Seguretat Computacional. Curs 2004-2005

Pràctica 3: *Criptografia Simètrica*.

Objectiu

L'objectiu d'aquesta pràctica és treballar amb les eines de la API criptogràfica de Java que permeten xifrar dades. Concretament, ens centrarem en l'ús d'algorismes de criptografia simètrica i la seva combinació amb algorismes asimètrics per crear sobres digitals.

Criptografia Simètrica

Generació i ús de Claus

Les claus de criptografia simètrica venen representades per la interfície *SecretKey*. Igual que *PublicKey* i *PrivateKey* aquesta interfície també deriva de la interfície general *Key*.

Per crear claus simètriques disposem, igual que en criptografia asimètrica, de dues maneres diferents segons vulguem generar claus de forma aleatòria o a partir d'un material previ:

- *KeyGenerator* Aquesta classe permet generar claus de forma aleatòria per a diferents algorismes de criptografia simètrica com DES, Triple DES o AES.
- *SecretKeyFactory* Aquesta classe permet generar a claus a partir de cert material previ. Aquest material (pot ser una colla de bits), ha d'anar encapsulat en una classe del tipus *KeySpec*. Si l'algorisme utilitzat per generar les claus és DES o TripleDES, podem utilitzar les classes *DESKeySpec* o *DESedeKeySpec* respectivament. En el cas de que l'algorisme utilitzat sigui un altre, podem utilitzar la classe *SecretKeySpec* que és de propòsit general per a claus de qualsevol algorisme simètric. Aquesta última classe, que també implementa la interfície *SecretKey*, pot ser utilitzada directament com una clau sense necessitat d'utilitzar una *KeyFactory* prèviament.

Xifrat de dades

Així com a la pràctica anterior disposàvem de la classe *Signature* per al signat de dades, la classe *Cipher* ens permet xifrar i desxifrar dades usant algorismes

simètrics o asimètrics. Aquesta classe és un motor criptogràfic com els que hem utilitzat fins ara, pel que la inicialització sempre es durà a terme mitjançant el típic mètode *.getInstance("Algorisme")*. Alguns dels mètodes més importants d'aquesta classe són els següents:

- *getInstance(String algorithm)* Com ja hem esmentat, aquest mètode instància un xifrador / desxifrador amb un algorisme concret. Per veure els algorismes possibles per a xifrar consulteu [2].
- *void init(int opmode, Key key)* Inicialitza el xifrador. Amb una clau per xifrar o desxifrar. El paràmetre opmode pot valer *Cipher.ENCRYPT_MODE* o *Cipher.DECRYPT_MODE*.
- *void init(int opmode, Key key, AlgorithmParameters param* Afegeix la possibilitat de passar paràmetres que un algorisme de xifrat concret pugui necessitar a l'hora de xifrar o desxifrar.
- *byte[] update(byte[] b)* Xifra un bloc de dades *b* i retorna el corresponent bloc xifrat.
- *byte[] doFinal(byte[] b)* Xifra l'últim bloc corresponent a les dades que vulguem xifrar.

Xifrat i desxifrat usant Streams

Quan volem xifrar qualsevol dada utilitzant el motor criptogràfic Cipher, ens trobem que hem de utilitzar bucles que vagin cridant successives vegades el mètode *update()* i finalment *doFinal()*. Java proporciona dues classes Stream que permeten automatitzar molt més aquest procés:

- CipherInputStream
- CipherOutputStream

Aquestes classes, s'inicialitzen amb un objecte del tipus Cipher ja creat i un stream d'entrada o sortida, depenent de quina de les dues classes utilitzem.

PBE: Password-Based Encryption

La *Password-Based Encryption* ens permet generar claus d'un algorisme simètric a partir d'un password de forma segura. La forma de dur a terme aquest procés està definida a l'estàndard PKCS#5 [4] de RSA Laboratories. El principal avantatge d'utilitzar aquest tipus de claus, és que no cal emmagatzemar la clau usada a

cap lloc ja que simplement recordant el password a partir de la qual la hem creat podem obtenir-la.

La diferència principal a l'hora de utilitzar aquest tipus de xifrat respecte el vist anteriorment són fonamentalment dues.

En primer lloc, el procés de creació de claus es du a terme mitjançant una *SecretKeyFactory* ja que estem creant una clau a partir de material previ (un password). L'algorisme amb el qual instanciarem aquesta *SecretKeyFactory* serà un algorisme propi de PBE, com per exemple *PBEWithMD5AndTripleDES*. Per veure la llista de possibles algorismes a utilitzar consulteu [2]. Una vegada instanciada la *SecretKeyFactory* construirem la clau passant-li un objecte de la classe *PBEKeySpec* inicialitzat amb el password que haguem triat.

En segon lloc, a la hora de xifrar utilitzant un motor criptogràfic *Cipher* hem de tenir en compte que els xifrats usant PBE necessiten ser inicialitzats amb més paràmetres a part de la clau i el mode d'operació. Aquests paràmetres són la *salt* i el nombre de *rounds* utilitzats per generar la clau a partir del password. Aquests paràmetres els especificarem al inicialitzar el motor *Cipher* per mitjà de la classe *PBEParameterSpec*.

- PBEKeySpec(char[] password)
- PBEParameterSpec(byte[] salt, int rounds)

Enunciat

El treball a realitzar en aquesta pràctica es divideix en dues parts: La creació d'un programa per xifrar arxius amb un algorisme simètric, i un segon programa que xifri dades utilitzant la criptografia simètrica utilitzant sobres digitals.

Xifrat simètric

Aquesta part de la pràctica consisteix en omplir les funcions buides proporcionades en la classe *Simetric*.

- *SecretKey generaClau()* Genera una clau i l'emmagatzema en la variable de la classe interna anomenada *_sk*.
- *void guardaClau(String fname)* Emmagatzema la clau generada a disc en el fitxer de nom especificat per paràmetres.
- *SecretKey llegeixClau(String fname)* Llegeix una clau emmagatzemada a disc i la guarda en la variable interna *_sk*.

- *void xifra(String forigen, String fdesti)* Xifra el fitxer *forigen* usant la clau *_sk* i emmagatzema el contingut xifrat en un fitxer de nom *fdesti*.
- *void desxifra(String forigen, String fdesti)* Desxifra el fitxer *forigen* usant la clau *_sk* i emmagatzema el contingut desxifrat en un fitxer de nom *fdesti*.

Part Opcional: Xifrat de claus

En aquesta part consisteix de les següents funcions:

- *void guardaClauPBE(String fname, String password)*
- *SecretKey llegeixClauPBE(String fname, String password)*

Aquestes funcions tenen la mateixa finalitat que les funcions *guardaClau()* i *llegeixClau()* de l'apartat anterior, amb la diferència que emmagatzemen la clau a disc de forma xifrada. El xifrat de la clau es durà a terme mitjançant un algorisme de PBE.

La salt utilitzada en el xifrat amb PBE ha de ser un array de bytes aleatori de longitud 8. Per crear-lo podem utilitzar la classe *Random*. Com que el valor de sal utilitzat en el xifrat, es necessita en el procés de desxifrat, haurem de concatenar aquest valor a l'arxiu en disc, per tal de que pugui ser recuperat en el procés de desxifrat de la clau.

Sobres Digitals

A l'hora de xifrar grans quantitats de dades la criptografia asimètrica sol ser una eina molt ineficient. És per això, que quan volem els avantatges de la criptografia de clau pública i l'eficiència de la criptografia simètrica usem sobres digitals.

Per xifrar dades utilitzant un sobre digital es duen a terme els següents passos:

1. Creació d'una clau aleatòria *k* per un algorisme simètric, per exemple, Triple DES o AES. Anomenarem aquesta clau, clau de sessió.
2. Xifrat de les dades M usant la clau de sessió, obtenint $E_k(M)$
3. Xifrat de la clau de sessió *k* utilitzant la clau pública del receptor *p*, obtenint $E_p(k)$.
4. Enviar la concatenació de $E_p(k)$ i $E_k(M)$ al receptor del missatge.

En aquesta part de la pràctica crearem un programa que xifri un arxiu mitjançant un sobre digital. Per això utilitzarem l'esquelet buit anomenat Sobre.java. Aquest programa rebrà per paràmetres tres noms d'arxiu. A continuació els anomenem per el mateix ordre en el que s'especificaran per línia de comandes:

- *arxiu a xifrar* Serà el primer paràmetre que passarem al programa, i serà el nom de qualsevol arxiu que vulguem xifrar usant un sobre digital.
- *arxiu xifrat* Nom de l'arxiu que contindrà el sobre digital sobre les dades extretes del fitxer anterior.
- *arxiu desxifrat* Nom de l'arxiu en el qual es bolcarà el resultat de desxifrar el contingut del sobre digital.

Per a crear el sobre, el programa durà a terme els següents passos:

1. Crear un parell de claus RSA
2. Crear una clau de sessió Triple DES (DESede)
3. Xifrar la clau de sessió amb la clau pública RSA i emmagatzemar-la a les primeres posicions del fitxer de sortida.
4. Xifrar el contingut del fitxer d'entrada usant la clau de sessió i emmagatzemar el resultat al fitxer de sortida.
5. Obrir el fitxer de sortida
6. Llegir la clau de sessió i desxifrar-la usant la clau privada RSA
7. Usar la clau de sessió per desxifrar el contingut restant
8. Emmagatzemar el contingut desxifrat al tercer dels fitxers especificats per paràmetres.

Serialització d'Objectes

La serialització d'objectes s'utilitza, per exemple, per emmagatzemar objectes a disc. En aquesta pràctica utilitzarem aquesta capacitat per emmagatzemar la clau de sessió xifrada dins l'arxiu que conté el sobre digital.

Els objectes, es graben a disc utilitzant dues classes stream:

- ObjectInputStream
- ObjectOutputStream

Quan tinguem la clau de sessió xifrada, la emmagatzemarem dins d'un objecte de la classe *DadesSobre*. Un cop creat aquest objecte utilitzarem la serialització per grabar-lo en l'arxiu que conté el sobre digital.

Part Opcional: Sobres Digitals Signats

La part opcional corresponent a aquesta part consisteix en crear un sobre digital xifrat i signat. Si observeu la classe *DadesSobre* hi tenim un paràmetre per emmagatzemar la signatura feta sobre l'arxiu original. Els passos que haurà de dur a terme el programa si la part opcional està implementada seran els següents:

1. Crear un parell de claus RSA corresponents al creador del sobre
2. Crear un parell de claus RSA corresponents al receptor del sobre
3. Usar la clau privada del creador per calcular la signatura digital sobre l'arxiu origen.
4. Crear una clau de sessió Triple DES (DESede)
5. Xifrar la clau de sessió amb la clau pública del receptor.
6. Emmagatzemar la clau xifrada i la signatura digital en les primeres posicions del fitxer de sortida.
7. Xifrar el contingut del fitxer d'entrada usant la clau de sessió i emmagatzemar el resultat al fitxer de sortida.
8. Obrir el fitxer de sortida
9. Llegir la clau de sessió i desxifrar-la usant la clau privada RSA del receptor
10. Usar la clau de sessió per desxifrar el contingut restant
11. Calcular la signatura digital sobre el contingut desxifrat usant la clau pública del creador.
12. Emmagatzemar el contingut desxifrat al tercer dels fitxers especificats per paràmetres.
13. Treure un missatge per pantalla que digui si la verificació de la signatura és positiva o no.

Referències

[1] Java API. http://java.sun.com/j2se/1.5.0/docs/api/

[2] Java Cryptography Architecture. http://java.sun.com/j2se/1.5.0/docs/guide/s

[3] Programació en java. http://java.programacion.net

[4] Estàndards PKCS. http://www.rsasecurity.com/rsalabs/pkcs/index.html