Course Project 1 Description

Project 1: Implementation of DSA and proof-of-work in Bitcoin blockchain

1 Introduction

Payments are made in bitcoins (BTC's), which are digital coins issued and transferred by the Bitcoin network. The data of all these transactions, after being validated with a proof-of-work system, is collected into what is called the blockchain. You have learned how transactions are formed in the Bitcoin blockchain and how each block is added, i.e., how a hash-chain is formed through the proof-of-work (PW) mining process.

In this project, you will implement the Digital Signature Standard (DSS) as a signature scheme to sign each transaction in the Bitcoin network, and produce a hash-chain of length 3, i.e., a blockchain with length 3, as shown in Figure 1, with a simplified network structure and each block only consists of one transaction (so there is no need of Merkle tree authentication root here). The parameters will have real world security strength, i.e., 112-bit security.

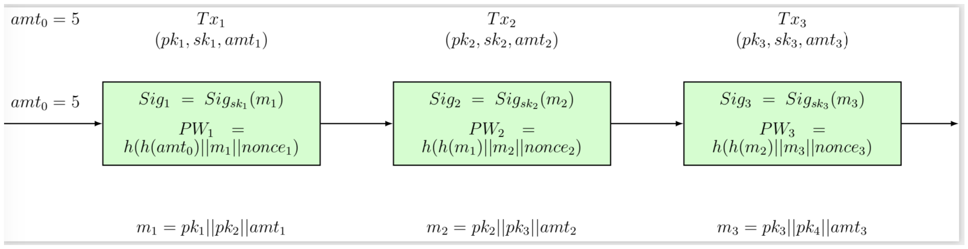
$amt_0 = 5$
Tx_1
(pk_1, sk_1, amt_1)
Tx_2
(pk_2, sk_2, amt_2)
Tx_3
(pk_3, sk_3, amt_3)
$amt_0 = 5$
$Sig_1 = Sig_{sk_1}(m_1)$
$PW_1 = h(h(amt_0)||m_1||nonce_1)$
$Sig_2 = Sig_{sk_2}(m_2)$
$PW_2 = h(h(m_1)||m_2||nonce_2)$
$Sig_3 = Sig_{sk_3}(m_3)$
$PW_3 = h(h(m_2)||m_3||nonce_3)$
$m_1 = pk_1||pk_2||amt_1$
$m_2 = pk_2||pk_3||amt_2$
$m_3 = pk_3||pk_4||amt_3$

Figure 1: A block chain of length 3, where $input_i, i = 1, 2, 3$ is omitted.

The parameters amt_0, amt_1, and amt_2 are three integer numbers with unit BTC, i.e.,

$$amt_0 = 5\ (BTCs), amt_1 = 4\ (BTCs), amt_2 = 3\ (BTCs).$$

You use 1 byte to represent one of those integers, i.e. (in hexadecimal) `05`, `04`, `03`, respectively.

In this project, you will act as both a user to send a payment and a miner to validate the transaction. You will act as the only miner in the system to validate the transaction, compute the proof-of-work for each transaction (nobody will compete with you to complete the computation) to link the blocks.

2 Crypto primitives

2.1 Digital Signature Algorithm (DSA)

In the lecture, we call it DSS. In this project, we adopt the specification of NIST FIPS 186-4. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`. NIST FIPS 186-4 uses a different sign in the signing equation. Below we list both formats:

$h(m) = xr + ks \bmod q$	in Lecture Notes
$h(m) = -xr + ks \bmod q$	in NIST FIPS 186-2 (2001), 186-4 (2013, EC-DSA added)
	186-5 (2019 draft, SHA3 added)

The verification is shown below

$u = h(m)s^{-1} \bmod q, v = -rs^{-1} \bmod q, w = g^u y^v$	in Lecture Notes
$u = h(m)s^{-1} \bmod q, v = rs^{-1} \bmod q, w = g^u y^v$	in NIST FIPS 186-*

For doing this project, you are requested to use the NIST format. In this way, you can check your results with the result computed from using some exiting library.

To work with DSA we first need to generate a set of domain parameters (will be given in ext subsection) satisfying the three criteria listed in Table 1.

Table 1: Parameter selection criteria

A1. p: a prime number lying between 1024 and 2048 bits
A2. q: a 224-bit prime factor of $p-1$
A3. g: an element $g \in \mathbb{F}_p$ with order q

2.2 A hash function

. The hash function you will use in the project is SHA3-224, for which c code and python code are provided. You can use any available code or library. Here is an online hash calculator that you can play around `https://www.pelock.com/products/hash-calculator`. an example computation of SHA3-224 is as follow, the input is a hexadecimal string.

```
    sha3_224("38363138363536383336")=
    0000000089cdf55ab2f8463c7438d555f8fd2c91c87ede3235c9bd18
```

Encoding convention for this project is the same as specified in NIST FIPS 186-4.

1. Bit string concatenation:

   (a) $m_i = pk_i || pk_{i+1} || amt_i$, $i = 1, 2, 3$, in this project, the bit length of m_i will always be 2048+2048+8= 4104. If $\text{len}(pk_i) < 2048$, we use m_1 as an example to explain how to pad 0 bits as follows. We denote $\ell_i = 2048 - len(pk_i), i = 1, 2$.

   $$m_1 = \underbrace{00...0}_{\ell_1} || pk_1 || \underbrace{00...0}_{\ell_2} || pk_2 || amt_1 \tag{1}$$

   where pk_1 and pk_2 are in their bit stream representations with the order from the most significant bit to the least significant bit.

   (b) The bit length of $h(*)||m_i||nonce_i$ will always be 224+4104+128=4456. If $\text{len}(nonce_i) < 128$, pad 0 bits as follows:

   $$h(*)||m_i|| \underbrace{00...0}_{t} ||nonce_i, t = 128 - \text{len}(nonce_i \tag{2}$$

2. Input to SHA3-224: a bit string whose length is a multiple of 8.

3. Convert the output of SHA3-224 to an integer in digital signature algorithm:

$$\begin{aligned} h(m_i) &= b_0, b_1, \cdots, b_{222}, b_{223} (\text{a bit string}) \\ &\leftrightarrow b_0 2^{223} + b_1 2^{222} + \cdots + b_{222} 2 + b_{223} \text{ (an integer)} \end{aligned}$$

2.3 Pseudo-random number generator (PRNG)

PRNG is implemented by SHA3-224 with an input message i for the ith signature and the 224 bits of the output of SHA3-224 is the random number k for ith signature.

3 DSA domain parameters

Each of you will use the same domain parameters p, q, g but has different private keys sk_i and public keys pk_i. You select your parameters with the file that matches your first two letters of your last name on LEARN.

```
"""
System parameters
"""

## length of p: 2048 bit
p=1615850420240242625399113195036680055148205339919365512280505165762 9
7060402526413293692294259272190069564737424769039787887283726796625612
6774959275647858465318737966807007747164023305326786794089976226985553
```

```
8496229272646267260199331950754561826958115323964167572312112683234368
7455831898884993636928081952280556386163355423282412423160031884910769
5302897851906422234787872466832362119565128334137884512840126331307093
2229612943555693076384094095923209888318983438374236756194589851339672
8731943262465539550908053983915501927699944385942431782427666188838032
561211221470832998214120910951662139914399589260156069735 43

## length of q: 224 bit
q=13479974306915323548855049186344013292925286365246579443817723220231
## length of g: 2047 bit
g=9891663101749060596110525648800442312262047621700008710332290803354 4
1973441523940037409297250576036855503397888372709087879878652786910610
2125568674515087767296064898813563305491697474743999164538645162593480
3406145834202726976694594399560579577756646531379694852178900779667311
4553543597150973233536157598924038645446910353512441488171918287556367
8656993578542852492841425689150799337502572709476677921927236216347614
5807006574858890795533331544043409550469603768594139262836640434472848
0845324408489345349308782555446303365930909965625721154544418491662738
79649173203959816263964230538954908382267559776340755836 0

## example of a pair of the private key and the public key for DSA
sk_1=40507772459734853117603686686565178027878053099049446926995119401 35

pk_1=g^{sk_1} mod{p
    }=16021750821812976439470928212502735284221334214433233771086754474 23
1825551129980536679574226328591119267911830788492584198650036143444110 0
4984902145848092829898125413492767792803354878092176707999742249466380 1
2295870437382871396123863492720568843604018161534643406614564043726751 0
1492680803185659232042879502820825606782020845674619554649222950789135 8
0861448063265031824569444828217846981889322971192244965625267511561592 0
7850470591952637284104064770126241354651607724842901672611232013081052 1
6563097662828437485365905816566633798684072289149535635146920770329628 3
380744068539298510146023010291677329404515292392021 71
```

4 Your tasks

Your tasks are to compute Sig_i and PW_i for $i = 1, 2$, as shown in Figure 1 where Sig_i serves as Transaction i $(=Tx_i)$ and PW_i is the Proof-of-Work (PW) for Transaction i. In this project, we omit the competing processing in the Bitcoin blockchain (i.e., you are the only miner in the system). Also each block only contains one transaction. Your tasks are specified as follows.

(a) (2 marks) Check that the system parameters p, q and g satisfy the three criteria in Table 1. You need to provide the factorization of $p - 1$, and explain your verification process. Explain why we only pick sk_1 as a number less than 224 bits. Compute $pk_i, i = 1, 2, 3$.

(b) (3 marks) Sign and validate transactions by DSA (i.e., DSS, the signing equation defined in NIST FIPS 186-4): implement a DSA module which enables the user can sign the transactions and a miner can verify the signed transaction. (Note that since we set all inputs being zero, so $m_i = Tx_i, i = 1, 2, 3.$)

(1) User 1: sign his transaction: $Sig_{sk_1}(m_1) = (r_1, s_1)$, i.e., compute the signature over m_1, k_1, $r_1 = (g^{k_1} \bmod p) \bmod q$, $k_1^{-1} \bmod q$ and $s_1 = k_1^{-1}(h(m_1) + xr \bmod q$.

(2) A miner: verify $Sig_{sk_1}(m_1)$, i.e., compute the values of u, v, w and verify whether $w = r$.

(3) User 2: sign his transaction $m_2 = Tx_2$ using the key pair (sk_2, pk_2), i.e., executing the same steps as User 1.

(c) (3 marks) Proof-of-Work (PW): Implement a module for a miner to compute a PW where SHA3-224 is used as a hash function h in PW_1 and PW_2 computations.

(1) Find pre-images of h such that

$$\begin{aligned} PW_1 = h(h(amt_0)||m_1||nonce_1) &= \underbrace{00\cdots0}_{k}\underbrace{**\cdots*}_{224-k} \\ PW_2 = h(h(m_1)||m_2||nonce_2) &= \underbrace{00\cdots0}_{k}\underbrace{**\cdots*}_{224-k} \end{aligned}$$

where $*$ means any value and $nonce_i, i = 1, 2$ are any 128-bit numbers. Here you use $k = 32$. You should vary a nonce in order to obtain a 32-consecutive leading zeros of SHA3-224 hash value. Your results on hash values PW_1 and PW_2 should be represented as hexadecimal numbers. (You need to establish c/c++ environment if you wish your computation can done shortly. Otherwise, you may need a few hours to have it done.)

(2) Determine the average number of trials which you need to get one PW in (1).

(3) (Optional) If you do this reasonably, i.e., you have a proper analysis for that, you will get one bonus mark. Increase the value of k until you cannot finish the computation. You may define a bound as the limit of your computation, say the program does not get a solution within 8-10 hours for computation, and you need to provide progressive computational complexities, i.e., for $k = 32, 33, \cdots, T$ where T is the limit of your computation with estimated complexity.

(d) (2 marks) Security analysis: Discuss why PW can prevent double spending in the Bitcoin network and identify two possible attacks on PW.

5 What do you need to submit?

1. Complete the `ECE 458 Project 1 Skeleton solution file.py`, name it as `yourfirstname-lastname.py`.

2. You need to submit a course project report which explains each step of your implementation of DSS in detail, and answers to those questions which are not included in `yourfirstname-lastname.py`, specifically,(c)-(2), (c)-(3)(Optional) and (d). For finding pre-image for a fixed hash format, you need to estimate its time complexity (i.e., how many exhaustive search for varying nonce in order to get such a hash value). Generally the report should be 4∼7 pages.

3. Submit a single zip file that contains above two files to the Dropbox on Learn. The name of your zip file should be `yourfirstname-lastname.zip`. You also need to submit your report to Crowdmark.

Some Remarks

- If you want to have some practice for big integer computations in order to compute public-keys and performing DSA, you may use the BigInteger class[1] in java.math package to do all the big integer operations involving in DSA. For simple computation, you may simply call any crypto library.

- For computing SHA3-224, you may use the c code and python code provided to you including test vectors or you can use any package as you wish. If you use Python/Java packages, you have to be careful regarding the encoding format, since the encoding specified in this project is bit string (provided code takes bit string as inputs), while hash function packages usually take character strings as input, then you should encode the input to utf-8 format before hash it. If you want find out the nonce in an efficient manner, it is recommend to use the c code. Otherwise, you may take few hours in Python/Java.

- Note that SHA3-224 has an input 1152-bit string for each absorbing process. But the code provided you or any other crypto library will automatically do the padding.

6 What to do next?

You may work along or team with one person. Even you are in a group of two, you still need to submit a separate report, which will be marked individually (it should be in the standard way that two reports are not similar).

[1]see `http://docs.oracle.com/javase/1.5.0/docs/api/java/math/BigInteger.html`

7 Due Date

The report is due at

June 26, 11:59am ET, 2020.

You need to submit i) Project Report, ii) Code, and upload them to the Crowdmark.

Notes.

(1) Note that you earn 10 marks based on how well you answer the questions and the correctness of the computation, and how good of your solutions in terms of complexity for solving that.

(2) If there are any questions or something is confusing, please ask me. I reserve the right to provide updates and clarifications as needed.