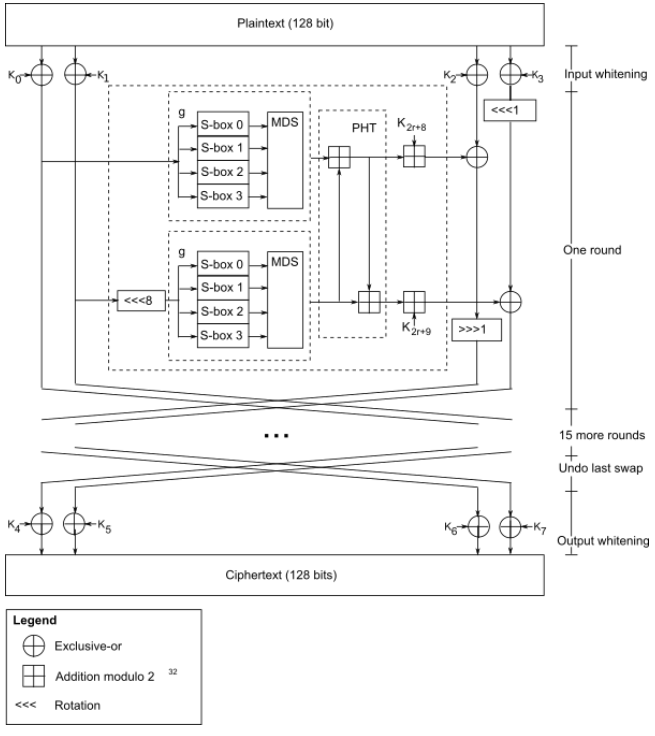
Szimmetrikus titkosítás

- E és D minden modern és biztonságos kriptográfiai rendszerben publikus, mindenki által ismert
- Bob számára az egyedüli szükséges információ tehát: K, ennek segítségével P bármikor előállítható
- Ha a használt algoritmus jó, akkor P nem nyerhető vissza[1], még akkor sem, ha K-n kívül minden adott (nyílván P nem).

[1] jelentős számítási teljesítmény befeketetése nélkül

Blokktitkosítók (block ciphers)

- Nagyon sok létezik belőlük, és NAGYON bonyolultak, pl. kép (Twofish)
- Első publikus: 1975, DES
- Példák blokktitkosító algoritmusokra:
  - a DES és a 3DES ('75, '95 egyik sem biztonságos)
  - az AES (jelenlegi ipari szabvány, ezzel működik a program, nagyon biztonságos)



Példa

1. Alice és Bob megegyezik a paraméterekben, és biztonságos csatornán egyeztetik az algoritmust, és a titkos kulcsot
   - Legyen $M :=$ "Hello Bob!", $K :=$"kortefa", $E :=$AES, így tehát $C :=$AES("Hello Bob!", "kortefa")
2. Alice elvégzi a titkosítást, ezzel megkapja a titkosított szöveget, $C :=$ ”$97272b719354b8857e4d070da16d22b2$” (hexadecimális)
3. C szabadon továbbítható bármilyen nem biztonságos csatornán

A Java lehetőségei

- A Java API mint minden máshoz, természetesen a titkosításhoz is nagyon sok segítséget nyújt.
- A `javax.crypto.*` csomag tartalmazza a legtöbb kriptográfiával kapcsolatos dolgot,
- köztük a `javax.crypto.Cipher` osztályt, amely a szimmetrikus titkosítás megvalósításának alapja.

Bele a közepébe

- A következő példa azt fogja megmutatni, hogy hogyan kell egy egyszerű fájl titkosító funkciót készíteni Javaban. (a megoldás nem biztonságos)
- Ehhez a `main()` metódust tartalmazó osztályunkban hozzunk létre egy statikus

```
public static void titkosit() {
   Scanner sc = new Scanner(System.in);
   String k = sc.nextLine();
   File f = new File(sc.nextLine());
   Titkositas t = new Titkositas(f,k);
}
```

A bekért adatokat később érdemes ellenőrizni (`f` létezik-e, `k` megfelelő hosszú, stb).

Titkositas osztály I.

Hozzuk létre a `Titkositas` osztályt![2]

```
public class Titkositas {


private Cipher c;
private File src;
private String k;

public Titkositas(File src, String k) {
   c = Cipher.getInstance("AES/ECB/PKCS5Padding");
   this.src = src;
   this.k = k;
}

public byte[] encrypt() {

}

}
```

[3] Az importálások helytakarékosság céljából hiányoznak.

Titkositas osztály II.

- Az előző kódrészletben elkészítettük a Titkositas osztály adattagjait, konstruktorát, és egy `byte[]` visszatérési típusú `encrypt()` nevű metódust.
- Ahhoz, hogy megérthessük, hogy mi történt a konstruktorban, az alábbi fogalmakkal tisztában kell lenni:
  - blokktitkosító algoritmus (*24. dia*)
  - mode of operation (*24. dia*)
  - padding (*25. dia*)
- A `Cipher` osztály `getInstance()` (*22. dia*) metódusát használtuk a példány lekéréséhez, egy string paraméterrel, amely a következőképpen nézett ki: `"AES/ECB/PKCS5Padding"`
  - AES: titkosító algoritmus (cipher)
  - ECB: mode of operation
  - PKCS5Padding: padding

Titkositas osztály III.

Az `encrypt()` metódus törzsét kell megírnunk ahhoz, hogy használható legyen az osztály.

```
1 public byte[] encrypt() {
2     byte[] filedata = Files.readAllBytes(this.f);
3     c.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(this.k.getBytes(StandardCharsets.UTF_8));
4     return c.doFinal(filedata);
5 }
```

1. metódus feje
2. a `Files` osztály segítségével beolvassuk az egész fájlt egy `byte[]` tömbbe
3. inicializáljuk a c objektumot (mód *[titk. vagy visszaf.]*, kulcs `SecretKeySpec` objektumba csomagolva)
4. visszaadjuk a titkosított bájtokat

A kész program

- Az alapokkal készen vagyunk, van egy osztályunk, amely képes egy fájl tartalmát titkosítani, és visszaadni `byte[]` formában.
- Innentől kezdve ahhoz kezdünk a titkosított bájt tömbbel, amihez csak akarunk.
  Egy tetszőleges eset:
- A `main()` metódust tartalmazó osztályunkból a `titkosit()` metódust kiegészítve fájlba írhatjuk a titkosított bájtokat:

```
1  public static void titkosit() {
...
5     Titkositas t = new Titkositas(f,k);
6     byte[] b = t.encrypt(); // f tartalmat titkositjuk, es b tombbe kitesszuk
7     Files.write(f.toPath(), b); // f-be kiirjuk a titkositott bajtokat
}
```

- Ezzel a fájl tartalmát felülírtuk a fájl titkosított bájtjaival.
  TODO: felhasználó tájékoztatása, értékek ellenörzése, kivételkezelés

A program felépítése

- A program, ahogy a neve is utal rá, arra való hogy fájlokat titkosítson
- A használt titkosítási eljárás biztonságos, feltörése min. 10^{18}-on évbe telne[3]
  - a használt algoritmus: AES, CBC móddal, 128 bites blokkmérettel
- A program 4+1 csomagból, és 6 osztályból áll.

```
.
|
+filetitok
    |
    +FileTitok.java
    +Constants.java
    |
    +crypto
      |
      +Cryptography.java
    +gui
      |
      +Window.java
    +io
      |
      +FileIO.java
    +misc
      |
      +Util.java
```

[3] Megfelelő bonyolultságú jelszót és megfelelő key derivation funkciót használva a titkosításhoz, illetve egy 2010 körüli szuperszámítógépet használva a feltöréshez

Tervezési alapelvek

- A program tervezésénél igyekeztem az objektum-orientált paradigmát használni.
- Ezért az osztályok egyértelműen, funkció alapján vannak csomagokra bontva.
- A feladatnak megfelelően, a program grafikus felülettel készült. Ehhez a Swing nevű GUI-kezelő csomagot (`javax.swing.*`) használtam.
- A tervezéshez nem nagyon használtam semmilyen különleges dolgot, főleg fejben terveztem.
- Nagy segítség volt a stackexchange fórumok közül egy jó pár, ahol a profi programozók személyreszabottan véleményezték a feltöltött kódot, amely alapján el tudtam végezni a hiányosságok javítását.

Osztályok feladatainak rövid összefoglalása

Az (f) jelzés azt jelzi, hogy a komponens a frontend munkájában vesz rész, a (b) pedig azt, hogy a backendében.

(f) `filetitok.FileTitok` általános, `main()` metódust tartalmazó osztály

(f) `filetitok.Constants` gyakran használt szöveg-, és számkonstansokat tartalmazó osztály

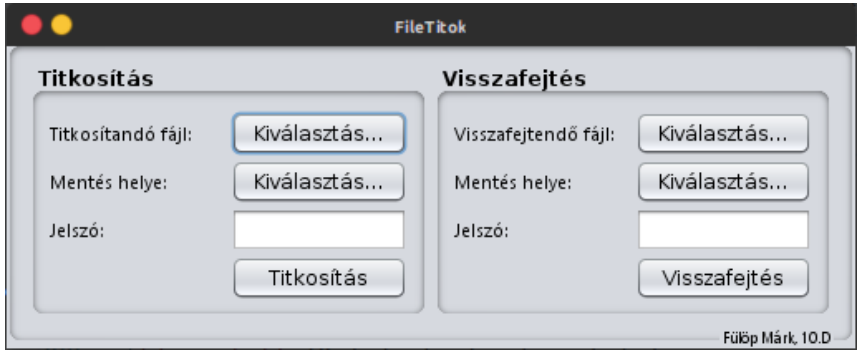
(f) `filetitok.misc.Util` gyakran végzett, de különböző funkciók

(f) `filetitok.gui.Window` a GUI-ért felelős osztály, továbbá végez alapvető ellenőrzéseket, mielőtt átkerülne a vezérlés a program backend részéhez

(b) `filetitok.io.FileIO` fájlok I/O műveleteit vezérli, az adatokat és az utasításokat a `Window` osztálytól kapja, a `Cryptography` osztállyal kommunikál

(b) `filetitok.crypto.Cryptography` a program lelke, mindenféle kriptográfiai műveletet (titkosítás és visszafejtés) végez

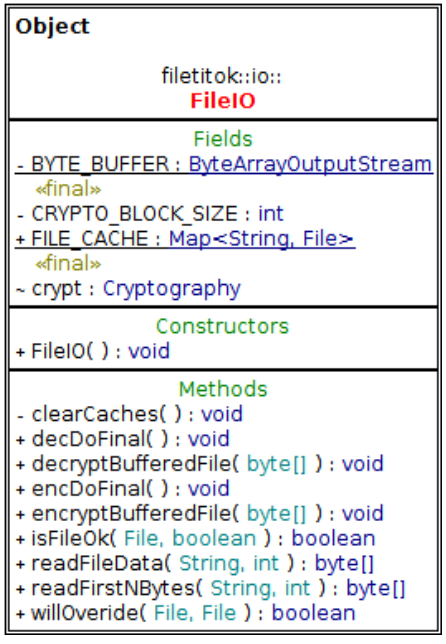
Felület felépítése

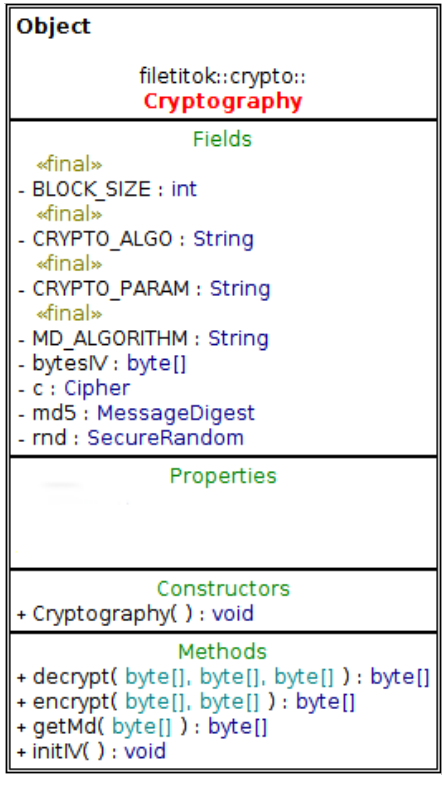


- Egyszerű felépítésű
- A `Nimbus` nevű look and feelt használja
- Két panel: Titkosítás és Visszafejtés
- Fájlválasztó gombok mindkét panelen (forrásfájl, és mentés helye)
- Mindkét panelen szövegmező (a titkosításhoz használt jelszó) + egy OK gomb

Két fő backend osztály

A program lelke a `FileIO` és a `Cryptography` osztály.





Az adattagok és a funkciók leírása a következő diákon.

FileIO osztály

adattagok

`BYTE_BUFFER`	dinamikusan növekvő nagyságú `ByteArrayOutputStream`, egy adott titkosítási-visszafejtési folyamat során az éppen szükséges bájtokat tároljuk benne, majd általában kiírjuk a fájlba, és ürítjük
`FILE_CACHE`	szintén dinamikus növekvő, kulcs-érték típusú adattároló (`String-File`). szintén a folyamat során használt fájlok mutatóit pakoljuk bele. minden betett fájl kap egy kulcsot, amivel később lehet rá hivatkozni, ha szeretnénk vele csinálni különböző dolgokat
`crypt`	a `Cryptography` osztály egy példánya. gyakorlatilag ez a példány végzi az összes titkosítási műveletet
`CRYPT_BLOCK_SIZE`	a fájlbeolvasásoknál szükséges bájt tömbök allokációjához szükséges tudni, hogy a használt blokktitkosító mekkora blokkokkal dolgozik (AES = 16 bájt)

metódusok

`isFileOk(File file, boolean writeAccess)`	ellenőrzi, hogy a fájl létezik, és olvasható, ha `writeAccess = true`, akkor azt is, hogy írható-e
`encrypt/decryptBufferedFile(byte[] pw)`	a `FILE_CACHE` tárolóban lévő egyik fájl bájtjait titkosítja/visszafejti, a `pw` MD5 hashjével, és a `BYTE_BUFFER`-be írja az eredményt
`readFileData(String fileKey, int skip)`	a `FILE_CACHE` `fileKey` kulccsú fájljának tartalmát olvassa be egy byte tömbbe úgy, hogy az első `skip` bájtot átugorja
`enc/decDoFinal()`	elvégzi a `BYTE_BUFFER`-ben lévő aktuális adat fájlba írását, és meghívja a `clearCaches()` metódust
`willOveride(File file1, File file2)`	`true`-t ad vissza, ha `file1` felülírná `file2`-t, különben `false`-t
`readBlock(String fileKey, int blockSize)`	beolvas egy `blockSize` bájt hosszú blokkot a `FILE_CACHE` `fileKey` kulcsú fájljából
`clearCaches()`	kiüríti a `BYTE_BUFFER`-t, és a `FILE_CACHE`-t

Cryptography osztály

adattagok

`c`	`Cipher` objektum, "titkosító" (23)
`md5`	`MessageDigest` objektum, hash funkció (23)
`rnd`	biztonságos random generátor (23)
`CRYPTO_ALGO, CRYPTO_PARAM, MD_ALGORITHM`	titkosításhoz és hash számításhoz szükséges algoritmus konstansok
`BLOCK_SIZE`	megegyezik a `FileIO` osztályban lévő ilyen adattaggal, a blokkméretet tárolja (25)
`bytesIV`	az aktuális inicializációs vektort tárolja (24)

metódusok

`initIV()`	új random értéket ad a `bytesIV` adattagnak
`encrypt(byte[], byte[])`	a paraméterként kapott első tömböt titkosítja a `c` `Cipher` objektum segítségével, és a második tömb kulccsal, majd visszaadja a titkosított bájtokat
`decrypt(byte[], byte[], byte[] )`	az első paraméteren végez visszafejtést a második paraméter (kulcs), és a harmadik paraméter segítségével (IV)
`getMd(byte[])`	a kapott paraméter `MD_ALGORITH` típusú hash-jét számítja ki, és adja vissza

Egy tetszőleges fájl titkosításának rövid leírása

1. A felhasználó kiválasztja a titkosítandó fájlt (e_src_file), a mentés helyét (e_dir), és a jelszót (pw).
2. A két File objektumot rögzítjuk a `FILE_CACHE` tárolóban.
3. A `FileIO` osztály `encryptBufferedFile(byte[])` metódusa megkapja paraméterként a byte tömbbé alakított jelszót, ebből MD5 hash-t (25) képez, majd kiírja a `BYTE_BUFFER`-be a fájl titkosított bájtjait.
4. Ezután az `encDoFinal()` metódus hívódik meg, amely kiíírja a `BYTE_BUFFER`-ből a titkosított bájtokat a megadott könyvtár (`e_dir`) `e_src_file` nevű fájljába, majd kitakarítja a `BYTE_BUFFER`-t és a `FILE_CACHE`-t.

OOP Fogalmak I.

Singleton osztály

Olyan osztály, amely mindösszesen egy példánya létezhet egy időben. Tehát a program futása során kettő darab különböző `Kutya` példány nem létezhet, ha a `Kutya` singleton osztály.

`getInstance()` metódus

A `getInstance()` metódust singleton osztályok esetében használjuk. Megvizsgálja, hogy létezik-e már példány az osztályból, ha igen, annak a mutatóját adja vissza, ha nem akkor pedig készít egy új példányt, és annak a mutatóját adja vissza.

Egy osztályt úgy tehetünk singletonná, ha konstruktorát privát hozzáférési szintre állítjuk, és készítünk hozzá egy `getInstance()` metódust.

Java osztályok rövid leírásai

Az osztályok neveire kattintva megnyílik az osztály javadoc oldala. A bővebb definícióért és értelmezésért lsd. ezeket.

Cipher | javax.crypto

titkosítás alapjait képező osztály

SecureRandom | java.security

biztonságos véletlenszám generátor (értékei nem határozhatók meg előre)

MessageDigest | java.security

hash funkciókat megvalósító singleton osztály

Kripto fogalmak I.

A definíciók le vannak egyszerűsítve a könnyebb értelmezés miatt.

blokktitkosító (block cipher)

Olyan titkosító algoritmus, amely egyszerre csak egy, fix hosszúságú bit tömbön végez műveletet. pl.: **AES**, 3DES, Twofish

működés módja (mode of operation)

Az a pontos eljárás, amely leírja, hogy a blokktitkosítónak milyen folyamat alapján kell titkosítani és visszafejtenie. pl.: ECB, **CBC**

inicializációs vektor, IV (initialization vector)

Ha egy blokktitkosító nem ECB módban működik, szüksége van egy IV-re, amellyel az első blokkon hajt végre $\oplus$ (xor) műveletet. `SecureRandom`-ból generáljuk. Olyan hosszú, mint a titkosító blokkmérete.

Kripto fogalmak II.

kitöltés (padding)

A blokktitkosítók csak pontosan `n` bit hosszú tömbön tudnak műveletet végezni. Ha az utolsó tömb nem pontosan `n` hosszú, kiegészítjük semleges bitekkel.pl.: **PKCS7**, PKCS5

blokkméret (block size)

A blokktitkosítók csak pontosan n bit hosszú tömbön tudnak műveletet végezni. Az n a blokkméret. pl. AES esetében, $n = 128$

hash

A hash függvény egy olyan eljárás, amellyel bármilyen hosszúságú adatot adott hosszra tudunk leképezni. Két különböző szónak nem lehet ugyan az a hash értéke. pl: "alma" -> "ebbc3c26a34b609..." "almb" -> "73c4b336025..." (MD5-nél 16 bájt)

Képek forrásai

1. (4. dia) etutorials.org, link
2. (6. dia) By Ssims - Own work, Public Domain, link