CPE 640 Homework 3

Issued: 04/07/2015, Due: 04/21/2015

Web Programming using JSP and JavaScript

This homework is done in groups (same as your group project arrangement) with final write up (in addition to moodle email submission of the code) consisting of:

- how the group effort was organized ("who did what"),
- time taken to complete each section (individual person hours and total person hours for each task of the homework), along with,
- final UML diagram representing the static class view and the dynamic process view, and the use case view of the project, as discussed in class.

Using the latest version of Tomcat, eclipse and JSP, build a functional user content management system,

Problem 1: Implement the SHA-256 bit algorithm to encrypt short passwords using Javascript.(Estimated Time of completion: 1 Week)

SHA256 belongs to the latest and the strongest family of cryptographic hash functions. The "256" stands for the length (in number of bits) of hash value calculated on an input raw message of length 512 bits. The password encryption is performed locally on the browser, without sending the plain-text password to the server. Write a Javascript program that can be run on the browser to encrypt the entered password in the html form.

Relevant part of the algorithm is also available from Wikipedia page on SHA-256 algorithm

http://en.wikipedia.org/wiki/SHA-2

and at the link to the final standard released by Federal Information Processing Standards (FIPS).

http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

The pseudocode for the SHA-256 algorithm is described below.

Note 1: All variables are unsigned 32 bits in hexadecimal format and wrap modulo 2^{32} when calculating.

Note 2: All constants in this pseudo code are in big endian (meaning most significant part of the number is to the left and the least significant part to the right)

The Algorithm

Initialize variables

(first 32 bits of the *fractional parts* of the square roots of the first 8 primes 2..19):

h[0..7] :=

0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19

Initialize table of round constants

(first 32 bits of the *fractional parts* of the cube roots of the first 64 primes 2..311):

k[0..63] :=

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, xc19bf174,
0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

Pre-processing:

1. append the bit '1' to the message
2. append k bits '0', where k is the minimum number >= 0 such that the resulting message length (in *bits*) is modulo 512, minus 64 bits for the length.
3. append length of message (before pre-processing), in *bits*, as 64-bit big-endian integer

Note: In this homework, the password is an 8-16 character word whose characters are converted to its respective ASCII equivalent (8 bits each) thus resulting in a 64-128 bit message.

Process the message in successive 512-bit chunks:

```
1. break message into 512-bit chunks
2. for each chunk
     break chunk into sixteen 32-bit big-endian words = w[0..15]
```

Extend the sixteen 32-bit words into sixty-four 32-bit words:

```
for i from 16 to 63{
        s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15]
rightshift 3)
        s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2]
rightshift 10)
        w[i] := w[i-16] + s0 + w[i-7] + s1
}
```

Initialize hash value for this chunk:

```
a := h0,
b := h1,
c := h2,
d := h3,
e := h4,
f := h5,
g := h6,
h := h7
```

Main loop:

```
    for i from 0 to 63{
        s0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22);
        maj := (a and b) xor (a and c) xor (b and c);
        t2 := s0 + maj;
        s1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25);
        ch := (e and f) xor ((not e) and g);
        t1 := h + s1 + ch + k[i] + w[i];

        h := g;
        g := f;
        f := e;
        e := d + t1;
        d := c;
```

```
        c := b;
        b := a;
        a := t1 + t2;
}
```

```
Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h
Produce the final hash value(digest) (big-endian):
digest = hash = h0 append h1 append h2 append h3 append h4 append h5 append
h6 append h7
```

Remember that the password is encrypted on the client before sending it to the server via plain http. The raw password is never revealed to the administrator or the user. Only the encrypted versions of the passwords are compared for authentication. The uniqueness of SHA256 hash ensures that no two raw messages produce the same hash values, in other words, the hash function is collision free. The strength of the hash function ensures that there is no direct approach other than brute force evaluation of hash to decipher the raw message, which is O(2^256).

Validate your SHA256 hashing function with the following online resource for a few password strings.

http://www.xorbin.com/tools/sha256-hash-calculator

or see the FIPS SHA256 documentation for detailed step by step result after each round for a few test messages.

```
SHA256("")
0x e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

```
SHA256("The quick brown fox jumps over the lazy dog")
0x d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
SHA256("The quick brown fox jumps over the lazy dog.")(Note the period at the end)
0x ef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c
```

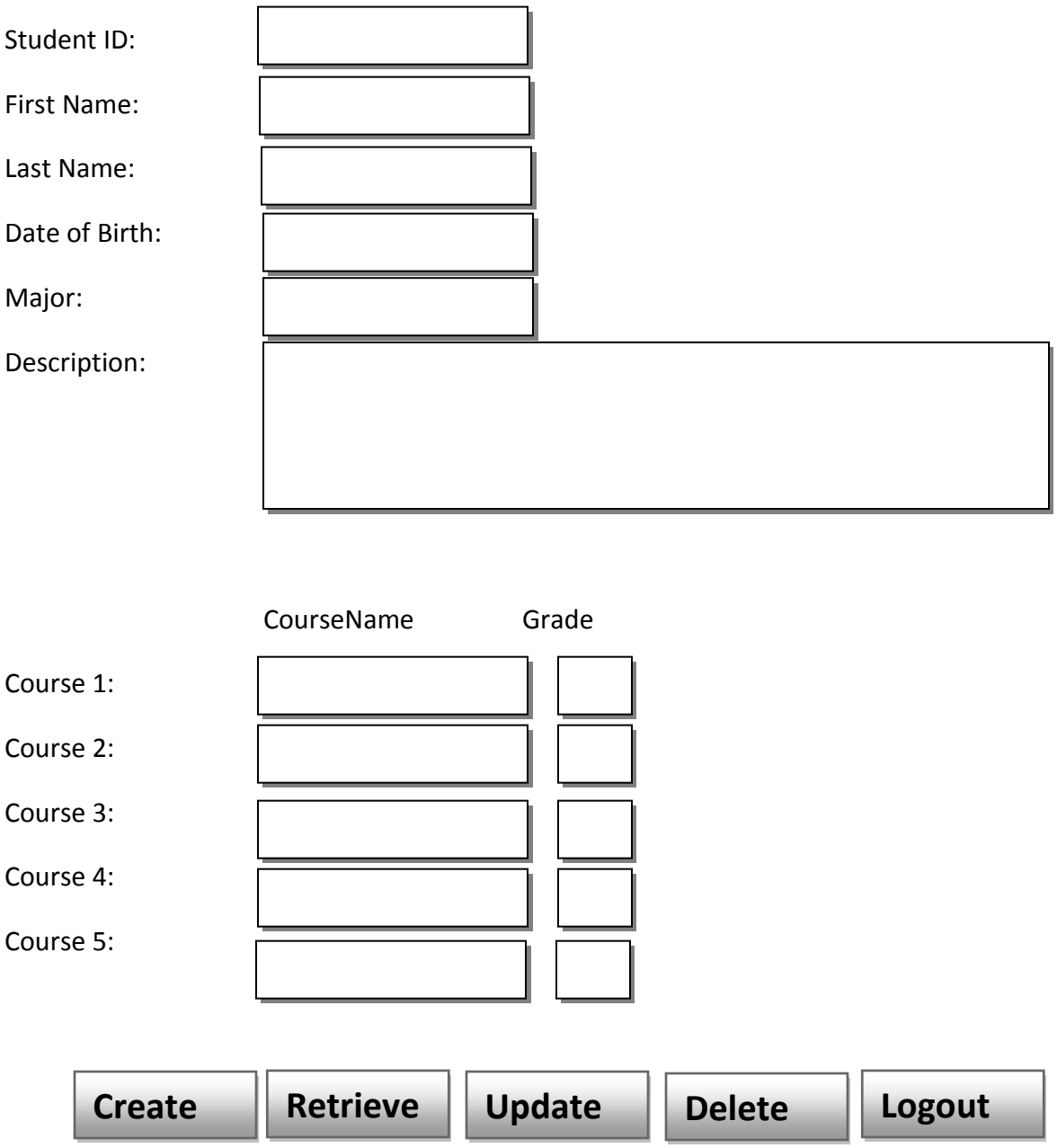
Problem 2: (Access Control Validation) (Estimated time of completion: 1 week)

Design a simple login interface that collects the username and password of the user, encrypts the password using the SHA2 algorithm developed in the previous problem on the client side and sends it via http to the server. Upon receiving the username and password, the server checks it against the database of usernames and encrypted passwords, validates the session and returns a session id to the client.

Note that the interface for data submission and authentication can be worked on in parallel. The final functional version of the SHA256 function can then be plugged in to the existing client code for encryption for access control interface.

Problem 3: (Student Content Management System) (Estimated Time of Completion: 1 week)

Design a web based student content management system that accepts student information and stores, updates, retrieves or deletes them from a server.



Create action shall assign a new student id and display it on the student id field, based on existing list of students on the server. A new line is appended to the flat text file on the server (comma separated values) with the current student id and blank fields. The student id is then displayed in the corresponding text box on the page.

Retrieve action shall retrieve the student information based on the student id. An unknown id, which is not present on the server will produce an error ("student not found").

Update action shall update (overwrite) the information for the student based on the corresponding student id field. An unknown id, which is not present in the server will produce

an error(“student not found”) otherwise the student information is updated following a confirmation on the page “update successful”.

Delete action shall delete the student record from the server text file.

Logout action shall destroy the current session id on the server.

Note that this part can also be designed in parallel with the help of JAVA string operations and file IO, in the same amount of time.

Now combine the three parts to produce a functional content management system which authenticates a user based on the username and encrypted password information available on the server, and takes the user to the student information management interface by creating a valid session id. The session id is destroyed upon logout.