Implementing Cryptographic Primitives for BlockChain

Cryptography CS 411 & CS 507 Term Project for Fall 2024

E. Savaş
Computer Science & Engineering
Sabancı University
İstanbul

Abstract

You are required to develop essential building blocks of cryptocurrency using block chains.

1 Introduction

The project has three phases:

- Developing software for digital signature and blockchain transactions
- Developing software for proof of work
- Developing software for other building blocks and integration

More information about the phases are given in the subsequent sections.

2 Phase I: Developing software for digital signature and Blockchain Transactions

In this phase of the project, you are required to upload two files: "DS.py" and "Tx.py". We will be able to test your codes using "PhaseI_Test.py'. If your software cannot be tested by "PhaseI_Test.py' as it is, you will get no credit.

2.1 Developing software for digital signature

Here, you will develop a Python module "DS.py," that includes functions for signing any given message and verifying the signature. For digital signature (DS) you will use an algorithm, which consists of four functions as follows:

- **Public parameter generation:** Two prime numbers p and q are generated with $q|p-1$, where q and p are 224-bit and 2048-bit integers, respectively. The generator g generates a subgroup of Z_p^* with q elements. Naturally, $g^q \equiv 1 \bmod p$. Note that in your system q, p, and g are *public parameters* shared by all users, who have different secret/public key pairs. Refer to the slide (with the title "DSA Setup" in chapter 9 for an efficient method for parameter generation).

- **Key generation:** A user picks a random secret key $0 < \alpha < q-1$ and computes the public key $\beta = g^{\alpha} \bmod p$.
- **Signature generation:** Let m be an arbitrary length message. The signature is computed as follows:
  1. $k \leftarrow \mathbb{Z}_q$, (i.e., k is a random integer in $[1, q-2]$).
  2. $r = g^k \pmod p$
  3. $h = \text{SHA3\_256}(m||r) \pmod q$
  4. $s = k - \alpha \cdot h \pmod q$
  5. The signature for m is the tuple (s, h).
- **Signature verification:** Let m be a message and the tuple (s, h) is a signature for m. The verification proceeds as follows:
  - $v = g^s \beta^h \pmod p$
  - $u = \text{SHA3\_256}(m||v) \pmod q$
  - Accept the signature only if $u = h$
  - Reject it otherwise.

Note that the signature generation and verification of this DS are different than those discussed in the lecture.

You are required to develop Python software that implements those four functions; namely SETUP for public parameter generation, KEY GENERATION, SIGNATURE GENERATION and SIGNATURE VERIFICATION. You are required to test your software using the test routines in "PhaseI_Test.py" provided in the assignment package.

In this part of "PhaseI_Test.py", there are four basic test functions:

1. CHECKDSPARAMS(q, p, g) takes your public parameters (q, p, g) and checks if they are correct. It returns 0 if they are. Otherwise, it returns a code that indicates the problem.
2. CHECKKEYS(q, p, g, α, β) takes your public parameters (q, p, g) and key pair (α, β) and checks if the key pair is correct. It returns 0 if they are; otherwise it returns -1.
3. CHECKSIGNATURE(q, p, g, α, β) takes your public parameters (q, p, g), key pair (α, β) and generates a signature for a random message and verifies the signature. It returns 0 if the signatures verifies; otherwise it returns -1.
4. CHECKTESTSIGNATURE() reads the file "TestSet.txt" (provided in the assignment package), which contains public parameters, a public key, and 10 randomly chosen messages and their signatures. The test code reads them and runs signature verification function. The test code returns 0 if all signatures verify; otherwise it returns -1.

2.2 Generating Random Transactions

Here, you will develop a Python module "Tx.py" that includes functions for generating a random Blockchain transaction. A transaction contains information of a payment (transaction) from the *payer* to the *payee* and is in the following format:

```
*** Bitcoin transaction ***
Signature (s):
Signature (h):
Serial number:
Amount:
Payee public key (beta):
Payer public key (beta):
```

Explanations of these fields are as follows

`Signature (s):` Signature (s part) of the transaction by Payer
`Signature (h):` Signature (h part) of the transaction by Payer
`Serial Number:` is a uniformly randomly generated 128-bit integer
`Amount:` is the amount in *Satoshi* being transferred in range [1, 1000000]
`Payee public key (beta):` is the public key of the person receiving the payment
`Payer public key (beta):` is the public key of the person making the payment

Note that the payer and payee are identified by their public keys in the transaction. In the transaction, the `Serial Number`, `Amount`, `Payee public key (beta)`, and `Payer public key (beta)` fields are signed by the private key of the payer. Therefore, the transaction can be verified by the payer's public key.

In this part of "PhaseI_Test.py", there are two test functions:

1. CheckTransaction(q, p, g) creates a random transaction and verifies its signature. For this, you will develop a function named "gen_random_tx(q, p, g)" in "Tx.py".

2. CheckBlockofTransactions() reads sample transactions given in file "transactions.txt" (also provided in the assignment package) and verifies the signatures of all transactions using your verification function in "DS.py"; namely "DS.SignVer".

3 Appendix I: Timeline & Deliverables & Weight & Policies etc.

Project Phases	Deliverables	Due Date	Weight
Project announcement		12/12/2024	
First Phase	Files: DS.py and Tx.py	19/12/2024	30%
Second Phase	Files: TBA	26/12/2024	30%
Third Phase	Files: TBA	TBA	40%

3.1 Policies

- You may work in groups of two.
- Submit all deliverables in the zip file "cs411_507_tp1_yourname.zip".
- You may be asked to demonstrate a project phase to a TA or the instructor.
- In every phase, we will provide you with a validation software in Python language that can be used to check your implementation for correctness. We will also use it to check your implementation. If your implementation in a project phase fails to pass the validation, you will get no credit for that phase.
- Your codes will be checked for their similarity to other students' codes; and if the similarity score exceeds a certain threshold you will not get any credit for your work.