ECE 458 (Spring 2021) - Assignment 2: Digital Signatures and Cryptographic Hash Functions

Due on Wednesday, July 7, 2021, 11:59 P.M.

The main objective of this assignment is to implement a simple blockchin with Proof of Work (PoW) protocol using Digital Signatures and Cryptographic Hash Functions. The implementation must be done in **python3**, and you need to make sure your code compiles on ECE Linux. **You are allowed to use any additional libraries, modules or packages that you'd like. However you are required to provide a Makefile file and a file explaining the instruction to install them and how to run your code. The suggested libraries and modules are 'numpy', 'math', 'hashlib' and 'random'.** The assignment is done in groups of 2 students.

You need to submit a Zip file containing your code and a report on Learn. The report should be in PDF format. You can either use Latex or Word and convert it to PDF. If you prefer handwriting your answers, you need to make sure they are clear and readable, scan them (or take clear pictures) and include them in your PDF report. You only need to submit 1 submission per group. Make sure the report contains the names of all of your group members.

You are allowed to work with your classmates and discuss ideas and clarify the instructions; however, the submitted work MUST BE STRICTLY YOUR OWN WRITING. This also applies to looking up ideas on the Internet or textbooks.

The standard advice on what to do about collaborations to avoid getting in trouble is: you may discuss and share ideas with classmates, but then, put aside any annotations and DO NOT LOOK at them when writing your own solution (this ensures that your writing is original and distinct). Also, NEVER share any of your written solutions or your written code with anyone; preferably, DO NOT ALLOW ANYONE to look at your written solutions (keep in mind that if they have a good memory and write the exact text in your solutions and submit it, you will be equally liable in the academic dishonesty incident|see Course Outline).

It is assumed that you have carefully read the additional conditions included in the Course Outline (available through LEARN) regarding academic integrity, assignments submissions, etc.

Overview

In this exercise, you will implement a DSS and hash chain based authentication (a simple blockchain) of length 3 as shown in figure below, with a simplified network structure in **python3**. You need use the DSS algorithm as given in your lecture slides as the digital signature algorithm and SHA3-224 as a hash function in DSS.

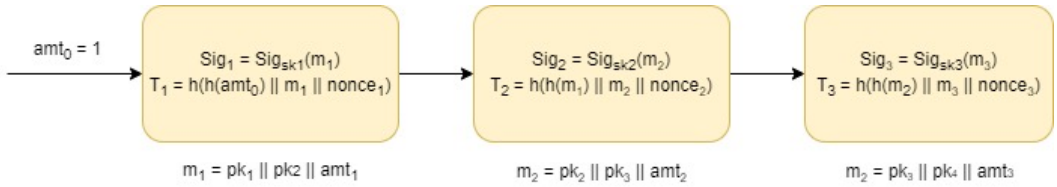

Figure 1: A block chain of length 3

Your task is to generate Sig_i and T_i for i = 1; 2. T_i is the Proof-of-Work (PoW) of that Block. We generally store the public key of each node along with the public address of the node. However, in this project, for simplicity we use the public keys as the public addresses as well. In this project, we omit the completing processing in a simple blockchain. So you just need to find two PoWs. Your code can have any structure or number of functions that you find suitable. **You need to have clear comments in your code as comments have 10 marks.**

You should use the following parameters:

p = 168199388701209853920129085113302407023173962717160229197318

545484823101018386724351964316301278642143567435810448472465887 1

432229345451549430057142651244452442479887774717731938471315140 8

303074040754323361669655019764351945813446570069156968090556800 00

630258300895992604000962594307264986830871384154651074 99

q = 95945266147545120932543359563494111215000386582 1

g = 943891927763273985898453269803498145264338690934127823454309

460592065688040051816008558259061429672718725483758773894987581

254043322344496846135078946138504377502996390063812318343513353 7

262152973355498432995364505138912569755859623649866375135353179 3

626707987717707118474306269548642698889883711135675028 52

pk1 = 493360183248080935347335488404117524857260585278296306689 6

748056885475641656749621629491905191014868618662270686970232166 4

465094703247368646506821015290302480990450130280616929226917246 2

551470632923017242976806834012586361821855991241311700775484507 5

429408372888507551698514494498492001013849289727206925716 0

sk1 = 432398415306986194693973996870836079581453988813

sk2 = 165849943586922055423650237226339279137759546603

sk3 = 627658512551971075308886219669315148725310346887

amnt0 = 1, amnt1 = 2, amnt2 = 3, amnt3 = 4

$nonce_i$, i = 1; 2 are 128-bit that you'll need to find in this project using the instruction below.

Instructions

1. **[15 marks]** Implement a function to perform square and multiply algorithm. This function takes (g,x,n) as inputs and the output is $g^x mod n$. Once you have the function, verify (sk1, pk1) and generate the other two pairs of private and public key (ski; pki); i = 2; 3

Include the three pairs of (ski,pki) for i = 1, 2, 3 in your report.

2. **[10 marks]** Implement a function that takes an integer of any length and outputs the SHA3-224 in binary representation. Print the SHA3-224 of the numbers 10, 20 and 30 and include them in your report. Using 'hashlib' library is recommended.

Note: you may need to convert the output to integer in some parts of the implementation.

To verify your work, the output of the number 10 should be (It is fine if your output has the leading zeros as well):

10011110100001101111111101101001010101010111110010101001010111110100 00

00010111110000100000010010011010010110100001011011001110001110001110 10

10000001100110110011000010011110111010010100001011110100100000101011 01

10101010001011

Note: Depending on the approach you use to find the hash of an integer, you may get a different result. If you're having difficulties getting the above hash, you can explain the approach you used to find the hash of an integer. As long as your approach is correct and you're consistent with your approach in this assignment, you'll get full marks. See Piazza post (@120) for more details.

3. **[10 marks]** SHA3-224 has an input 1152-bit string and output 224-bit string (i.e.(r; c) = (1152; 448)). Any input of 1152 bits would make the hash function inefficient and slow. For simplicity, you may work on only one block of input to SHA3-224. knowing the output of SHA3-224 has 224 bits and each nonce has 128 bits, each message should only be 800 bits. Each amount has 2 bits. So in order to make the length of the message shorter than 800 bits, you may only use the the most significant 399 bits of pks when making each message.

Implement a function that generate the messages as described above. The output should be an integer. Include the result of m1 and m2 in your report.

To verify your work, m1 = generate_message(pk1, pk2, amnt1) =

365994925241611024384269563519318617533231406238127784

027569728070618158438754809801398506612223847655456215

464778150797585415038472343151776312375262437581444708

474252108665778396807853172296143884711480927645629612

79398753963135003232097 74

4. **[15 marks]** Implement a DSS module that is used to sign transactions and verify the signed transactions. Here you may randomly pick an integer k for each signature generation and use SHA3-224 as a hash function in DSS. Generate Sig1 and Sig2 using your implementation and verify Sig1, and include them in your report. You can use the DSS algorithm in your lecture slides.

5. **[15 marks]** Proof-of-Work: Find Nonce1 and Nonce2. Using SHA3-224 as a hash function h in T1 and T2 computations, find a pre-image of h such that the most significant 24 bits of each transaction are zeros. **Add a function that verifies your nonce value.** Calculate the time it took to find each Nonce value and include that in your report.

$$T_1 = \underbrace{00\cdots0}_{24}\underbrace{*\,*\cdots*}_{200}$$
$$T_2 = \underbrace{00\cdots0}_{24}\underbrace{*\,*\cdots*}_{200}$$

6. **[15 marks]** Comment your code and explain your implementation for each part in your report.

7. **[20 marks]** Explain the following questions, in your report:

- How many exhaustive search is required for varying nonce in order to get such a hash value that satisfies the the equations in question 5 with k leading zeros. In order words, how many different values we need to try so that we are guaranteed that at least once of them generates a hash with k leading zeros.
- Explain how much more effort did you need to put if we wanted to find each nonce with 32 leading zeros.

Note Provide a Makefile and instruction to run your code. You will lose marks if we cannot run your code!