technische universität
dortmund

Sicherheit: Fragen und Lösungsansätze Übung 5

Thomas Ruhroth | Dortmund
13.Nov 2012

Übung 5

- ~~Hinweise~~
- Evaluation
- Gruppenübung
- Besprechung der Abgaben

- Ziele:
  - Wissen Kryptographie
  - Verstehen der Evaluation (mein Ziel :-)

Hinweise zu Ihren Abgaben

- Fragen: Gerne
  - Bitte ins InPUD

- **Wer in den Übungskasten abgegeben hat, bekommt die Aufgaben nächstes mal zurück.....**

Hinweise zu Ihren Abgaben

- ✓ Richtig (kleinere Probleme werden ignoriert)
- ~ Überwiegend Richtig
  - Ungenauigkeit
  - Unrichtigkeiten
  - Achten sie auf Unterstreichungen
- ¬ K „nicht knakig“
- NL nicht lesbar
- LB:
  - 10 … 0 (lesbar bis unleserlich)
  - Schlechter als 5: Ich musste Raten! Gefahr von Punktverlust!

Test

- Zettel raus :-)
- 9 Min!

Test

- Was ist der Unterschied zwischen public-key und secret-key Verschlüsselung? (1p)
- Was sind die Eigenschaften eines One-Way-Hashs? (3p)
- Was sind die Vor- und Nachteile eines One-Time-Key-Verfahrens? (3p)
- Wie funktioniert die Entschlüsselung eines One-Time-Key-Verfahrens? (2p)

Zettel tauschen

- Prüfen Sie:
  - Ist die Schrift gut lesbar?
  - Verstehen Sie die Antwort

- Inhalt erarbeiten wir auf den nächsten Folien, bewerten sie dabei die Aufgabe.

Public/Private-Key

- ***symmetric* (or *secret-key*) mechanism:**
  - the test key is (basically) *equal* to the authentication key

- ***asymmetric* (or *public-key*) mechanism:**
  - the test key is essentially *different* from the authentication key
  - an additional *secrecy property* (*naive* version) is required:

    the (private) authentication key ak_S

    cannot be “determined” from the (public) test key tk_S

One-Way-Hash

- some item of interest is often represented in a concise, disguised and unforgeable form, called a *fingerprint*, a *digest* or a *hash value*
- *concise*:
  - representation consists of a suitably short bit string of an agreed format
  - a large domain of items is mapped onto a small domain of representations: there must be collisions
- *disguised*:
  a represented item cannot be “determined” from its representation
- *unforgeable*:
  nobody can “determine” a representation of an item without a knowledge of that item
- *collision resistant*:
  nobody can “determine” pairs of items that share a representation

One-Time-Key

- restriction to using a key *only once* is crucial:
  observing a ciphertext/plaintext pair, an attacker achieves complete success:
  solve, for each position *i*, the equation $y_i = k_i \oplus x_i$
  regarding the secret key bit as
  $k_i = y_i \oplus x_i$
- considering the transmission of a *single* message:
  qualified to the best possible extent regarding *secrecy* and *efficiency*
- as a trade-off for the best secrecy - proved to be inevitable:
  - secret cipher key can be used only once
  - secret cipher key must be as long as the anticipated plaintext
- as a *stand-alone* mechanism,
  pure one-time key encryption is practically employed
  only in dedicated applications with extremely high secrecy requirements
- however, basic approach is widely exploited in
  - variants
  - subparts of other mechanisms

Entschlüsselung

- ***key generation*** algorithm *Gen(erate_Cipher_Key)*

selects a "truly random" *cipher key* (k_1 ,… , k_n)

- ***encryption*** algorithm *Enc*

handles the plaintext (x_1 ,… , x_n) and the cipher key (k_1 ,… , k_n) as streams;

treats each corresponding pair of a plaintext bit x_i and a cipher key bit k_i

as input for a XOR operation, yielding a ciphertext bit

$y_i = k_i \oplus x_i$

- ***decryption*** algorithm *Dec*

handles the ciphertext (y_1 ,… , y_n) and the cipher key (k_1 ,… , k_n) as streams;

treats each corresponding pair of a ciphertext bit y_i and a cipher key bit k_i

as input for a XOR operation,

yielding the original plaintext bit x_i *correctly*:

$k_i \oplus y_i = k_i \oplus (k_i \oplus x_i) = (k_i \oplus k_i) \oplus x_i = 0 \oplus x_i = x_i$

HA 1.1 (eine abgegebene Lösung)

```
public static String hash(String in){
            int[] strhash;
            strhash = new int[26];
            int asciiCode;
            String out;
            out = "";

            String str = in;
            str = str.toLowerCase();

            for(int i = 0; i<26; i++){
                  strhash[i] = 0;}

            for(int i = 0; i < str.length(); i++) {
            asciiCode = str.charAt(i);
            if (asciiCode > 96 && asciiCode <123 ){ strhash[asciiCode-97]++; };
            }

            for(int i = 0; i<26; i++){
                  out = out + strhash[i];}
            return out;
      }
```

HA 1.2

Die Hash-Funktion ist eine One-Way-Hash-Funktion:

- some item of interest is often represented in a concise, disguised and unforgeable form, called a *fingerprint*, a *digest* or a *hash value*
- *concise*:
  - representation consists of a suitably short bit string of an agreed format
  - a large domain of items is mapped onto a small domain of representations: there must be collisions
- *disguised*:
  a represented item cannot be “determined” from its representation
- *unforgeable*:
  nobody can “determine” a representation of an item without a knowledge of that item
- *collision resistant*:
  nobody can “determine” pairs of items that share a representation

HA 1.2

Domäne	kollisionsfrei
Strings	nein
Deutsche Texte	nein
Sonette	ja

```
public final class SimpleHash extends java.security.MessageDigestSpi {
        private byte[] ba = "".getBytes();

        @Override
        protected byte[] engineDigest() {
                try {
                        byte[] rba = { ba[0], ba[ba.length - 1] };
                        return rba;
                } catch (Exception e) {
                        return null;
                }
        }

        @Override
        protected void engineReset() {
                ba = "".getBytes();
        }

        ….
}
```

```
public final class SimpleHash extends java.security.MessageDigestSpi {

                    ...

        @Override
        protected void engineUpdate(byte input) {
                byte[] ba2 = new byte[ba.length + 1];
                System.arraycopy(ba, 0, ba2, 0, ba.length);
                ba2[ba2.length - 1] = input;
                ba = ba2;
        }

        @Override
        protected void engineUpdate(byte[] input, int offset, int len) {
                for (int i = 0; i < len; i++)
                        engineUpdate(input[i + offset]);
        }

        public SimpleHash() {
                engineReset();
        }
}
```

```
import java.security.Provider;

public final class SimpleHashProvider extends Provider {

   public SimpleHashProvider() {
       super("SimpleHashProvider", 1.0, "SFuL Uebung 7 Simple Hash Provider");
       put("MessageDigest.SimpleHash", "SimpleHash");
     }

     private static final long serialVersionUID = 1L;

}
```

```
public class SimpleHashProviderTest {
        public static void main(String[] args) {
        try {
                Provider shp = new SimpleHashProvider();
                Security.addProvider(shp);

                System.out.println("Name: " + shp.getName() + ", Version: "
                                + shp.getVersion() + ", Info: " + shp.getInfo());
                MessageDigest md = MessageDigest.getInstance("SimpleHash");
                System.out.println(md.getAlgorithm());

                md.update("foo".getBytes());
                System.out.println("Hashwert foo: " + new String(md.digest()));

                md.update("bar".getBytes());
                System.out.println("Hashwert foobar: " + new String(md.digest()));

                md.reset();
                md.update("hello world".getBytes(), 2, 7);
                System.out.println("Hashwert hello world[2-8]: "
                                + new String(md.digest()));

        } catch (Exception e) {
                System.out.println(e);
        }
    }
}
```