simple_blockchain_cryptocurrency

March 24, 2022

1 Simple Cryptocurrency in Ruby

This is an attempt to create a proof of concept based on the Bitcoin whitepaper by Satoshi Nakamoto.

1.1 References

- Bitcoin whitepaper
- Dumbcoin by julienr
- blockchain.rb - Build your own blockchain from scratch in 20 lines of ruby by openblockchains
- Bitcoin wiki

1.2 Disclaimer

There will most likely be A LOT of things wrong & insecure in this implementation.

[1]:

```
require 'openssl'
require 'minitest'
require 'benchmark'
require 'base64'

# Enable assertions outside of Minitest::Test
include Minitest::Assertions
Object.class_eval do
  attr_accessor :assertions
end
self.assertions = 0

# Override IRuby::Kernal to hide backtrace from display
# https://github.com/SciRuby/iruby/blob/master/lib/iruby/kernel.rb
IRuby::Kernel.class_eval do
  def error_message(e)
    { status: :error,
        ename: e.class.to_s,
        evalue: e.message,
        traceback: ["\e[31m#{e.class}\e[0m: #{e.message}"],
        execution_count: @execution_count }
    end
end
```

```
[1]: :error_message
```

1.3 The Wallet

Here we're creating a simple coin wallet with a set of Public/Private keys using a 512 bit RSA for Public Key encryption.

We'll simply use the an URL safe base64 encoded Public Key as the wallet address.

```
[2]: class Wallet
       attr_reader :address

       def initialize
         @key = OpenSSL::PKey::RSA.new(512)  # obviously insecure
         @address = Base64.urlsafe_encode64(@key.public_key.to_der)
       end

       def sign(message)
         digest = OpenSSL::Digest::SHA256.new(message)
         sig = @key.sign(digest, message)
         Base64.urlsafe_encode64(sig)
       end
     end

     w = Wallet.new
```

```
[2]: #<#<Class:0x00000001089b44d0>::Wallet:0x000000010898f4c8
     @key=#<OpenSSL::PKey::RSA:0x000000010898f248 oid=rsaEncryption>, @address="MFwwD
     QYJKoZIhvcNAQEBBQADSwAwSAJBAKtjwbYyAQ0_meUD2TdcOTkBCowxe2i7fjLZg4v41mDZAl3QE64rH
     I7ElcnXlfImZFNVDcCPsRokJQ5F_X0tIsMCAwEAAQ==">
```

And then we want to have some helper methods to verify the encryption.

```
[3]: module PKeyVerifier
       def verify(pkey_base64, signature_base64, message)
         pkey = Base64.urlsafe_decode64(pkey_base64)
         signature = Base64.urlsafe_decode64(signature_base64)

         key =  OpenSSL::PKey::RSA.new(pkey)
         digest = OpenSSL::Digest::SHA256.new(message)
         key.verify(digest, signature, message)
       end
     end

     class TestVerifier; include PKeyVerifier; end
     pkey_verifier = TestVerifier.new
```

```
[3]: #<#<Class:0x00000001089b44d0>::TestVerifier:0x000000010897e308>
```

[4]:
```
signature = w.sign('some transaction')
assert pkey_verifier.verify(w.address, signature, 'some transaction')
```

[4]:
```
true
```

[5]:
```
assert pkey_verifier.verify(w.address, signature, 'some altered transaction'),␣
 ↪'Message altered'
```

```
Minitest::Assertion: Message altered
/Users/jimmy/.rubies/ruby-3.1.0/lib/ruby/gems/3.1.0/gems/minitest-5.15.0/lib/
 ↪minitest/assertions.rb:183:in `assert'
(irb):in `<top (required)>'
```

[6]:
```
another_wallet = Wallet.new
assert pkey_verifier.verify(another_wallet.address, signature, 'some␣
 ↪transaction'), 'Message not signed by same wallet'
```

```
Minitest::Assertion: Message not signed by same wallet
/Users/jimmy/.rubies/ruby-3.1.0/lib/ruby/gems/3.1.0/gems/minitest-5.15.0/lib/
 ↪minitest/assertions.rb:183:in `assert'
(irb):1:in `<top (required)>'
```

1.4 The Transaction

A transaction would need to have a `source` wallet, and an instructions of amounts to send to a `destination` wallets.

[7]:
```
class Transaction
  attr_reader :source_wallet, :instruction

  def initialize(source_wallet, instructions)
    @source_wallet = source_wallet
    @instructions = instructions
  end
end
```

[7]:
```
:initialize
```

1.5 The Transaction Instruction

We'll define a class for Transaction Instructions. Each transaction instruction will define the target wallet & the amount to send.

[8]:
```
class Transaction::Instruction
  attr_reader :target_wallet, :amount
  def initialize(target_wallet, amount)
    @target_wallet = target_wallet
    @amount = amount.to_f
  end

  def to_hash
    {
      target:  @target_wallet,
      amount: @amount
    }
  end
end
```

[8]:

```
:to_hash
```

So a typical transaction will look more of less like so

[9]:

```
w1 = Wallet.new
w2 = Wallet.new

t1 = Transaction.new(w1, Transaction::Instruction.new(w2, 25.0))

nil  # hide noisy output
```

1.6 Transactions List

In a ledger system, all transactions will be a chronologically ordered list. In our case, we'll just add them into a global variable as soon as each transaction is defined.

[10]:

```
class Transaction
  attr_reader :source_wallet, :instruction

  def initialize(source_wallet, instruction)
    @source_wallet = source_wallet
    @instruction = instruction

    append_to_transactions_list
  end

  def to_hash
    {
      source: @source_wallet,
      instruction: @instruction.to_hash
    }
  end

  def append_to_transactions_list
```

```
    $transaction_list = [] if $transaction_list.nil?
    $transaction_list << self
  end
end
```

[10]:
```
:append_to_transactions_list
```

Now, according to the Bitcoin whitepaper, each transaction owner should digitally sign a hash of previous transaction with public key of next owner, so that a payee can verify the chain of ownership.

To achieve that, we'll need to find the last transaction of the `source` wallet, and use it as create a hash & signature. Time to tweak `Wallet`.

[11]:
```
class Wallet
  def sign_instruction(instruction)
    previous_hash = last_wallet_transaction && last_wallet_transaction.hash
    hash_payload  = {
      target_address: instruction.target_wallet,
      previous_hash: previous_hash,
      instruction: instruction.to_hash
    }
    hash = OpenSSL::Digest::SHA256.new(hash_payload.to_s)
    signature = sign(hash.to_s)

    [hash, signature, previous_hash]
  end

  def last_wallet_transaction
    $transaction_list.select {|t| t.source_wallet == self.address || t.
↪instruction.target_wallet == self.address }.last
  end
end
```

[11]:
```
:last_wallet_transaction
```

[12]:
```
class Transaction
  attr_reader :hash, :signature, :previous_hash

  def initialize(source_wallet, instruction, hash, signature, previous_hash)
    @source_wallet = source_wallet
    @instruction = instruction
    @hash = hash
    @signature = signature
    @previous_hash = previous_hash

    append_to_transactions_list
  end
  def to_hash
    {
      owner: @source_wallet,
      instruction: @instruction.to_hash,
      hash: @hash,
      signature: @signature,
      previous_hash: @previous_hash,
    }
  end

end
```

[12]:
```
:to_hash
```

[13]:
```
$transaction_list = []

w1 = Wallet.new
w2 = Wallet.new
w3 = Wallet.new
```

[13]:
```
#<#<Class:0x00000001089b44d0>::Wallet:0x0000000108b8ab38
@key=#<OpenSSL::PKey::RSA:0x0000000108b8a840 oid=rsaEncryption>, @address="MFwwD
QYJKoZIhvcNAQEBBQADSwAwSAJBALY0ztVrlBkBVQDprfoRxCdBhRoA50effTGU_5fXezYgtW04ZmeMG
XWQkww1yP18ediKm066I0zcvR4z9lmAFH8CAwEAAQ==">
```

We'll pretend that `$transaction_list` is being transferred through the network as the public ledger. Since `Transaction` data will be public, all data stored within should be kosher for public viewing (i.e. no private keys).

[14]:
```
t1 = Transaction.new(w1.address, i = Transaction::Instruction.new(w2.address,␣
  ↪25.0), *w1.sign_instruction(i))
t2 = Transaction.new(w1.address, i = Transaction::Instruction.new(w2.address,␣
  ↪10.0), *w1.sign_instruction(i))
t3 = Transaction.new(w2.address, i = Transaction::Instruction.new(w3.address, 5.
  ↪0), *w2.sign_instruction(i))
t4 = Transaction.new(w2.address, i = Transaction::Instruction.new(w3.address, 2.
  ↪0), *w2.sign_instruction(i))

nil  # hide noisy output
```

Let's make a simple inspection function for all transctions.

[15]:
```
def inspect_all_transactions
  $transaction_list.map(&:to_hash).each_with_index do |t, index|
    source_address = t[:owner][33..42]  # for display purposes
    target_address = t[:instruction][:target][33..42]  # for display purposes
```

```ruby
    puts "Transaction ##{index+1}: \n"
    puts "From: \t\t\t#{source_address}"
    puts "To: \t\t\t#{target_address}"
    puts "Amount: \t\t#{t[:instruction][:amount]}"
    puts "Hash: \t\t\t#{t[:hash]}"
    puts "Signature: \t\t#{t[:signature]}"
    puts "Previous Hash: \t#{t[:previous_hash]}"
    puts "----------------------\n"
  end
end

inspect_all_transactions

nil  # hide noisy output
```

```
Transaction #1:
From:                   OWPbt64KV6
To:                     N5DMA0sI1U
Amount:                 25.0
Hash:
daff11991f304ad01af657c4266b0ab4c17e679a7aa962373f87c0fec521cd5e
Signature:              wR_vA9_ceNoiMQJfMaNd3ZdfMZG_g8e52t9GhuPV79gwTx0t298ArfPs
AEYsc7aFI8X5nM5hyAiwUB5ARKgFUQ==
Previous Hash:
----------------------
Transaction #2:
From:                   OWPbt64KV6
To:                     N5DMA0sI1U
Amount:                 10.0
Hash:
ae35093bbd7e9ab17430ab8ea2f365acca073321fe5a3b1c7ee3d08efe967a23
Signature:              HomMvDybsh3vFV-
eKiyElZlQBn4Oox9aUZkFYVZgxWO1-LuLZN7CkoY9AKF6tboVbPIqimSso-Pp82qSqkHedw==
Previous Hash:  daff11991f304ad01af657c4266b0ab4c17e679a7aa962373f87c0fec521cd5e
----------------------
Transaction #3:
From:                   N5DMA0sI1U
To:                     LY0ztVrlBk
Amount:                 5.0
Hash:
65ded7a81e845018836402c0677a61040d2d0dd9394a03ec68fd67c0267568e8
Signature:              cWkwc1UCE4V0fd5tDhHQWWoBfgGvRSvLT1DUBfxB2hA7Cvk9VOOokpBp
Ic2X-_qFxEAkjQNu_R_Gh6cnI6aASw==
Previous Hash:  ae35093bbd7e9ab17430ab8ea2f365acca073321fe5a3b1c7ee3d08efe967a23
----------------------
Transaction #4:
```

```
From:                   N5DMA0sI1U
To:                     LY0ztVrlBk
Amount:                 2.0
Hash:
869355ef539eb385f48030d030f9e55f78a5519c797484367998c581be66db74
Signature:              tbQe-oRrvHXNrY1jj9WjHd_X6YV0LtjthjlrNS4JiqrEVeA3mmXO0Hc1
z6gCbg9FxrMRwHg8b-42XxedknPPQA==
Previous Hash:  65ded7a81e845018836402c0677a61040d2d0dd9394a03ec68fd67c0267568e8
----------------------
```

So as the payees, the data integrity can be verified through the signature chains.

[16]:

```ruby
class Transaction
  include PKeyVerifier

  def self.find_transaction(hash)
    $transaction_list.find { |t| t.hash == hash }
  end

  def verify_self_and_ancestors
    ancestor_valid = true

    if @previous_hash
      # check validity of previous transaction, and fail if transaction is missing
      ancestor_valid = (previous_transaction && previous_transaction.verify_self_and_ancestors ) || false
    end

    ancestor_valid && verify_transaction
  end

  def verify_transaction
    verify(@source_wallet, @signature, recalculate_hash.to_s)
  end

  def recalculate_hash
    hash_payload = {
      target_address: @instruction.target_wallet,
      previous_hash: previous_transaction && previous_transaction.hash,
      instruction: @instruction.to_hash
    }
    OpenSSL::Digest::SHA256.new(hash_payload.to_s)
  end

  def previous_transaction
    self.class.find_transaction(@previous_hash)
  end
```

```
end

assert t1.verify_self_and_ancestors
```

[16]:

```
true
```

Changes to the transaction instructions or target wallets are quickly detectable throughout the chain.

[17]:

```
hacker_wallet = Wallet.new

t2.instance_variable_set(:@instruction, Transaction::Instruction.
 ↪new(hacker_wallet.address, 10.0))

assert t2.verify_self_and_ancestors, 'Problem with ancestral transaction'
```

```
Minitest::Assertion: Problem with ancestral transaction
/Users/jimmy/.rubies/ruby-3.1.0/lib/ruby/gems/3.1.0/gems/minitest-5.15.0/lib/
 ↪minitest/assertions.rb:183:in `assert'
(irb):4:in `<top (required)>'
```

1.7 Blocks

Now that we have a way to create and verify transactions, it's time to build the block mechanism that would mine and distribute the transactions throughout the network.

1.7.1 Simple Miner

The miner takes in a message & finds a `nonce` that satisfies a difficulty level

[18]:

```
module Miner
  def mine(message, difficulty_level = 2) # 2 leading zeros
    nonce = 0
    loop do
      hash = OpenSSL::Digest::SHA256.new(message + nonce.to_s).to_s
      return [hash, nonce] if hash.start_with? '0' * difficulty_level
      nonce += 1
    end
  end
end
```

[18]:

```
:mine
```

Time required to mine increases exponentially when difficulty increases. Difficulty would increase according to the moving average of time for each block to be added to the blockchain.

[19]:

```
class TestMiner; include Miner; end

puts Benchmark.measure { puts TestMiner.new.mine 'bar' }
puts Benchmark.measure { puts TestMiner.new.mine 'bar', 4 }
# puts Benchmark.measure { puts TestMiner.new.mine 'bar', 6 }
```

```
00b894f575e9311a064511b37bc5bfd57365980ba9157aaf69afec3fecf8178a
5
  0.000059   0.000006   0.000065 (  0.000061)
000065a302ba3d5ff98dd372b82ab1837fbbbae10c4576a9b17d778dfda89955
61125
  0.070183   0.002840   0.073023 (  0.073297)
```

So now let's build a `Block`. A `Block` needs to know some info of the last `Block`, and also a list of `transactions` to mine.

[20]:

```
$blockchain = []
$mining_difficulty = 2

class Block
  include Miner
  attr_accessor :nonce, :hash, :previous_block_hash

  def initialize(transactions)
    @transactions = transactions
    @previous_block = $blockchain.last
    @previous_block_hash = @previous_block && @previous_block.hash || ''
    mine_transactions
    add_to_blockchain
  end

  def mine_transactions
    transactions_hash = @transactions.map(&:to_hash).join

    message = previous_block_hash + transactions_hash
    @hash, @nonce = mine(message, $mining_difficulty)
  end

  def add_to_blockchain
    $blockchain << self
  end
end
```

[20]:

```
:add_to_blockchain
```

Helper function to inspect the blockchain

[21]:

```
def inspect_all_blocks
  $blockchain.each_with_index do |b, index|
    puts "Block ##{index+1}: \n"
    puts "Previous Hash: \t#{b.previous_block_hash}"
    puts "Nonce: \t\t\t#{b.nonce}"
    puts "Hash: \t\t\t#{b.hash}"
    puts "----------------------\n"
  end
end
```

[21]:

```
:inspect_all_blocks
```

[22]:

```
$transaction_list = []

t1 = Transaction.new(w1.address, i = Transaction::Instruction.new(w2.address,␣
 ↪25.0), *w1.sign_instruction(i))
t2 = Transaction.new(w1.address, i = Transaction::Instruction.new(w2.address,␣
 ↪10.0), *w1.sign_instruction(i))
t3 = Transaction.new(w2.address, i = Transaction::Instruction.new(w3.address, 5.
 ↪0), *w2.sign_instruction(i))
t4 = Transaction.new(w2.address, i = Transaction::Instruction.new(w3.address, 2.
 ↪0), *w2.sign_instruction(i))

inspect_all_transactions

nil # nide noisy output
```

```
Transaction #1:
From:                   OWPbt64KV6
To:                     N5DMA0sI1U
Amount:                 25.0
Hash:
daff11991f304ad01af657c4266b0ab4c17e679a7aa962373f87c0fec521cd5e
Signature:              wR_vA9_ceNoiMQJfMaNd3ZdfMZG_g8e52t9GhuPV79gwTx0t298ArfPs
AEYsc7aFI8X5nM5hyAiwUB5ARKgFUQ==
Previous Hash:
----------------------
Transaction #2:
From:                   OWPbt64KV6
To:                     N5DMA0sI1U
Amount:                 10.0
Hash:
ae35093bbd7e9ab17430ab8ea2f365acca073321fe5a3b1c7ee3d08efe967a23
Signature:              HomMvDybsh3vFV-
eKiyElZlQBn4Oox9aUZkFYVZgxWO1-LuLZN7CkoY9AKF6tboVbPIqimSso-Pp82qSqkHedw==
Previous Hash:  daff11991f304ad01af657c4266b0ab4c17e679a7aa962373f87c0fec521cd5e
----------------------
```

```
Transaction #3:
From:                   N5DMA0sI1U
To:                     LY0ztVrlBk
Amount:                 5.0
Hash:
65ded7a81e845018836402c0677a61040d2d0dd9394a03ec68fd67c0267568e8
Signature:              cWkwc1UCE4V0fd5tDhHQWWoBfgGvRSvLT1DUBfxB2hA7Cvk9VOOokpBp
Ic2X-_qFxEAkjQNu_R_Gh6cnI6aASw==
Previous Hash:  ae35093bbd7e9ab17430ab8ea2f365acca073321fe5a3b1c7ee3d08efe967a23
----------------------
Transaction #4:
From:                   N5DMA0sI1U
To:                     LY0ztVrlBk
Amount:                 2.0
Hash:
869355ef539eb385f48030d030f9e55f78a5519c797484367998c581be66db74
Signature:              tbQe-oRrvHXNrY1jj9WjHd_X6YV0LtjthjlrNS4JiqrEVeA3mmXOOHc1
z6gCbg9FxrMRwHg8b-42XxedknPPQA==
Previous Hash:  65ded7a81e845018836402c0677a61040d2d0dd9394a03ec68fd67c0267568e8
----------------------
```

[23]:

```
$blockchain = []

Block.new([t1, t2])
Block.new([t3, t4])

inspect_all_blocks

nil # nide noisy output
```

```
Block #1:
Previous Hash:
Nonce:                  183
Hash:
0090bdd07696ed5a16b351e92fab004e1ef5f73dcec9fe3480796247e08dfdaa
----------------------
Block #2:
Previous Hash:  0090bdd07696ed5a16b351e92fab004e1ef5f73dcec9fe3480796247e08dfdaa
Nonce:                  93
Hash:
00589b1bf9013cc2e37ff6469ceae1310eb297345a0c5d2f16f820a9f0e24449
----------------------
```

1.8 Tampering with the transactions

1.8.1 Reversing transactions

It is still possible to alter the transactions within the ledger if those Transactions originated from the attacker's wallet, since they can re-sign the hashes, allowing them to reverse any transactions made by them.

[24]:

```
$transaction_list = []

t1 = Transaction.new(hacker_wallet.address, i = Transaction::Instruction.new(w2.address, 25.0), *hacker_wallet.sign_instruction(i))
t2 = Transaction.new(hacker_wallet.address, i = Transaction::Instruction.new(w2.address, 10.0), *hacker_wallet.sign_instruction(i))

nil  # hide noisy output
```

A hacker can easily manipulate the stored hash by resigning all their transactions.

[25]:

```
def hacker_wallet.l33t_sign_instruction(instruction, current_transaction)
  # getting the previous transaction instead of the last one on the chain like normal
  previous_hash = current_transaction.previous_hash

  hash_payload  = {
    target_address: instruction.target_wallet,
    previous_hash: previous_hash,
    instruction: instruction.to_hash
  }
  hash = OpenSSL::Digest::SHA256.new(hash_payload.to_s)
  signature = sign(hash.to_s)

  [hash, signature, previous_hash]
end

# Reversing 25.0 transfer to w2
i = Transaction::Instruction.new(t1.instruction.target_wallet, 0.0)
hash, signature, previous_hash = hacker_wallet.l33t_sign_instruction(i, t1)
t1.instance_variable_set(:@instruction, i)
t1.instance_variable_set(:@hash, hash)
t1.instance_variable_set(:@signature, signature)

# Update t2's previous_hash
t2.instance_variable_set(:@previous_hash, hash)

# Recalculate hash for all later transactions
i = Transaction::Instruction.new(t2.instruction.target_wallet, t2.instruction.amount)
hash, signature, previous_hash = hacker_wallet.l33t_sign_instruction(i, t2)
t2.instance_variable_set(:@hash, hash)
t2.instance_variable_set(:@signature, signature)

nil  # hide noisy output
```

[26]:

```
# Passes tests
assert t1.verify_self_and_ancestors
assert t2.verify_self_and_ancestors
```

[26]:

```
true
```

[27]:

```
inspect_all_transactions

nil  # hide noisy output
```

```
Transaction #1:
From:                        LsgfGzhFON
To:                          N5DMA0sI1U
Amount:                      0.0
Hash:
8f64da547860e8dc7ffc08f298bc15d013025372df066a5d484c5232221bc108
Signature:                   nJ4pT-0UfnY6FGE5pBYNmRwQuzfZkNU8rK0ub5nFfXjb3dPjm4bL_4x-
GzYjnn4Cspxzv5N3wIOD8UBwKvbeWA==
Previous Hash:
----------------------
Transaction #2:
From:                        LsgfGzhFON
To:                          N5DMA0sI1U
Amount:                      10.0
Hash:
0ea9c3edb47ecfe4d615032c9b6b3b32217d897807473d08ee8f80ab831d13e5
Signature:                   Ms78P66rvN5Qr7EANUN8RAafKAGW4DOh-
osmPbEs751_aJXd15wQj3zlsOhpUYWGi7L1r2_12D62LD7KcsRU0w==
Previous Hash:  8f64da547860e8dc7ffc08f298bc15d013025372df066a5d484c5232221bc108
----------------------
```

1.8.2 Double Spending

[TODO]

1.8.3 Why not generate a hash chain using the last transaction?

Need more research on this …

This will just ensure a totally chronological chain of transactions, and the reversal can only be done before another wallet transaction is added to the end. So why not use one single transaction chain?

Maybe, since lots of transactions could be created at any given time, making the last transaction hard to determine. It might be too slow for a consensus to made by the entire network to determine which transaction is the last transaction due to the sheer number of transactions made by each node at any given time. This won't be too much of an issue if we only find the last transaction of the wallet that's making the transaction.

Maybe that's why transactions are verified by Blocks instead?

To be continued ...

```
[ ]:
```