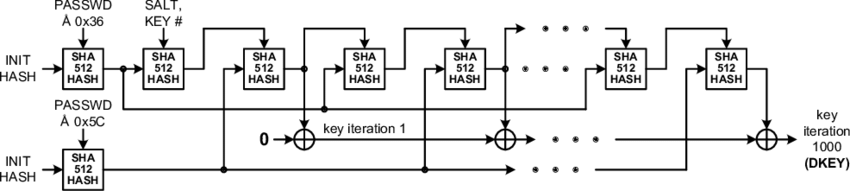
PBKDF2 Cryptography

Για την κρυπτογράφηση των κωδικών στην βάση δεδομένων έγινε χρήση του αλγορίθμου **PBKDF2(**Password-Based Key Derivation Function 2**)**.Ο παραπάνω αλγόριθμος κρίθηκε ικανοποιητικός λόγω της καλής αναλογίας ασφάλειας και απόδοσης που παρέχει απέναντι σε επιθέσεις τύπου brute-force και dictionary σε μία βάση δεδομένων. Όσο ανεβαίνει η ισχύς των υπολογιστών, τόσο πιο ευάλωτο είναι ένα είδος κρυπτογράφησης. Το πλεονέκτημα που παρέχει αυτός ο αλγόριθμος είναι η δυνατότητα να γίνει πιο αργός με βάση των αριθμό των επαναλήψεων και ως αποτέλεσμα να παράγεται ένα πιο δυνατό hash, καθιστώντας τον με αυτό τον τρόπο σχετικά πιο ασφαλή στον παραπάνω κίνδυνο που αναφέρθηκε. Για την παραγωγή του hash λαμβάνει ως παραμέτρους έναν αλγόριθμο SHA(Secure Hash Algorithms), τον αριθμό των επαναλήψεων(iteration number) που θα πρέπει να γίνουν για την παραγωγή του και το salt, το οποίο είναι ένας αριθμός από τυχαία παραγμένα bytes.



Εικόνα 1 Περιγραφή της λειτουργίας του PBKDF2WithHmacSHA512

Η κλάση PasswordAuthentication περιέχει τις ακόλουθες μεθόδους:

• private static byte[] pbkdf2(char[] password, byte[] salt, int iterations)

Η παραπάνω μέθοδος επιστρέφει ένα πίνακα από bytes που περιέχει το κλειδί που παράχθηκε χρησιμοποιώντας τον ορισμένο αλγόριθμο PBKDF2WithHmacSHA512.

• public String hash(char[] password)

Η παραπάνω μέθοδος λαμβάνει έναν κωδικό σε έναν πίνακα χαρακτήρων και επιστρέφει το ολοκληρωμένο hash σε string για να αποθηκευτεί στην βάση δεδομένων. Η διαδικασία είναι η παρακάτω με την ακόλουθη σειρά βημάτων:

-Παραγωγή ενός πίνακα τύπου bytes μέσω της κλάσης SecureRandom της java, ο οποίος ορίζεται ως το salt.

-Κλήση της συνάρτησης pbkdf2(…), η οποία μας επιστρέφει το παραγόμενο κλειδί σε έναν πίνακα τύπου bytes.

-Δημιουργία ενός πίνακα τύπου bytes συγχωνεύοντας το κλειδί(προήλθε από την κλήση της συνάρτησης pbkdf2(…)) και το salt. To salt τοποθετείται στην αρχή και στην συνέχεια ακολουθεί το κλειδί.

-Τέλος, μέσω της κλάσης Base64.Encoder κωδικοποιούμαι τον παραπάνω πίνακα τύπου bytes σε base64 και επιστρέφουμε το τελικό hash που έχει το συγκεκριμένο pattern: *ID* + cost + '$' + hash όπου ως ID είναι ένας αναγνωριστικός αριθμός για να γνωρίζουμε ότι το συγκεκριμένο hash προήλθε από αυτή την κλάση και ως cost είναι το κόστος των επαναλήψεων (αριθμός επαναλήψεων = 2^κοστος) που θα χρειαστεί να το γνωρίζουμε σε επόμενο βήμα.

• public boolean authenticate(char[] password, String token)

Η συγκεκριμένη συνάρτηση συγκρίνει εάν ο κωδικός που τοποθέτησε στο login o χρήστης ταιριάζει με το hash που είναι αποθηκευμένο για τον ίδιο στην βάση δεδομένων.

- char[] password, ο κωδικός του χρήστη σε πίνακα τυπου char.

-String token, το hash που υπάρχει για τον ίδιο μέσα στην βάση δεδομένων.

Return true εάν ο κωδικός ταιριάζει με το hash και false σε αντίθετη περίπτωση.

Η διαδικασία σύγκρισης του κωδικού με το hash είναι η ακόλουθη:

-Αρχικά, συγκρίνουμε εάν το δοσμένο hash ταιριάζει με το pattern : .*ID* + cost + '$' + hash . Εφόσον ταιριάζει, προχωράμε στο επόμενο βήμα.

-Υπολογίζουμε τον αριθμό των επαναλήψεων με βάση το κόστος που μας δίνει το hash μέσω του pattern του.

-Κάνουμε decode το hash από base64.

-Τοποθετούμε το salt που βρίσκεται στην αρχή του hash σε έναν πίνακα τύπου Bytes.

-Τέλος, καλούμε την μέθοδο pbkdf2(char[] password, byte[] salt, int iterations), οπού της δίνουμε τον κωδικό του χρήστη, το salt που διαχωρίσαμε από το hash που υπάρχει καταχωρημένο στην βάση δεδομένων και τον αριθμό των επαναλήψεων που χρειαζονται, ώστε στην συνεχεία να τσεκάρουμε εάν το hash που θα παραχθεί ταιριάζει με το υπάρχον στην βάση.

