CS 5000: F24: Theory of Computability
Assignment 10

Vladimir Kulyukin
Department of Computer Science
Utah State University

November 24, 2024

1 Learning Objectives

1. Primitive Roots of Primes
2. Digital Cash
3. Digital Cash Systems
4. Double Spending Fraud Detection

Introduction

In this assignent, we will work a simple version of the digital cash system outlined in references [1] and [2] in Lectures 24 and 24. For your convenience, I included the Lecture 25 PDF in the zip. It has all the formulas needed to complete this assignment. I wrote a detailed, step-by-step unit test `test_digcash_system` in `digcash_uts.py` to help you familiarize yourself with the system and to assist you with your implementations in Problems 1 and 2 below.

Please review the Lecture 25 PDF and/or your class notes to become comfortable with the concepts of digital cash and its properties, digital coin creation, coin spending and depositting, double spending, and digital fraud control. The amazing thing about these concepts is that they can all be implemented with equivalence classes modulo some prime numbers, primitive roots of primive numbers, and discrete logs.

The zip also contains the files

1. `ntutils.py`;
2. `pubrepo.py`;
3. `auth.py`;
4. `spender.py`;
5. `bank.py`;
6. `merchant.py`;
7. `vendor.py`.

The file `ntutils.py` contains a few of my implementations of the number-theoretic functions that I used in this assignment (e.g., Extended Euclid, identifying primitive roots of a prime number, etc.) The file `pubrepo.py` implements a public repository class. The particpants use an object of this class to publish and get all publicly available information. The file `auth.py` implements a regulatory authority. The files `bank.py`, `merchant.py`, and `vendor.py`. implement the bank and two merchants.

To start working on the assignment you may want to run `digcash_uts.py` as is. I commented out the last part of the unit test related to Problems 1 and 2 below. When you do, you should see something like this at first.

```
*** Authority initialization done...
p  == 227
q  == 113
g  == 169
g1 == 108
g2 == 112
```

```
*** Bank initialization done...
h  == 30
h1 == 214
h2 == 104
```

The above output is produced by the followig Python code.

```
## 1. Initialize public repo, authority
pbr = pubrepo()
aut = auth()
aut.init_p_q_g_g1_g2_H_H0(pbr)

## 2. Bank initialization
bnk = bank()
bnk.create_h_h1_h2(pbr)
```

We create a public repository `pbr` and a regulatory authority `aut`. Then `aut` calls its method `init_p_q_g_g1_g2_H_H0(pbr)` that saves in the repo the numbers p, q, g_1, g_2 and two cryptographic hash functions H and H_0 (cf. Slides 13 – 16 in Lecture 25). Then the bank is initialized whereby the numbers h, h_1, and h_2 are published in the public repo (cf. Slides 16, 17 in Lecture 25).

You will then see something like this.

```
Spender 10 requested Bank to create a spender account.
Bank created the spender account 10 with 10 credit units
Bank computed z_prime == 189
Spender 10 received z_prime 189 from Bank.
Merchant 29 requested Bank to create a merchant account.
Bank created the merchant account 29 with 0 credit units
Spender 10 requested gw and beta from Bank.
Bank computed w == 15
Bank computed gw == 25 and beta == 104
Spender created secret random 5-tuple (20, 6, 4, 9, 12)
Spender 10 computed c == 71.
Spender 10 requested c1 Bank.
Bank computed c1 == 8
Spender 10 computed r == 84.
Spender 10 received the unspent coin (116, 219, 190, 69, 225, 84).
```

The above output is produced by the following Python code in `digcash_uts.py`.

```
## 3. Spender initialization
spr = spender()
spr.create_bank_account(bnk, pbr)

## 4. Merchankt initialization
mrt = merchant()
mrt.create_bank_account(bnk)

## 5. merchant requests a coin from the bank.
spr.request_coin(bnk, pbr)
```

In other words, the spender is created. The spender opens a spender account with the bank. Then the first merchant `mrt` is created and the merchant also opens a merchant account with the bank. Then the creation of a new coin is carried by at the spender's request when the spender object calls `request_coin()`. These methods implement the coin creation logic on Slides 19 – 26.

Finally, you should see something like this.

```
Spender 10 requested Merchant 29 to compute d
Merchant 29 computed d == 73.
Spender 10 requested Merchant 29 to accept coin (116, 219, 190, 69, 225, 84) with signature (2, 108, 73)
Merchant 29 found coin (116, 219, 190, 69, 225, 84) with signature (2, 108, 73) acceptable.
Merchant 29 requested Bank to deposit coin (116, 219, 190, 69, 225, 84) with signature (2, 108, 73)
Bank deposited coin (116, 219, 190, 69, 225, 84) with signature (2, 108, 73) to merchant account 29
Merchant 31 requested Bank to create a merchant account.
Bank created the merchant account 31 with 0 credit units
```

The above output is produced by the following Python code in `digcash_uts.py`.

```
## 6. spender spends a newly issued coin.
spr.spend_unspent_coin(mrt, pbr)
## 7. merchant deposits the coin.
mrt.deposit_coin(bnk, pbr)
## 8. spender's balance is now 10 - 1.
assert spr.get_balance(bnk) == 9
## 9 the merchant's balance is now 0 + 1.
assert mrt.get_balance(bnk) == 1
## 10. vendor is initialized
vdr = vendor()
vdr.create_bank_account(bnk)
```

Steps 6 – 9 implement the coin spending and depositting logic on Slides 27 – 31 of Lecture 25. The spender spends the coin at the merchant and the merchant deposits the coin. In Step 10, we initialize a nother merchant `vdr` where the spender will attempt to double spend this coin.

Problem 1 (2 points)

Implement the method `double_spend_coin()` in `spender.py`. This method should reflect the logic in slides 32, 33, and 34 of Lecture 25 where the spender attempts to double spend a coin with the vendor that the spender previously spent with the merchant. The merchant and the vendor are two different merchants.

Problem 2 (3 points)

Implement the method

```
double_spending_faud_control(self, merchant_id, coin, coin_signature, pbr)
```

in `bank.py`. This method is called by the bank when from the method

```
deposit_coin_for_merchant(self, merchant_id, coin, coin_signature, pbr)
```

when the bank detects that the coin has been deposited before.

My Unit Test Output

The last part of the method `test_digcash_system(self)` in `digcash_uts.py` looks as follows.

```
### spender attempts to double spend a coin at the vendor vndr.
spr.double_spend_coin(vdr, pbr)
### vendor deposits it at the bank.
double_spender_id = vdr.deposit_coin(bnk, pbr)
assert double_spender_id == spr.reveal_secret_id()
### spender's balance is still 9 because the bank
### did not withdraw a unit from the spender's account.
assert spr.get_balance(bnk) == 9
### the vendor's balance is still 0, because
### the double spent coint was not deposited.
assert vdr.get_balance(bnk) == 0
```

When I uncomment it and run it with my implementations of Problems 1 and 2, my input looks as follows.

```
Spender 182 attempts to double spend coin = (78, 99, 186, 34, 73, 50)
Merchant 31 computed d == 12.
d_prime = 12
Merchant 31 found coin (78, 99, 186, 34, 73, 50) with signature (79, 82, 12) acceptable.
Merchant 31 requested Bank to deposit coin (78, 99, 186, 34, 73, 50) with signature (79, 82, 12)
Bank initiated double spending fraud control
double spender id = 8
```

When I run it again, I see this.

```
Spender 81 attempts to double spend coin = (33, 133, 48, 161, 40, 66)
Merchant 31 computed d == 62.
d_prime = 62
Merchant 31 found coin (33, 133, 48, 161, 40, 66) with signature (75, 102, 62) acceptable.
Merchant 31 requested Bank to deposit coin (33, 133, 48, 161, 40, 66) with signature (75, 102, 62)
Bank initiated double spending fraud control
double spender id = 11
```

The important point that in all cases of double spending, as in both cases above, the randomly initialized ID of the double spender is discovered by the bank and no credit is added to the vendor `vdr`'s bank account.

What to Submit?

Save your implementations in `bank.py` and `spender.py`, zip these two files and the other files of the digital cash system into hw10.zip and upload the zip in Canvas.

Enjoy Digital Cash!