SHActivity

ICT2105 Mobile Application Development Spring 2018

Some background…

This quiz requires you to address a toy problem of encrypting a message permanently (no prior knowledge of encryption is required).

SHA-256 is a Secure Hash Algorithm that converts any input string (the *message*) into a unique 256-bit output string (the *digest*). In this quiz, you will be implementing an app that allows users to encrypt a secret text sentence using SHA-256. Since we want the user to feel 15 times exponentially more secure, we will rehash the sentence 2^{15} times repeatedly. As you would've guessed, this is computationally expensive and we will defer this task to native code.

What is rehashing in the first place you may ask? Hashing once means you apply the SHA-256 hashing algorithm once on a message to get a digest. Rehashing means you re-apply SHA-256 to the digest to get a subsequent rehashed digest. Rehashing 2^3 times means you re-apply SHA-256 repeatedly to each subsequent resultant digest repeatedly for a total of 8 times, i.e.,

```
sha256(sha256(sha256(sha256(sha256(sha256(sha256(sha256(message))))))))
```

As hash browns always taste better with some salt, you will add *salt* to your rehashes as well. **So what is rehashing with salt?** Well it turns out that rehashing naively like above does not add much value to the strength of your encryption (i.e., hackers just need to guess how many times you rehashed). So enter the salt. This means at each rehashing step, you will do `sha256(message + salt)` instead of just `sha256(message).` This salt must also be different each time. For this quiz we have provided a C++ static method that creates unique salts (using the same SHA-256 algorithm) according to the hashing iteration you are at. So if you are at the first hashing iteration, you will use `sha256(message + getSalt(0))`, and then second iteration `sha256(message + getSalt(1))` and so on. For more information on this, go google "hash salt" (remember not to accidentally type "brown" in there, else you will feel too hungry during the quiz, although I'd actually suggest learning more about hash browns only after the quiz when you have time).

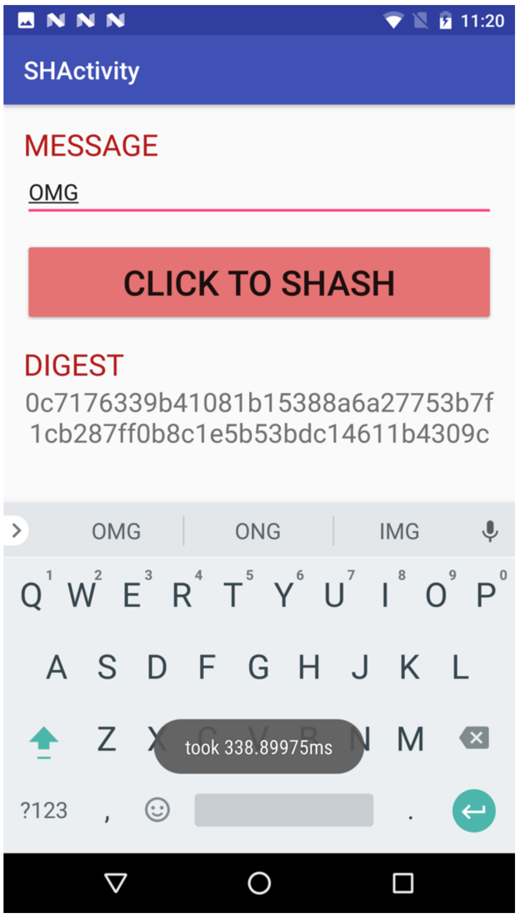
What needs to be done

Fork the repo **ict2105-quiz04-2018** and inspect the code within the project.

Examine the skeleton code before working on the lab. Refer to the screenshots on the following page for the following descriptions.

You are required to implement the main Activity (**SHActivity):**

1. (1 mark) Provide an EditText UI element (id: **textEditMessage**) for users to type in a message string of arbitrary length, and a TextView element (id: **textViewDigest**) for users to see the output digest.
2. (1 mark) Provide a Button UI element that, when clicked, generates the hashed message digest in the TextView. The label text for this button must be "**CLICK TO SHASH**"
3. (3 mark) Create the **public native function `shaMe()`** to compute the SHA-256 hashed string using the rehashing with salt algorithm described above. The hash computation must be done natively in C/C++ code. For a single hash, you must use the C++ class provided in **sha-256.cpp** . Our scripted tests will be based on the algorithm in this C++ class, and not your own SHA algorithm. Use the hashed results in the screenshots to help you check whether you have done the hashing correctly. Your outputs must match the digest EXACTLY as the algorithm above guarantees unique hashed digests.
4. (1 mark) Create a Toast to show the time taken in milliseconds as shown in the screenshots.
5. (1 mark) Create a thread (pick an appropriate threading pattern) to call the native hashing function. Your app will **ANR and crash** if you do not do this. If your app crashes during operation, you will get 3 marks deducted.
6. (1 mark) Modular code design.
7. (1 mark) Well-commented code.
8. (1 mark) Good naming conventions for methods and variables.


SHActivity

MESSAGE

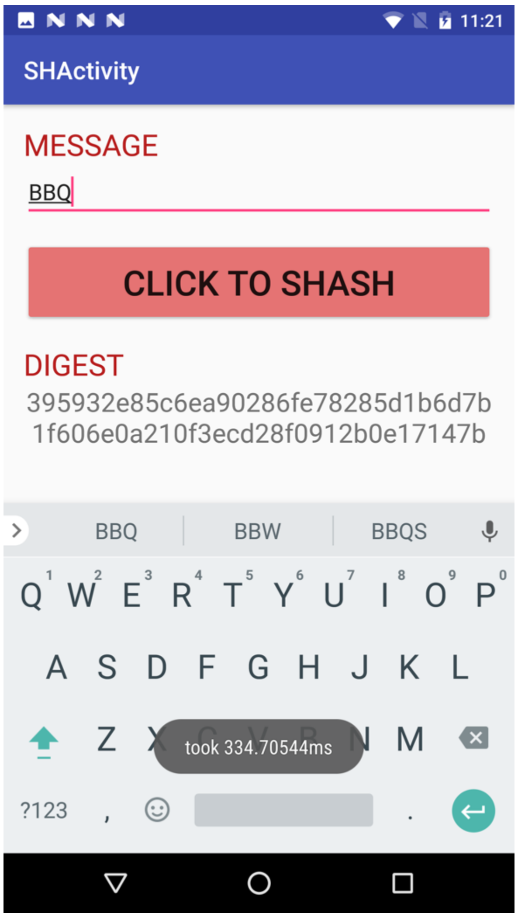
OMG

CLICK TO SHASH

DIGEST

0c7176339b41081b15388a6a27753b7f1cb287ff0b8c1e5b53bdc14611b4309c

took 338.89975ms


SHActivity

MESSAGE

BBQ

CLICK TO SHASH

DIGEST

395932e85c6ea90286fe78285d1b6d7b1f606e0a210f3ecd28f0912b0e17147b

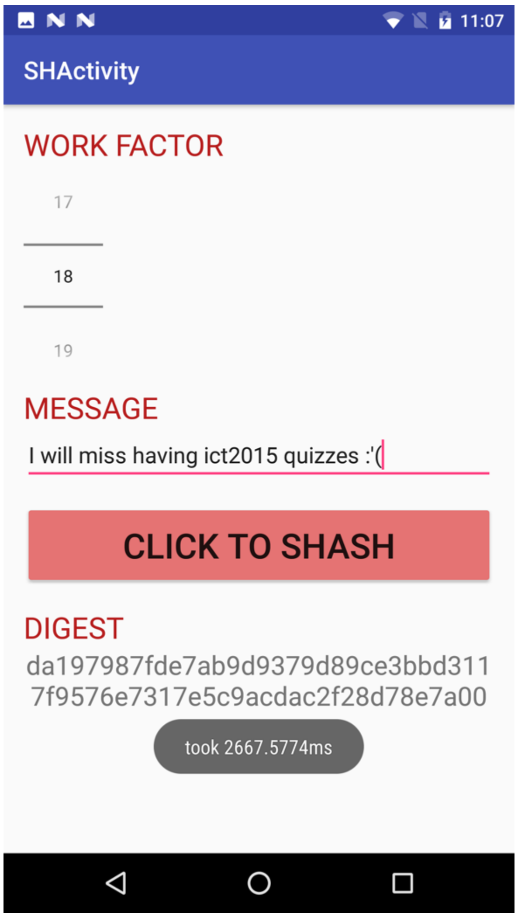
took 334.70544ms

The Final Stretch Goal (Bonus)

Having a fixed number of hashed iterations is no fun, or secure. So in this section, you are required to implement the necessary components to enable users to change the work factor number. Check out the screenshot below for an illustration of what needs to be done, as also described below:

S1. (1 mark) Provide a NumberPicker UI element (id: **numPickerWorkFactor**) to select the work factor required for rehashing. Set the min value of the NumberPicker to be “1” and the max value to be “30”.

S2. (1 mark) Relay this work factor value to your C++ code to enable the correct number of hashing iterations depending on the work factor. Note that the higher the number, the longer it should take for the digest message to appear. The work factor value should be of type **short**.


SHActivity

WORK FACTOR

17

18

19

MESSAGE

I will miss having ict2015 quizzes :'(

CLICK TO SHASH

DIGEST

da197987fde7ab9d9379d89ce3bbd3117f9576e7317e5c9acdac2f28d78e7a00

took 2667.5774ms

Lab Quiz 4

1. Fork the repo **ict2105-quiz04-2018**, and then clone it. This code is incomplete.
2. Create and implement any native code needed in the **/cpp** directory. Do not change the directory name, or put your native code elsewhere.
3. Complete implementation of the code in **SHActivity**.
4. Create and implement any other helper classes you need.
5. Commit and push all changes to your forked repository **ict2105-quiz04-2018. Do not change the main activity name, repository name or package name.**

END OF DOCUMENT