Turvalisus

Autentimine, krüptograafia, võtmed, õigused

Signeerimine

Lugedes teksti või käivitades programmi soovime mõnikord kindlad olla, et kellegi kuri käsi ega lihtsalt tehniline viperus pole algseid andmeid muutnud, et võime uskuda oma kettal olevaid andmeid samuti nagu usume oma käe ja tindiga telefoniraamatusse kirjutatud numbreid. Samuti soovime mõnikord kontrollida, et saadetis tuli ikka sellelt inimeselt, kust me arvasime, et see peaks tulema. Relvastatud valve ning mitmekordsete topelteksemplaride kõrval on üheks lahenduseks avaliku ja salajase võtme võtme krüptograafia, kus materjalide looja lisab andmete ning oma salajase võtme abil loodud digitaalallkirja. Hiljem saab autori avaliku võtme ning digitaalallkirja järgi kontrollida, kas andmed on muutumatul kujul säilinud. Nii võib kontrollida andmete terviklust oma kettal, sama moodusega võib ka kaugel asuv adressaat enamjaolt kindel olla, et kohale jõudnud saadetis ikka sellelt inimeselt pärit on, kellelt seda arvatakse olevat. Põhiehitus on pea kõigil sellistel kontrollivahenditel sarnane, erinevused on peamiselt krüptoalgoritmide ning pealisehituse osas. Levinumaid algoritme on kümmekond, kasutajaliideseid aga vähemalt nii palju kui ennast tähtsaks pidavaid tarkvaratootjaid ning eks aeg näitab, palju neid aja jooksul juurde tuleb või edasi areneb.

Java standardkomplekti kuuluvad keytool ning jarsigner. Esimese abil õnnestub võtmeid ja sertifikaate luua ning talletada, teise abil võib jar-arhiivi talletatud andmeid salajase võtmega signeerida ning avaliku võtmega kontrollida, kas arhiiv arvatud kohast tervena pärale jõudnud on.

Et õnnestuks andmeid turvama hakata, selleks peab omale võtmepaari looma. Keytoolile tuleb öelda, et soovime võtit luua (-genkey), määrata kuhu andmed panna (-keystore jaagup.store, muul juhul pandaks andmed faili .keystore) ning nimi mille juurde kuuluvaks võtmed luua. Tulemusena küsitakse veidi isikuandmeid ning loodud võtmed koos nendega väljastatakse sertifikaadina andmehoidlasse.

```
C:\User\jaagup\0104\turva\sert>keytool -genkey -keystore jaagup.store -alias jaagup
Enter keystore password:  123456
What is your first and last name?
  [Unknown]:  Jaagup Kippar
What is the name of your organizational unit?
  [Unknown]:  Informaatika oppetool
What is the name of your organization?
  [Unknown]:  TPU
What is the name of your City or Locality?
  [Unknown]:  Tallinn
What is the name of your State or Province?
  [Unknown]:  Harjumaa
What is the two-letter country code for this unit?
  [Unknown]:  EE
Is <CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE>
correct?
   [no]:  Y

Enter key password for <jaagup>
        (RETURN if same as keystore password):  123456
```

Kui soovin näha, kelle andmed võtmehoidlas on, siis piisab keytoolile anda käsklus list. Võtmehoidla teenusepakkujaks on SUN, Java looja ning hoidla tüübiks jks. Neid andmeid läheb vaja, kui hiljem soovida oma programmis hoidlast sertifikaate ning

võtmeid lugeda. Sees on vaid üks sertifikaat, jaagupi-nimelisele isikule. Selge, sest sinna baasi pole rohkem kirjeid loodud. Kui teha või lugeda sertifikaate juurde, siis tuleb ka väljastatavaid ridu rohkem.

```
C:\User\jaagup\0104\turva\sert>keytool -list -keystore jaagup.store
Enter keystore password:  123456

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:

jaagup, Tue Apr 03 20:43:04 GMT+03:00 2001, keyEntry,
Certificate fingerprint (MD5): 0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
```

Soovides oma sertifikaati lähemalt uurida või sõbrale edasi anda, et too saaks hiljem kontrollida kas saadetis ikka minult on, tuleb hoidlast vastav sertifikaat eraldi faili kirjutada. Faili nimeks saab jaagup.cert.

```
C:\User\jaagup\0104\turva\sert>keytool -export -keystore jaagup.store -alias ja
gup -file jaagup.cert
Enter keystore password:  123456
Certificate stored in file <jaagup.cert>
```

Kui vaadata kataloogi, siis seal paistab kaks faili. .store-laiendiga sertifikaadihoidla ning .cert laiendiga üksik sertifikaat.

C:\User\jaagup\0104\turva\sert>dir

```
.              <DIR>        03.04.01  20:39 .
..             <DIR>        03.04.01  20:39 ..
JAAGUP~1 STO         1 278  03.04.01  20:43 jaagup.store
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
```

Soovides sertifikaadi sisuga lähemalt tutvuda, aitab võti -printcert .

```
C:\User\jaagup\0104\turva\sert>keytool -printcert -file jaagup.cert
Owner: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Issuer: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa,
C=EE
Serial number: 3aca0b9a
Valid from: Tue Apr 03 20:42:50 GMT+03:00 2001 until: Mon Jul 02 20:42:50 GMT+03:00
2001
Certificate fingerprints:
         MD5:  0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
         SHA1: AA:2B:73:28:A1:F9:B4:8F:71:D8:D8:C7:66:CC:37:14:36:8C:8F:EE
```

Kui tuttav (kelle kataloogiks on sert2) soovib edaspidi minult saadud teadete autentsust kontrollima hakata, siis ta muretseb enesele mu sertifikaadi. Kas lihtsalt kopeerib selle mu kataloogist või tähtsamal juhul saame pidulikult linnas kokku, kus talle disketil oma avalikku võtit sisaldava sertifikaadi ulatan.

```
C:\User\jaagup\0104\turva\sert2>copy ..\sert\jaagup.cert .
        1 file(s) copied
```

Saadud sertifikaati oma hoidlasse tõmmates peab Juku nüüd tõesti kindel olema, et tegemist oli kindlasti minu käest saadud kettaga ja keegi pole tema taskus vahepeal andmeid muutnud. Muul juhul võib see keegi tundmatu kergesti minu nime

all esinema hakata ning vale tuleb välja alles siis, kui ise midagi Jukule saadan ning selle signatuur valeks loetakse.

```
C:\User\jaagup\0104\turva\sert2>keytool -import -keystore juku.store -alias jaagup
-file jaagup.cert
Enter keystore password:  123456
Owner: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Issuer: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa,
C=EE
Serial number: 3aca0b9a
Valid from: Tue Apr 03 20:42:50 GMT+03:00 2001 until: Mon Jul 02 20:42:50 GMT+03:00
2001
Certificate fingerprints:
         MD5:  0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
         SHA1: AA:2B:73:28:A1:F9:B4:8F:71:D8:D8:C7:66:CC:37:14:36:8C:8F:EE
Trust this certificate? [no]:  y
Certificate was added to keystore
```

Igaks juhuks veel küsiti üle, et kas võtmehoidla omanik sellise sertifikaadi lisamisega nõus on. Kahtluste hajutamiseks võib üle kontrollida sertifikaadiga kaasnevad sõnumilühendid. Kui need on samasugused kui minu antud paberitükil, siis pole suurt põhjust enam kedagi sertifikaatide salajases vahetamises süüdistada, sest vähemalt lähiajal pole karta, et keegi suudaks ise samasuguse sõnumilühendiga andmeid luua.

Jukul niisiis praegu kataloogis kaks faili. Võtmehoidla ning minu sertifikaat. Kui viimane on hoidlasse imporditud, siis võib selle siit kustutada, sest edasised toimingud käivad hoidla kaudu.

```
C:\User\jaagup\0104\turva\sert2>dir
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
JUKU~1   STO           871  03.04.01  21:05 juku.store
```

Kui nüüd soovin Jukule saata signeeritud programmi, siis kõigepealt loon lähtekoodi

```
C:\User\jaagup\0104\turva\sert>edit Teretus.java
```

, kompileerin

```
C:\User\jaagup\0104\turva\sert>javac Teretus.java
```

ning arhiveerin selle jar-faili.

```
C:\User\jaagup\0104\turva\sert>jar cvf teretus.jar Teretus.class
added manifest
adding: Teretus.class(in = 429) (out= 297)(deflated 30%)
```

Edasi signeerin arhiivi oma salajase võtmega

```
C:\User\jaagup\0104\turva\sert>jarsigner -keystore jaagup.store teretus.jar jaagup
Enter Passphrase for keystore: 123456
```

ning saadan signeeritud arhiivi Jukule.

```
C:\User\jaagup\0104\turva\sert>copy teretus.jar ..\sert2
        1 file(s) copied
```

Avastades uue faili oma kataloogist või saades selle kirjaga võiks tal ikka huvi tekkida, kellelt saadetis on tulnud. Jarsigner pakub võimaluse

```
C:\User\jaagup\0104\turva\sert2>jarsigner -verify -keystore juku.store teretus.jar
jar verified.
```

ning teatab, et arhiiv sobis, st., et arhiivi oli signeerinud inimene, kelle sertifikaat asub Juku võtmehoidlas. Kui lisada käivitamisel võtmed -verbose ning -certs, siis on näha, kellelt fail tulnud on ning mis toiminguid kontrollimise ajal tehakse. Uskudes nüüd, et teade tuleb tuttavalt inimeselt kes talle halba ei soovi, võib Juku rahus arhiivi lahti pakkida,

```
C:\User\jaagup\0104\turva\sert2>jar xvf teretus.jar
extracted: META-INF/MANIFEST.MF
extracted: META-INF/JAAGUP.SF
extracted: META-INF/JAAGUP.DSA
  created: META-INF/
extracted: Teretus.class
```

vaadata huvi pärast, mis talle kataloogi tekkinud on

C:\User\jaagup\0104\turva\sert2>dir

```
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
JUKU~1   STO           871  03.04.01  21:05 juku.store
TERETUS  JAR         2 029  03.04.01  21:27 teretus.jar
META-INF       <DIR>        03.04.01  21:33 META-INF
TERETU~1 CLA           429  03.04.01  21:33 Teretus.class
```

ning saabunud programmi käima panna.

```
C:\User\jaagup\0104\turva\sert2>java Teretus
Soovin head lugemist!
```

Selgus, et oli tegemist lihtsa tervitusega.

Kui keegi muu sooviks Jukule minu nime alt kurja programmi saata,

```
class Suurtervitus{
  static void main(String argumendid[]){
   for(int i=1; i<10000; i++){
    System.out.println("Minge kõik kuu peale");
   }
  }
}
```

siis see tal ei õnnestu. Ta võib küll programmi kirjutada, kompileerida ja arhiveerida

```
C:\User\jaagup\0104\turva\sert>jar cvf suurtervitus.jar Suurtervitus.java
added manifest
adding: Suurtervitus.java(in = 158) (out= 130)(deflated 17%)
```

kuid minu privaatvõtmega seda naljalt signeerida ei õnnestu. Kui ta ka pääseks ligi mu kataloogi, siis seal on andmehoidlast võtme kätte saamiseks vaja lahti muukida sealne parool. Mujal aga samasugust võtit välja mõtelda oleks peaaegu lootusetu. Kui õnnetu piraat otsustaks siiski signeerimata või mõne muu võtmega allkirjastatud faili Jukule saata,

```
C:\User\jaagup\0104\turva\sert>copy suurtervitus.jar ..\sert2
        1 file(s) copied
```

siis kohapeal kontrollides selguks, et tegemist pole õige asjaga.

```
C:\User\jaagup\0104\turva\sert2>jarsigner -verify -keystore juku.store
suurtervitus.jar
jar is unsigned. (signatures missing or not parsable)
```

Digitaalallkiri

Mida keytool'i ning jarsigner'i abil saab kasutada valmis vahenditena, seda võib oma programmides ka ise teha, sest eks eelnimetatudki ole Java abil kokku kirjutatud programmid. Võtmepaaride loomiseks on KeyPairGenerator, sinna saab ette anda, millise algoritmi järgi võtmed genereerida. Käsud getPublic ning getPrivate annavad võtmepaarist vastavalt avaliku ning salajase võtme ning getEncoded neist annab võtme sisu baidijadana, mida edaspidi üle kanda või talletada saab. Järgnevas näites luuakse käivitajale failidesse teineteise juurde kuuluvad avalik ning sajalane võti.

```
import java.security.*;
```

```
import java.io.*;
public class Votmetelooja{
  public static void main(String argumendid[]) throws Exception{
    String avavotmefail="avavoti.key";
    String salavotmefail="salavoti.key";
    KeyPairGenerator votmepaarilooja=KeyPairGenerator.getInstance("DSA");
    votmepaarilooja.initialize(512); //jagamisja"a"k
    KeyPair votmepaar=votmepaarilooja.generateKeyPair();
    FileOutputStream valja1=new FileOutputStream(avavotmefail);
    valja1.write(votmepaar.getPublic().getEncoded());
    valja1.close();
    FileOutputStream valja2=new FileOutputStream(salavotmefail);
    valja2.write(votmepaar.getPrivate().getEncoded());
    valja2.close();
  }
}
```

Teate allkirjastamiseks tuleb failist lugeda või muul moel enesele kättesaadavaks teha salajane võti ning teate moodustavad andmebaidid. Salajast võtit hoitakse PKCS#8 standardi järgi, kus lisaks võtme väärtusele on kirjas ka andmed versiooni ning krüptimisalgoritmi kohta. Võtme (tüübist PrivateKey) väljastab KeyFactory ning failist saabuvad baidid aitab viiamasele suupäraseks teha `PKCS8EncodedKeySpec`. Kogu allkirjastamine ise toimub paari käsuga

```
    Signature allkirjastaja=Signature.getInstance("DSA");
    allkirjastaja.initSign(salavoti);
    allkirjastaja.update(andmebaidid);
    byte[] allkiri=allkirjastaja.sign();
```

, kus initSign salajase võtmega määrab, et järgnevalt update abil allkirjastajast läbi lastavad baidid muudavad allkirja ning sign väljastab baitidena allkirja, mis sõltub salajasest võtmest ning andmetest ning mida peaks pea võimatu olema salavõtme puudumisel järele teha.

```
import java.security.*;
import java.security.spec.*;
import java.io.*;
public class Allkirjastaja{
  public static void main(String argumendid[]) throws Exception{
    String salavotmefail="salavoti.key";
    String andmefail="andmed.txt";
    String allkirjafail="andmed.sig";
    byte[] salavotmebaidid=new byte[(int)new File(salavotmefail).length()];
    FileInputStream sisse=new FileInputStream(salavotmefail);
    sisse.read(salavotmebaidid);
    sisse.close();
    PrivateKey salavoti=KeyFactory.getInstance("DSA").generatePrivate(
      new PKCS8EncodedKeySpec(salavotmebaidid)
    );
    byte[] andmebaidid=new byte[(int)new File(andmefail).length()];
    sisse=new FileInputStream(andmefail);
    sisse.read(andmebaidid);
    sisse.close();
    Signature allkirjastaja=Signature.getInstance("DSA");
    allkirjastaja.initSign(salavoti);
    allkirjastaja.update(andmebaidid);
    byte[] allkiri=allkirjastaja.sign();
    FileOutputStream valja=new FileOutputStream(allkirjafail);
    valja.write(allkiri);
    valja.close();
  }
}
```

Kontrollimisel aitab samuti Signature. Käsu initVerify juures määratakse, millise avaliku võtme järele andmete allkirjale vastavust kontrollima hakatakse. Käsk verify väljastab "tõene", kui allkiri, andmed ja võti sobisid kokku, muul juhul loetakse kontroll ebaõnnestunuks.

```
import java.security.*;
import java.security.spec.*;
import java.io.*;
public class Allkirjakontrollija{
  public static void main(String argumendid[]) throws Exception{
```

```
    String avavotmefail="avavoti.key";
    String andmefail="andmed.txt";
    String allkirjafail="andmed.sig";
    byte[] avavotmebaidid=new byte[(int)new File(avavotmefail).length()];
    FileInputStream sisse=new FileInputStream(avavotmefail);
    sisse.read(avavotmebaidid);
    sisse.close();
    PublicKey avavoti=KeyFactory.getInstance("DSA").generatePublic(
      new X509EncodedKeySpec(avavotmebaidid)
    );
    byte[] andmebaidid=new byte[(int)new File(andmefail).length()];
    sisse=new FileInputStream(andmefail);
    sisse.read(andmebaidid);
    sisse.close();
    byte[] allkirjabaidid=new byte[(int)new File(allkirjafail).length()];
    sisse=new FileInputStream(allkirjafail);
    sisse.read(allkirjabaidid);
    sisse.close();
    Signature allkirjakontrollija=Signature.getInstance("DSA");
    allkirjakontrollija.initVerify(avavoti);
    allkirjakontrollija.update(andmebaidid);
    System.out.print("Andmed failist "+andmefail+
                     " ning allkiri failist "+allkirjafail+ " ");
    if(allkirjakontrollija.verify(allkirjabaidid)){
      System.out.println("sobivad.");
    } else {
      System.out.println("ei sobi.");
    }
  }
}
```

Kui programmide tööde tulemusi vaadata, siis kõigepealt loodi võtmefailid

```
AVAVOTI  KEY           244  05.04.01  14:51 avavoti.key
SALAVOTI KEY           202  05.04.01  14:51 salavoti.key
         2 file(s)            446 bytes
         0 dir(s)   1 603 133 440 bytes free
```

seejärel allkiri

```
C:\User\jaagup\0104\turva>java Allkirjastaja
```

ning soovides sobivust kontrollida, saime teada, et andmed ja allkiri sobisid kokku

```
C:\User\jaagup\0104\turva>java Allkirjakontrollija
```

Andmed failist andmed.txt ning allkiri failist andmed.sig sobivad.

```
C:\User\jaagup\0104\turva>type andmed.txt
Juku tuli koolist.


C:\User\jaagup\0104\turva>edit andmed.txt

C:\User\jaagup\0104\turva>type andmed.txt
Juku tuli koolist
```

Kui andmefaili kas või ühe punkti jagu muuta, siis saadakse teade, et

```
C:\User\jaagup\0104\turva>java Allkirjakontrollija
Andmed failist andmed.txt ning allkiri failist andmed.sig ei sobi.
```

Sõnumilühend

Kui piisab vaid tehnilisest teadmisest, et andmed pole tee peal kannatada saanud, siis üheks võimaluseks on saata andmeid mitu eksemplari ja pärast neid omavahel võrrelda. Mahukamate tekstide ja piltide korral aga võib teise eksemplarina kasutada sõnumilühendit. Tegemist on kavala baidijadaga, mida algsest tekstist on vastava algoritmi teel kerge luua, lühendi järgi teksti aga taastada pole võimalik.

Lühendi eeliseks pikema teksti ees on lühidus. Sõltumata algandmete mahust on siinse algoritmi (SHA) järgi lühendiks ikkagi vaid 20 baiti, samas lühendi sisu muutub tundmatuseni, kui selle allikaks olnud kasvõi mitme gigabaidisest andmemassiivist vaid üks bitt muuta. Lühendit võib kasutada ka tervikluse ründe vastu. Kui näiteks tahame kindlad olla, et installeeritavasse tarkvarasse pole keegi pahalane pärast programmi ametlikku turule tulekut trooja hobust ega muud soovimatut lisanud, võime lasta installeerimisplaadist või installeeritud programmist sõnumilühendi arvutada ning tootjale helistada ja uurida, kas need ikka ilusti kokku langevad.

Java vahenditega käib sõnumilühendi loomine paari käsuga. Kõigepealt luuakse sobivat algoritmi oskav räsikoodi (sõnumilühendi) generaator. Käsule digest antakse ette andmebaidid, mille kohta tahetakse lühendit arvutada ning tulemusena väljastatakse baidimassiivina räsikood. Järgnev tsükkel lihtsalt trükib loodud koodi ekraanile.

```
import java.security.*;
public class Turva2{
  public static void main(String argumendid[]) throws Exception{
    String teade="Juku tuli koolist";
    byte[] teatebaidid=teade.getBytes();
    MessageDigest rasikoodigeneraator=MessageDigest.getInstance("SHA");
                    //Secure Hash Algorithm
    byte[] rasikood=rasikoodigeneraator.digest(teatebaidid);
    for(int i=0; i<rasikood.length; i++){
      System.out.print((rasikood[i] & 0xFF)+" ");
    }
  }
}
```

väljund:

```
233 166 35 250 225 13 241 133 131 60 82 76 215 146 95 66 163 89 142 64
```

Programmi õigused

Kui tekib oht, et käivitatav programm võimaldab meie andmetele või arvutile enesele mingil moel kurja teha, siis tuleks programmi õigusi nii piirata, et võiksime sel rahuliku südamega lasta tööd teha kartmata et salaja kas programmi kirjutaja pahatahtlikkuse või lihtsalt lohakuse tõttu võime millestki ilma jääda või et andmed, mida soovime vaid enese teada hoida, iseeneslikult võõraste silmade ette jõuavad. Rakendil näiteks on keelatud kohaliku failisüsteemiga suhelda ning võõrastesse masinatesse võrgu kaudu ühenduda, samuti mikrofonist häält lindistada. Nii võib kurjalt lehelt avatud programm küll hulga aknaid avada ning mälu ja võrguliiklust ummistada, kuid korralikult koostatud turvaaukudeta brauseri korral midagi tähtsat masinast kasutaja teadmata võrgu peale rändama või kaduma minna ei saa. Kui tekivad kahtlused rakendi kavatsuste osas, siis saame brauseri sulgeda ning kõik on vanaviisi nagu ennegi. Korraldatud on turvamine nii, et potentsiaalselt turvaohtlike käskude algusse on sisse kirjutatud pöördumine turvahalduri (SecurityManager) poole. Kui saadakse nõusolek, võib töö jätkuda, muul juhul heidetakse erind ning programm kas lõpetab töö või jätkab vastavalt erindi püüdmisega kaasnevatele korraldustele. Kuigi võimalikke käske, mis võiksid kasutajale liiga teha, on palju ning neid juurde kirjutades saaks kokku lugematu hulga, õnnestub kogu turvalisust siiski paarikümne õigusega juhtida, sest operatsioonisüsteemi ning muude turvaohtlike käskude poole pöördutakse vaid üksikutes kohtades ning kõik muud vastavat teenust vajavad käsud kasutavad sama "lüüsi". Näiteks kettalt lugemisel pöördutakse alati FileInputStream'i konstruktori poole, ükskõik kas on tegemist pildi avamise, teksti

lugemise või kahendfaili analüüsiga. Kui selles konstruktoris lasta turvahaldurilt kontrollida, kas lugemine lubatud, siis saabki ketta kasutusõiguse ühes kohas määrata ning määrang kehtib igal pool kogu programmi ulatuses. Kui objektorienteerituse teoreetikud rõhutavad pea igal võimalikul juhul, et kui vähegi võimalik, siis ei tohi koodi kopeerida vaid tuleb terviku moodustavad käsud ühte meetodisse koondada ning see siis igas vajalikus kohas kas otse või teiste meetodite kaudu välja kutsuda. Nii võib programm mõnigikord pikemaks ja mõne intepretaatori korral ka aeglasemaks minna, kuid kui igal toimingul on oma kindel koht, siis saab selle toiminguga seotud vea kergesti üles leida ning vajadusel käitumist ka muuta või keelata.

Kui iseseisval programmi laseksin ühendada end teise masinasse ning sealselt ajateenuselt kella küsida siis võrguühenduse olemasolul ning teise serveri korrasoleku puhul tõenäoliselt ilmub mulle ekraanile teises masinas olev kellaaeg.

```
C:\User\jaagup\0104\turva>type Kell1.java
import java.net.*;
import java.io.*;
public class Kell1{
  public static void main(String argumendid[]) throws Exception{
    Socket sc=new Socket("madli.ut.ee", 13);
    BufferedReader sisse=new BufferedReader(
      new InputStreamReader(sc.getInputStream())
    );
    System.out.println(sisse.readLine());
  }
}

C:\User\jaagup\0104\turva>java Kell1
Wed Apr  4 20:20:07 2001
```

Soovides aga hoolitseda, et käivitatav programm heast peast ei hakkaks ennast masinast väljapoole ühendama ega muid kahtlasi toiminguid tegema, võib virtuaalmasinale määrata uue kurja turvahalduri, kelle poolest vaid üksikuid toimingud on lubatud. Kui nüüd sama programm käivitada, siis teatatakse, et keelatud on juba madli.ut.ee-le vastava IP aadressi küsimine rääkimata sinna ühendumisest.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager Kell1
Exception in thread "main" java.security.AccessControlException: access denied (
java.net.SocketPermission madli.ut.ee resolve)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkConnect(SecurityManager.java:1042)
        at java.net.InetAddress.getAllByName0(InetAddress.java:559)
        at java.net.InetAddress.getAllByName0(InetAddress.java:540)
        at java.net.InetAddress.getByName(InetAddress.java:449)
        at java.net.Socket.<init>(Socket.java:100)
        at Kell1.main(Kell1.java:9)
```

Vaikimisi turvahaldur lubab vaid lugeda faile jooksvast ning sellele alanevatest kataloogidest. Pea kõik muu, mida vähegi keelata annab on keelatud. Kui soovida koostada programmi tarvis sobivat turvataset, võib hakata algsele peaaegu nulltasemele tasapisi õigusi juurde andma. Õiguste kirjeldused tuleb panna policy-failidesse, kust neid siis virtuaalmasinal programmi käivitamise ajal lugeda lasta.

```
C:\User\jaagup\0104\turva>type vork.policy
grant{
  permission java.net.SocketPermission "*.ut.ee", "connect";
};
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vork.policy Kell1
Wed Apr  4 20:21:15 2001
```

Käskluse grant abil määratakse, millises kohas (kataloogis, failis, serveris) asuvatele programmidele õigused kehtivad. Kui grant'i järel tulevad kohe loogilised sulud, siis kehtivad õigused kõikidele käivitavatele programmidele sõltumata asukohast. Kui ühendatava masina aadressis osa tärniga märkida, siis pääseb ligi kõikidele kirjeldusele vastavatele masinatele. Täpsemaks minnes võib aga piirduda ka ühe masina väratipiirkonna või sootuks ühe väratiga, kui soovime hoolitseda, et programm tõesti ainult vaid meile tuntud ja teatud kohaga piirdub.

```
C:\User\jaagup\0104\turva>type vork2.policy
grant{
  permission java.net.SocketPermission "madli.ut.ee:10-15", "connect";
};
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vork2.policy Kell1
Wed Apr  4 20:21:54 2001
```

Turvahalduri võib seada ka käsuga System.setSecurityManager. Kui eelmisel juhul pidi programmi käivitaja hoolitsema (lipuga -Djava.security.manager), et tööle lastud elukas midagi liialt kahtlast teha ei saaks, siis nii paneb programm seestpoolt omale ise päitsed pähe.

```
C:\User\jaagup\0104\turva>type Turva5.java
import java.io.*;
public class Turva5{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new SecurityManager());
    PrintWriter valja=new PrintWriter(new FileWriter("nimed.txt", true));
    valja.println("Siim");
    valja.close();
  }
}
```

Et vaikimisi õiguste juures faili kirjutamist ei lubata, siis antakse käivitamisel veateade.

```
C:\User\jaagup\0104\turva>java Turva5
Exception in thread "main" java.security.AccessControlException: access denied (
java.io.FilePermission nimed.txt write)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkWrite(SecurityManager.java:978)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:96)
        at java.io.FileWriter.<init>(FileWriter.java:52)
        at Turva5.main(Turva5.java:5)
```

Kui anda ennast kinni talitsenud programmile veidi õigusi, siis saab ta oma etteantud ülesannetega ilusti hakkama. Lisati õigus kirjutada kohaliku kataloogis faili nimed.txt ning võimegi näha kuidas pärast programmi töö lõppu seal Siimu nimi ilutseb.

```
C:\User\jaagup\0104\turva>type nimed.policy
grant{
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed.policy Turva5
```

```
C:\User\jaagup\0104\turva>type nimed.txt
Siim
```

Nagu ennist mainitud, võib codeBase abil määrata, kus asuvatele programmidele määratavad õigused laienevad. Nii on õigused kataloogile ning programm töötab.

```
C:\User\jaagup\0104\turva>type nimed_a.policy
grant codeBase "file:/C:/user/jaagup/0104/turva/" {
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_a.policy Turva5
```

Nõnda võivad faili "nimed.txt" kirjutada kõik programmid, mis asuvad kataloogis C:/user/jaagup/0104/ või sellest alanevates kataloogides. Kuna file:/C:/user/jaagup/0104/turva/Turva5 vastab sellele tingimusele, tuleb nime kirjutamine välja.

```
C:\User\jaagup\0104\turva>type nimed_b.policy
grant codeBase "file:/C:/user/jaagup/0104/-" {
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_b.policy Turva5
```

Kui õiguste kehtivuse piirkonnaks on aga vaid programmi ülemkataloog

```
C:\User\jaagup\0104\turva>type nimed_c.policy
grant codeBase "file:/C:/user/jaagup/0104/" {
 permission java.io.FilePermission "nimed.txt", "write";
};
```

, siis faili kirjutamine ei õnnestu.

```
C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_c.policy Turva5
Exception in thread "main" java.security.AccessControlException: access denied (
java.io.FilePermission nimed.txt write)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkWrite(SecurityManager.java:978)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:96)
        at java.io.FileWriter.<init>(FileWriter.java:52)
        at Turva5.main(Turva5.java:5)
```

Omaloodud turvahaldur

Klassi SecurityManager võib laiendada ning seal olevaid meetodeid üle katta. Kõikide turvakontrollide puhul käivitatakse meetod checkPermission ning parameetrina antakse toiming, mille teostamiseks luba küsitakse. Juhul kui turvahaldur leiab, et küsitud toimingu tarvis ei tohi luba anda, heidetakse meetodist välja erind SecurityException. Allloodud turvahalduri laiendaja Lubaja1 on eriti sõbralik: siin vaid trükitakse välja, mille kohta õigust sooviti saada ning mingeid toiminguid ei piirata.

```
C:\User\jaagup\0104\turva>type Lubaja1.java
import java.security.*;
class Lubaja1 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
  }
```

```
}

C:\User\jaagup\0104\turva>type Turva5b.java
import java.io.*;
public class Turva5b{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new Lubaja1());
    PrintWriter valja=new PrintWriter(new FileWriter("nimed.txt", true));
    valja.println("Siim");
    valja.close();
  }
}
```

Väljatrükist on näta, et õigusi küsiti neljal korral. Lisaks faili kirjutamise õigusele käivad standardosa programmid küsimas õigusi ka jooksva kataloogi nime teada saamiseks ning uurivad enne järele, kas rea vahetamiseks vajalikku sümbolit tohib pärida.

```
C:\User\jaagup\0104\turva>java Turva5b
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Soovides teada, mida algne turvahaldur koos policy-failidest saadud vihjetega peab vajalikuks keelata, tuleb välja kutsuda ülemklassi samanimeline meetod ehk super.checkPermission. Et sealtkaudu tekkivad erindid siin näites programmi tööd ei katkestaks, selleks on käsule püünis ümber pandud ning trükitakse välja, milliseid toiminguid algne haldur ei luba.

```
C:\User\jaagup\0104\turva>type Lubaja2.java
import java.security.*;
class Lubaja2 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
    try{
      super.checkPermission(p);
    } catch(Exception e){
      System.out.println("Probleem: "+e);
    }
  }
}
C:\User\jaagup\0104\turva>java Turva5c
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
Probleem: java.security.AccessControlException: access denied (java.util.Propert
yPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
Probleem: java.security.AccessControlException: access denied (java.util.Propert
yPermission user.dir read)
(java.io.FilePermission nimed.txt write)
Probleem: java.security.AccessControlException: access denied (java.io.FilePermi
ssion nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Virtuaalmasina turvahalduri väga leplikuks muutumiseks tuleb määrata õiguseks java.security.AllPermission. Nii võib mõnest paigast või ka igalt poolt pärit programmidel lubada kõike ette võtta.

```
C:\User\jaagup\0104\turva>type koiklubatud.policy
grant{
  permission java.security.AllPermission;
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=koiklubatud.policy Turva5c

(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Turvahalduri õiguste määramine

Ise lubades ja keelates tuleb meetodi ülekatmise juures uurida, millist tüüpi õigusesoov parameetrina anti ning vastavalt sellele reageerida sarnaselt nagu algnegi turvahaldur seda teeb. Kui tundub, et programm küsib liialt suuri õigusi, tuleb välja heita SecurityException ning soovitavalt konstruktori parameetrina antava teatega seletada mille vastu eksiti või millised tegevused lubatud on.

```
C:\User\jaagup\0104\turva>type Lubaja3.java
import java.security.*;
import java.io.FilePermission;
class Lubaja3 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
    if(p instanceof FilePermission){
      if(!p.getName().toLowerCase().startsWith("t")){
        throw new SecurityException("Lugemiseks lubatud vaid t-ga algavad
failinimed");
      }
    }
  }
}
```

Vaikimisi paigutushalduri puhul on kohaliku kataloogi failidest lugemine lubatud.

```
C:\User\jaagup\0104\turva>type Turva4a.java
import java.io.*;
public class Turva4a{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new SecurityManager());
    BufferedReader sisse=new BufferedReader(new FileReader("nimed2.txt"));
    System.out.println(sisse.readLine());
  }
}
```

Käivitamisel väljastab programm

```
C:\User\jaagup\0104\turva>java Turva4a
Katrin
```

, mis on ka loogiline, sest tekstifaili esimesel real asus nimi Katrin.

```
C:\User\jaagup\0104\turva>type nimed2.txt
Katrin
Kati
Kai
```

Kui aga määrata turvahalduriks isend, kes lubab tegelda vaid t-ga algavate failinimedega,

```
C:\User\jaagup\0104\turva>type Turva4c.java
import java.io.*;
public class Turva4c{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new Lubaja3());
    BufferedReader sisse=new BufferedReader(new FileReader(argumendid[0]));
    System.out.println(sisse.readLine());
  }
}
```

siis antakse faili lugemiseks avamisel veateade.

```
C:\User\jaagup\0104\turva>java Turva4c nimed2.txt
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed2.txt read)
Exception in thread "main" java.lang.SecurityException: Lugemiseks lubatud vaid
t-ga algavad failinimed
        at Lubaja3.checkPermission(Lubaja3.java:8)
```

```
        at java.lang.SecurityManager.checkRead(SecurityManager.java:890)
        at java.io.FileInputStream.<init>(FileInputStream.java:61)
        at java.io.FileReader.<init>(FileReader.java:38)
        at Turva4c.main(Turva4c.java:5)
```

Kui aga lugeda sama sisuga faili, mille nimi algab t-ga, siis lugemine õnnestub. Kõigepealt kirjutab turvahaldur välja, mille kohta luba küsiti ning lõpuks on ilusasti näha nimi, mis leiti faili esimeset realt.

```
C:\User\jaagup\0104\turva>java Turva4c tydrukud.txt
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission tydrukud.txt read)
Katrin
```

Hoiatusribaga aken

Terasemal jälgimisel olete võinud märgata, et rakendi poolt avatud akende allservas on pea alati kirjas Java Applet Window, kohapeal käivitatud programmi akende puhul aga sellist riba ei leia. Riba on mõeldud turvahoiatusena, et kasutaja teaks arvestada salapärase veebilehelt avanenud aknaga, mis muidu sarnaneb kõigi teiste akendega ning võib ennast maskeerida suisa kohaliku parooli küsiva sisestusakna sarnaseks, kuid võib saabunud andmed ilma pikema jututa saata veebilehe omanikule kel edaspidi siis nende üle vaba voli on. Riba ei teki raami alla mitte mingi ime läbi vaid virtuaalmasinas on nii korraldatud, et ribata raami saamiseks peab eraldi luba olema. Tavalistel käivitatavatel programmidel on selline luba olemas kuid rakenditel ning uue vaikimisi turvahalduriga programmidel sellist õigust pole ning seetõttu surutakse nende poolt avatud raamidele kasutajat hoiatav tempel külge.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager Raam1

C:\User\jaagup\0104\turva>type Raam1.java
import java.awt.Frame;
public class Raam1{
  public static void main(String argumendid[]){
    Frame f=new Frame("Iseseisev raam");
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```

Et omaloodud või määratud turvahalduri korral raami all olevast hoiatusribast vabaneda, tuleb ribast vabanemise õigus programmile juurde anda.

```
C:\User\jaagup\0104\turva>type vabaraam.policy
grant {
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
};
```

ning siis näeme tavalist akent nagu ikka harjunud nägema oleme.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vabaraam.policy Raam1
```

Valikuline õiguste loetelu

java.io.FilePermission	Asukoht, näit. /home/jaagup file://C:/temp/ file://C:/temp/- file://C:/temp/koolid.txt	Tegevus read, write, execute,delete
java.net.SocketPermission	masin[:värat] lin2.tpu.ee:79 *.ut.ee madli.ut.ee:7-80 * lin2.tpu.ee:80-	accept, connect, listen, resolve
java.util.PropertyPermission	* java.home java.*	read, write
java.lang.RuntimePermission	createClassLoader setSecurityManager exitVM setIO stopThread	
java.awt.AWTPermission	showWindowWithoutWarningBanner accessClipboard accessEventQueue listenToAllAWTEvents readDisplayPixels	
java.security.AllPermission		

Atribuudid (Properties)

Alati ühesuguse väärtuse (näiteks terve koera jalgade arv) saab kirjutada otse koodi sisse, kuigi pea alati oleks see viisakam koodi loetavuse huvides kusagil konstandina kirja panna ning edasi vastava nime järgi väärtust kasutada. Kui väärtus võib programmi käigus muutuda, siis enamasti võib vastava suuruse

meelespidamiseks võtta muutuja ning sinna siis vajalikul hetkel andmeid talletada või sealt lugeda. Kui salvestatavate parameetrite arv pole kindlalt teada, siis on mõistlikum kasutada muutujate asemel muid vahendeid. java.util.Hashtable sisaldab võtmete ja väärtuste paare nagu ka muud liidest Map realiseerivad klassid. Võtmeid võib alati juurde lisada ning võtme järgi saab väärtuse kätte. Väärtused võivad korduda, võtmed aga mitte. Nii on kindlalt määratud, et igale võtmele vastab korraga vaid üks väärtus. Sõnede tarvis on loodud Hashtable alamklass java.util.Properties, kus saab stringe teineteisega vastavusse seada, vastavusi küsida, faili talletatada, voogu pidi edasi kanda või taas voost kasutamiseks välja lugeda.

```
import java.util.Properties;
public class Atribuudid1{
  public static void main(String argumendid[]){
    Properties p1=new Properties();
    p1.setProperty("kokk","Maali");
    p1.setProperty("kassapidaja", "Juuli");
    System.out.println("Täna keedab suppi "+p1.getProperty("kokk"));
  }
}
```

Väljundina teatatakse:
Täna keedab suppi Maali
, sest küsitud parameetrile (võtmele) kokk vastas nimi Maali. Kui polnuks kokka määratud ega teda ka kusagilt mujalt teada saadud, siis väljastataks tühiväärtus null.

Kui tahta andmeid failis säilitada ning hiljem tarvitada, siis tuleb määrata (faili)voog andmete saatmiseks, andmeid kirjeldav pealkiri ning käsuga store saata andmed tekstina teele.

```
import java.io.*;
import java.util.Properties;
public class Atribuudid2{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=new Properties();
    p1.setProperty("kokk","Maali");
    p1.setProperty("kassapidaja", "Krõõt");
    p1.store(new FileOutputStream("toitjad.properties"), "Toitev personal");
  }
}
```

Andmed jõudsid ilusti faili. Trellidega algav rida loetakse kommentaariks ning seda uuel sisselugemisel ei arvestata. ASCII sümbolid kuni 127-ni salvestatakse nii nagu nad on, ülejäänute puhul kasutatakse Unicode sümbolite salvestamiseks tehtud kuju, kus sümbol algab \u-ga ning järgnevad neli märki tähistavad sümbolile vastavat arvu kuueteistkümnendsüsteemis. Soovides andmeid meile tuttava kooditabeli järgi lugeda, tuleks kirjutada native2ascii toitjad.properties, mis loeb faili ning väljatrükil teisendab sümbolid võimaluse korral loetavamale kujule. Kui soovida sellist andmefaili parandada ning ei jõua pidevalt mõtelda tähekoodide peale, siis võib teksti rahumeeli täpitähti kasutades valmis kirjutada ning hiljem kirjutada native2ascii -reverse failinimi > muudetud_faili_nimi , tulemusena peaksid sinna jõudma suuremate koodinumbritega väärtused \uxxxx kujul, kust virtuaalmasin oma vahenditega taas andmeid lugeda suudab.

```
#Toitev personal
#Mon Apr 09 10:48:12 GMT+03:00 2001
kokk=Maali
kassapidaja=Kr\u00F5\u00F5t
```

Failist või mujalt voost loetakse andmed sisse käsuga load. Edasi võib juba atribuutide objekti kasutada nagu ennistki. Kui me pole kindlad, kas meie küsitud võti ja väärtus andmete hulgas leidub, siis võib käsklusele getProperty anda kaks argumenti: võtme

ning vaikeväärtuse. Niimoodi ei väljastata väärtuse puudumise korral tühiväärtust null vaid võetakse vaikeväärtus-

```
import java.io.*;
import java.util.Properties;
public class Atribuudid3{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=new Properties();
    p1.load(new FileInputStream("toitjad.properties"));
    System.out.println("Täna keedab suppi "+p1.getProperty("kokk"));
    System.out.println("Veevärki hoiab korras "+
                                p1.getProperty("remondimees", "Ivan"));
  }
}
```

Väljund:

```
Täna keedab suppi Maali
Veevärki hoiab korras Ivan
```

Kui andmete hulka lisada ka remondimehe nimi, siis kuvatakse see ka väljundis sellisena nagu see failis kirjas on:

```
#Toitev personal
#Mon Apr 09 10:48:12 GMT+03:00 2001
kokk=Maali
kassapidaja=Kr\u00F5\u00F5t
remondimees=Maksim

Väljund
Täna keedab suppi Maali
Veevärki hoiab korras Maksim
```

Klassi Properties kasutab ka virtuaalmasin oma kasutatavate parameetrite hoidmiseks. Milliste parameetrite ja väärtustega on tegemist, selle saab järele uurida klassi System käsuga getProperties. Kui turvahaldur lubab, võib ükshaaval väärtusi lugeda ja muuta, samuti käsuga list soovitud voogu välja trükkida.

```
import java.io.*;
import java.util.Properties;
public class Atribuudid4{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=System.getProperties();
    p1.list(System.out);
  }
}
```

Süsteemi juurde kuuluvaid parameetreid saab ka käsurealt määrata. Piisab vaid ette kirjutada -D ning võti=väärtus (appletvieweri puhul -J-Dvõti=väärtus) kui juba ongi meie antud andmed atribuudiloendis kirjas. Nagu alljärgnevast loetelust näha võib, asuvad käsurealt antud väärtused võrdsetena nende kõrval, mida virtuaalmasin oma südamest pidas sobivaks avaldada. Kala tuli sisse uuena, kasutaja endine nimi jaagup aga muudeti jukuks.

```
C:\User\jaagup\0104\k1>java -Dkala=angerjas -Duser.name=juku Atribuudid4
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=C:\PROGRAM FILES\JDK130\JRE\bin
java.vm.version=1.3.0-C
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\User\jaagup\0104\k1
java.runtime.version=1.3.0-C
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
os.arch=x86
```

```
java.io.tmpdir=C:\WINDOWS\TEMP\
line.separator=

java.vm.specification.vendor=Sun Microsystems Inc.
java.awt.fonts=
os.name=Windows 95
java.library.path=C:\PROGRAM FILES\JDK130\BIN;.;C:\WIND...
kala=angerjas
java.specification.name=Java Platform API Specification
java.class.version=47.0
os.version=4.0
user.home=C:\WINDOWS
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding=Cp1257
java.specification.version=1.3
user.name=juku
java.class.path=.
java.vm.specification.version=1.0
java.home=C:\PROGRAM FILES\JDK130\JRE
user.language=et
java.specification.vendor=Sun Microsystems Inc.
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode
java.version=1.3.0
java.ext.dirs=C:\PROGRAM FILES\JDK130\JRE\lib\ext
sun.boot.class.path=C:\PROGRAM FILES\JDK130\JRE\lib\rt.ja...
java.vendor=Sun Microsystems Inc.
file.separator=\
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport...
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
user.region=EE
sun.cpu.isalist=pentium i486 i386
```

Krüptimine

Andmeid võõrastele nähtamatuks muuta on püütud juba aastatuhandeid. Algselt tegeldi põhiliselt tähtede vahetamise ning järjekorra muutmisega. Arvutiajastul leitakse, et sobivam on kodeerida bitijada.

Üks plokk

Krüptimisalgoritme on mitmesuguseid. Märgatav osa neist kodeerivad andmeid plokkide kaupa. DES-i nimelisel algoritmil on ploki suuruseks 8 baiti, mis tähendab, et kodeeritavad andmete baitide arv peab jaguma kaheksaga. Nii andmed (avatekst) kui võti on bitijada kujul. Võtme pikkus on ühekordses DESis alati 8 baiti, andmete maht pole aga piiratud. Siin näites on andmeteks määratud plokk, kus kõik bitid on nullid ning võtmeks plokk, kus kõik bitid ühed.

Enne kodeerima asumist tuleb luua Javale kasutataval kujul võtmeobjekt (SecretKey). Edasi luua ja initsialiseerida kodeerija, määrates kasutatava algoritmi.Edasi tuleb hakata andmeid kodeerijast läbi laskma. Lühemate andmete puhul nagu siin, võib kohe anda käskluse doFinal ja saada lõpptulemuse kätte. Kui aga andmed ulatuvad megabaitidesse või gigabaitidesse ning nende korraga mällu lugemine pole mõistlik ega võimalik, siis on õnnestub šifreerida ka väiksemate osade kaupa ning ühelt poolt andmeid sisestada ja teiselt poolt väljastada ja talletada.

Viimane for-tsükkel loodi vaid tulemuste vaatamiseks, kõikide baitide väärtused trükitakse välja kuueteistkümnendsüsteemis.

```
import javax.crypto.*;
import javax.crypto.spec.*;
public class DESKrypt{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid=new byte[8];
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      avatekstibaidid[i]=(byte)0x00;
```

```
      votmebaidid[i]=(byte)0xFF;
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti);
    byte[] salatekst=kodeerija.doFinal(avatekstibaidid);
    for(int i=0; i<salatekst.length; i++){
      System.out.print(Integer.toString(salatekst[i] & 0xFF, 16));
    }
  }
}
```

Ning programmi töö tulemuseks siis väljatrükitud krüptogramm.

```
D:\Home\jaagup\rak>java DESKrypt
caaaaf4deaf1dbae6aec3f62f460a08d
```

Šifreereeritud voog

Et Javas õnnestub enamike andmetega voogude abil suhelda, siis on ka krüptimise tarbeks voog loodud. Sest kui üks voog suudab võtta vastu objekte, teine neid kokku pakkida ja kolmas faili kirjutada ning kõik need vood võib omavahel kokku liita, siis on ju mugav ka luua voog kokku- ja lahtikrüptimiseks, et vajadusel selline ühenduslüli lihtsalt ahelasse juurde panna. Seda ülesannet täidab CipherOutputStream. Nii nagu plokkidegi kaupa krüptides, tuleb ka siin võti luua ja initsialiseerida. Vaid andmete sisestamine ja edastamine sarnaneb voogudega seotud lähenemisele, kus write-käsu abil baidimassiivist andmeid saata võib.Ning et andmed ilusti kättesaadavad püsiksid, selleks saadab loodud krüptiv voog oma andmed FileOutputStream'i abil DES2.dat-nimelisse faili.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt2{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid="Juku tuli kooli.  ".getBytes();
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      votmebaidid[i]=(byte)0xFF;
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti);
    CipherOutputStream salavaljundvoog=new CipherOutputStream(
      new FileOutputStream("DES2.dat"), kodeerija
    );
    salavaljundvoog.write(avatekstibaidid);
    salavaljundvoog.close();
  }
}
```

Kui programm käima pandud

```
D:\Home\jaagup\rak>java DESKrypt2
```

siis tekib jooksvasse kataloogi vastava nimega fail, praegusel juhul 24 baidi pikkune, ehk sama pikk kui sisendandmed.

```
D:\Home\jaagup\rak>dir des2.dat
 Volume in drive D is DATA
 Volume Serial Number is 3842-03F3

 Directory of D:\Home\jaagup\rak
```

```
19.03.2004  18:13                24 DES2.dat
               1 File(s)             24 bytes
               0 Dir(s)   9 682 993 152 bytes free
```

Püüdes faili sisu tekstina vaadata, väljub sealt vaid hulk suhteliselt seosetuid sümboleid ning algset teksti sealt ilma võtit teadmata peaks olema küllalt lootusetu kätte saada

```
D:\Home\jaagup\rak>more DES2.dat
;├[↕q|Ñ}]Ax-s÷:Rw┼Çп╬Ⱦ
```

Kui aga võti teada, siis võib kokku panna sarnase sisendvoo ning soovitud teate taas välja lugeda. Kõik muu näeb välja samasugune, vaid kodeerija ja voo suunad on vastupidised. Andmed loetakse baidimassiivi ning sealt omakorda muudetakse selgema väljatrüki huvides tekstiks.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt2a{
  public static void main(String argumendid[]) throws Exception{
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      votmebaidid[i]=(byte)0xFF;
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.DECRYPT_MODE, voti);
    CipherInputStream salasisendvoog=new CipherInputStream(
      new FileInputStream("DES2.dat"), kodeerija
    );
    byte[] b =new byte[1000];
    int pikkus=salasisendvoog.read(b);
    System.out.println(new String(b, 0, pikkus));
  }
}
```

Kui kompileeritud kood tööle panna, võib rõõmsasti algset teadet imetlema jääda. Ja kõike vaid seetõttu, et teada oli krüptimisel kasutatud võti. Praegusel juhul kaheksa baiti, kõik bitid püsti.

```
D:\Home\jaagup\rak>java DESKrypt2a
Juku tuli kooli.
```

Parooliga krüptimine

Kaheksabaidine võti on ilus lühike küll, aga kuutkümmend nelja bitti lihtsalt pähe õppida on mõnevõrra tülikas. Olgugi, et juhuslikult genereeritud väärtused peaksid kokku andma kõige raskemini ära arvatava vastava pikkusega kombinatsiooni, on inimesed harjunud meeles pidama vähemasti veidigi loogilisema ülesehitusega paroole. Siin on nõnda kokku pandud räsikoodi ja krüptimise võimalused, et parool ei pea olema kindla pikkusega.

Parooli hoidmiseks ja edastamiseks kasutatakse Java juures tavalise Stringi asemel tähemassiivi. Nõnda on võimalik paranoilisemate juhtude korral masina mälus vastavad baidid pärast kasutamise lõppu ükshaaval üle kirjutada ning pole nii suurt muret, et keegi neid mälutõmmiste abil lugema pääseks.

```
import javax.crypto.*;
```

```
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt3{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid=(loeRida("Teade:")+"       ").getBytes();
    byte[] sool=new byte[8];
    for(int i=0; i<8; i++){
      sool[i]=(byte)0xFF;
    }
    int segamiskordadearv=10;
    SecretKey voti=SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(
      new PBEKeySpec(loeRida("Parool:").toCharArray())
    );
    Cipher kodeerija=Cipher.getInstance("PBEWithMD5AndDES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti, new PBEParameterSpec(sool,
segamiskordadearv));
    CipherOutputStream salavaljundvoog=new CipherOutputStream(
      new FileOutputStream("DES3.dat"), kodeerija
    );
    salavaljundvoog.write(avatekstibaidid);
    salavaljundvoog.close();
  }
  public static String loeRida(String kysimus) throws IOException{
    System.out.println(kysimus);
    return new BufferedReader(new InputStreamReader(System.in)).readLine();
  }
}
```

Käivitamisel küsitakse nii teadet kui parooli:

```
D:\Home\jaagup\rak>java DESKrypt3
Teade:
Tere tulemast
Parool:
kalake
```

Tulemuseks on krüptogramm, mis nagu nõutud, ei näe kuigivõrd eelnevate järgi välja.

```
D:\Home\jaagup\rak>more DES3.dat
ôÆjce═·σ?►Rß∞Σêiíû0é■ß
```

Soovides algseid andmeid kätte saada, tuleb kogu toiming tagurpidi läbi käia. Sool ja segamiskordade arv peavad kattuma. Loodud nad selleks, et ka krüptitud paroolifailide väljatuleku korral poleks liialt lihtne võrrelda ja leida ühesuguseid paroole.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt3a{
  public static void main(String argumendid[]) throws Exception{
    byte[] sool=new byte[8];
    for(int i=0; i<8; i++){
      sool[i]=(byte)0xFF;
    }
    int segamiskordadearv=10;
    SecretKey voti=SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(
      new PBEKeySpec(DESKrypt3.loeRida("Parool:").toCharArray())
    );
    Cipher kodeerija=Cipher.getInstance("PBEWithMD5AndDES");
    kodeerija.init(Cipher.DECRYPT_MODE, voti,
                     new PBEParameterSpec(sool, segamiskordadearv));
    CipherInputStream salasisendvoog=new CipherInputStream(
      new FileInputStream("DES3.dat"), kodeerija
    );
    int nr=salasisendvoog.read();
    while(nr != -1){
      System.out.print(((char)nr));
      nr=salasisendvoog.read();
    }
    System.out.println();
  }
}
```

Kui õige parool ette anda, siis võib rõõmsasti teadet lugeda.

```
D:\Home\jaagup\rak>java DESKrypt3a
Parool:
kalake
Tere tulemast
```

Juhtus aga parool ununema, tuleb selleks korraks teatest suu puhtaks pühkida.

```
D:\Home\jaagup\rak>java DESKrypt3a
Parool:
kass
&■e?k?X↔4FkNHª♥?
```

Ülesandeid

Signeerimine

```
* Koosta enesele keytool'i abil sertifikaat
* Loo lihtne programm, paki see jar-faili.
* Signeeri loodud arhiiv.
* Anna oma avalikku võtit sisaldav sertifikaat pinginaabrile
  ja võta tema oma vastu.
* Saada signeeritud arhiivifail naabrile ning kontrolli ise naabrilt
  saabunud arhiivifaili õigsust.
* Kontrolli, et signeerimata või tundmata võtmega signeeritud arhiivi
  puhul antaks veateade.
```

Krüptograafia

```
* Tutvu näidetega
* Sifreeri arvude 0-15 ruute tähistavad baidid (0, 1, 4, 9, 16 ...
225) tähistavad baidid DES-i abil võtmega, mille 8 baiti on väärtused
alates  kahest (2, 4, 6 ...).
* Kontrolli tulemust desifreerimise abil.
* Sama võtmega krüpti kasutaja poolt määratud failis asuvad andmed.
* Loo failide krüptimiseks/dekrüptimiseks graafiline liides. Parool
ning  failinimi küsi kasutaja käest.
```

Digitaalallkiri

```
* Tutvu näidetega.
* Loo enesele võtmepaar.
* Signeeri tekst salajase võtmega
* Kontrolli allkirja sobivust.
* Loo graafiline liides, kus saab luua võtmepaari, signeerimisel ning
  kontrollimisel valida võtmefaili , allkirjafaili ja andmefaili.

* Lisa vahend sõnumilühendi loomiseks ning etteantud faili baitide
```
esitamiseks kuueteistkümnendsüsteemis.