1 Κρυπτογράφηση

Τα αρχεία του χρήστη στην περίπτωση που είναι μεγαλύτερα από 1MB τότε "σπάνε" σε αρχεία του 1MB. Όταν έχουμε ένα αρχείο 50MB τότε θα δημιουργηθούν 50 μικρότερα αρχεία (chunks) το ενός MegaByte. Για να αναδομηθεί σωστά το αρχείο, θα πρέπει και τα 50 chunks να μπουν στη σωστή σειρά. Επομένως έχουμε πάρα πολλούς συνδυασμούς (πόσους???) για να πάρουμε με τη σωστή σειρά τα chunks.

Όλη την πληροφορία για το πως θα ενωθούν τα chunks για να φτιάξουν το αρχικό αρχείο και γενικά τη δομή του προσαρτημένου καταλόγου την έχει το αρχείο με τα μεταδεδομένα (meta-data). Κάποιος κακόβουλος χρήστης, αν αποκτήσει το συγκεκριμένο αρχείο, θα μπορέσει να αναδομήσει όλο τον προσαρτημένο κατάλογο. Λόγω της φύσης των δεδομένων που αποθηκεύονται στο αρχείο αυτό, δύσκολα να ξεπεράσει σε μέγεθος τα 10MB. Αυτό σημαίνει ότι θα "σπάσει" σε 10 μικρότερα αρχεία. Ακόμα και έτσι είναι αρκετά εύκολο για κάποιον να φτιάξει το αρχικό αρχείο που θα του δώσει τον προσαρτημένο κατάλογο.

Για τον παραπάνω λόγο κρίθηκε αναγκαία η κρυπτογράφηση του αρχείου με τα metadata. Οι αλγόριθμοι που χρησιμοποιήθηκαν καθώς και ο τρόπος που τους υλοποιεί η Java παρουσιάζονται παρακάτω.

1.1 Παραγωγή κλειδιού

Για την κρυπτογράφηση είναι απαραίτητη η δημιουργία ενός κλειδιού κρυπτογράφησης. Συγκεκριμένα χρησιμοποιήθηκε η PBKDF2 (Password-Based Key Derivation Function) για την παραγωγή του κλειδιού. Για την παραγωγή του MAC (Message Authentication Code) χρησιμοποιήθηκε ο αλγόριθμος HMAC με αλγόριθμο κατακερματισμού SHA-1

Για αλγόριθμος κρυπτογράφησης χρησιμοποιήθηκε ο AES με μέγεθος κλειδιού 128 bit, σε CBC (Cipher-Block Chaining) mode και για padding αυτό που ορίζει το standard PKCS#5 (Public-Key Cryptography Standards νούμερο 5).

1.2 Υλοποίηση σε Java

Αρχικά για την παραγωγή του κλειδιού θα πρέπει να δημιουργήσουμε ένα byte array για την αποθήκευση του salt όπως φαίνεται παρακάτω:

```
1 SecureRandom sr=new SecureRandom();
2 byte[] salt=new byte[8];
3 sr.nextBytes(salt);
```

Στη συνέχεια για τη δημιουργία του κλειδιού δημιουργείται ένα αντικείμενο τύπου *SecretKeyFactory*. Έπειτα δημιουργούμε τις προδιαγραφές του κλειδιού, δηλαδή τον κωδικό που δίνει ο χρήστης, το salt που χρειάζεται και το μέγεθος του κλειδιού. Τέλος παίρνουμε το αντικείμενο *secret* τύπου *SecretKey* που είναι το τελικό μας κλειδί. Για τον αλγόριθμο κρυπτογράφησης αρκεί μία γραμμή για να πάρουμε το αντικείμενο *cipher*. Όπως φαίνεται παρακάτω δίνουμε σαν όρισμα τις προδιαγραφές του αλγόριθμου κρυπτογράφησης και τον provider των μεθόδων κρυπτογράφησης Στην συγκεκριμένη περίπτωση χρησιμοποιούμε τον Bouncy Castle.

```
1 SecretKeyFactory factory=SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
2 KeySpec spec=new PBEKeySpec(Constants.password,salt,1024,128);
3 SecretKey tmp=factory.generateSecret(spec);
4 SecretKey secret = new SecretKeySpec(tmp.getEncoded(),"AES");
5
6 Cipher cipher=Cipher.getInstance("AES/CBC/PKCS5Padding", "BC");
```

Έως αυτό το σημείο, τα βήματα είναι τα ίδια για την κρυπτογράφηση και την αποκρυπτογράφηση.

1.2.1 Κρυπτογράφηση

Για την κρυπτογράφηση, αρχικοποιούμε το αντικείμενο *cipher* για κρυπτογράφηση Για την αποκρυπτογράφηση χρειάζεται το salt και το initialization vector, οπότε και τα γράφουμε σε δύο αρχεία. Τέλος με τις μεθόδους *cipher.update()* και *cipher.doFinal()* κρυπτογραφούμε όσα bytes διαβάζουμε και τα γράφουμε σε ένα άλλο αρχείο.

```
1  cipher.init(Cipher.ENCRYPT_MODE,secret);
2  AlgorithmParameters params=cipher.getParameters();
3  byte[] iv=params.getParameterSpec(IvParameterSpec.class).getIV();
4
5  File saltFile=new File(Constants.personalDir+"/.salt");
6  FileOutputStream saFos=new FileOutputStream(saltFile);
7  saFos.write(salt);
8  saFos.flush();
9  saFos.close();
10
11 File ivFile=new File(Constants.personalDir+"/.iv");
12 FileOutputStream ivFos=new FileOutputStream(ivFile);
13 ivFos.write(iv);
14 ivFos.flush();
15 ivFos.close();
16
17 while((bytesRead=fis.read(buffer))!=-1){
18     byte[] output=cipher.update(buffer,0,bytesRead);
19     if(output!=null)
20         fos.write(output);
21 }
22 byte[] output=cipher.doFinal();
23 if(output!=null)
24     fos.write(output);
25 fos.flush();
```

1.2.2 Αποκρυπτογράφηση

Για την αποκρυπτογράφηση χρειαζόμαστε το salt και το initialization vector που χρησιμοποιήθηκαν κατά την κρυπτογράφηση οπότε και τα διαβάζουμε από τα αρχεία που τα είχαμε αποθηκεύσει. Έπειτα αρχικοποιούμε το αντικείμενο *cipher* για αποκρυπτογράφηση, διαβάζουμε τα bytes από το κρυπτογραφημένο αρχείο και πάλι με τις μεθόδους *cipher.update()* και *cipher.doFinal()* τα αποκρυπτογραφούμε και τα γράφουμε σε ένα άλλο.

```
1  File saltFile=new File(Constants.personalDir+"/.salt");
2  FileInputStream saFis=new FileInputStream(saltFile);
3  saFis.read(salt);
4  saFis.close();
5
6  File ivFile=new File(Constants.personalDir+"/.iv");
7  FileInputStream ivFis=new FileInputStream(ivFile);
8  ivFis.read(iv);
9  ivFis.close();
10
11 cipher.init(Cipher.DECRYPT_MODE,secret,new IvParameterSpec(iv));
12
13 while((bytesRead=fis.read(buffer))!=-1){
```

```
14          byte[] output=cipher.update(buffer,0,bytesRead);
15          if(output!=null)
16              fos.write(output);
17      }
18      byte[] output=cipher.doFinal();
19      if(output!=null)
20          fos.write(output);
21      fos.flush();
22      fis.close();
23      fos.close();
```