95-702 Distributed Systems For Information Systems Management

Project 3

Assigned: Friday, Feb-23, 2018
Due: Friday, Due Friday Mar-09, 11:59pm

Principles

One of our primary objectives in this course is to make clear the fundamental distinction between functional and nonfunctional characteristics of distributed systems. The functional characteristics describe the business or organizational purpose of the system. The non-functional characteristics affect the quality of the system. Is it fast? Does it easily interoperate with others? Is it fault tolerant? Is it reliable and secure?

In this project, we illustrate several important nonfunctional characteristics of distributed systems. We will be using digital signatures to authenticate messages. We will use web services to enhance interoperability. We will consider various styles of API design. We will build a stand-alone blockchain and remote clients that interact with a blockchain API.

Overview

In Task 0, you will write a blockchain by carefully following the directions in Javadoc format found here:

http://www.andrew.cmu.edu/course/95-702/examples/javadoc/index.html

In the remaining tasks, we will provide three different interfaces to this blochchain implementation.

In Task 1, you will deploy some of the blockchain methods as a SOAP based API. You will visit this API with a SOAP client. You will sign the transaction that you send to the blockchain and the blockchain will hold the transaction and the signature. There is a video here that will help in building a JAX-WS RPC-Style web service.

http://www.andrew.cmu.edu/course/95-702/video/SimpleJAX-WS/index.html

In Task 2, you will deploy blockchain methods using a single argument style JAX-WS web service. You will sign the transaction that you send to the blockchain and the blockchain will be used to store the transaction and the signature. See the videos on the course schedule titled “Service Design Styles” one through six.

In Task 3, you will deploy the blockchain methods using a resource based or REST web service. You will sign the transaction that you send to the blockchain and the blockchain will be used to store the transaction and the signature. See the videos on the course schedule titled “Service Design Styles” one through six.

Before beginning, be sure to study the Javadoc on the schedule. It describes the two classes that you need to write – Block.java and BlockChain.java.

For each task below, you must submit screenshots that demonstrate your programs running. These screenshots will aid the grader in evaluating your project.

Documenting code is also important. Be sure to provide comments in your code explaining what the code is doing and why. Be sure to separate concerns when appropriate. You may include the Javadoc comments (provided) in your own code. But do comment on any additions or modifications that you make.

Task 0 Execution

You should be able to write a solution to Task 0 by studying the Javadoc provided on the course schedule. You will need to make modifications to the code for the other tasks. Task 0 is essential. The logic found in Task 0 will be reused in Tasks 1,2, and 3.

The execution of Task 0, a non-distributed stand-alone program will look exactly like the following (of course, your block data will differ). Note, there is no signature shown and there is no signature required in Task 0.

```
Java BlockChain
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
Chain verification: true
Total execution time to verify the chain is 2 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
3
```

View the Blockchain
{"chain" : [{"index" : 0,"time stamp " : "2018-02-23 20:33:08.822","Tx ":
"Genesis","PrevHash" : "","nonce" : 32,"difficulty": 2}
],
"chainHash":"0049093C92967F864A7E5D021A4B8498A20D73C2BEB323A828232
096F59E5C66"}
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
1
Enter difficulty
3
Enter transaction
Alice pays Bob $500.00
Total execution time to add this block 30 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
Chain verification: true
Total execution time to verify the chain is 3 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
1
Enter difficulty
4
Enter transaction
Bob pays Eve $100.00
Total execution time to add this block 547 milliseconds

1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
3
View the Blockchain
{"chain" : [{"index" : 0,"time stamp " : "2018-02-23 20:33:08.822","Tx ": "Genesis","PrevHash" : "","nonce" : 32,"difficulty": 2},
{"index" : 1,"time stamp " : "2018-02-23 20:34:03.281","Tx ": "Alice pays Bob $500.00","PrevHash" : "0049093C92967F864A7E5D021A4B8498A20D73C2BEB323A828232096F59E5C66","nonce" : 126,"difficulty": 3},
{"index" : 2,"time stamp " : "2018-02-23 20:34:38.929","Tx ": "Bob pays Eve $100.00","PrevHash" : "0002458B5B34FE074DED185E9BEF87A1EF2D46622643A66072859E0DE3FD0249","nonce" : 15928,"difficulty": 4}
],
"chainHash":"00005BE57CABFDDFB7A1DB1355DBE092B559C318918D7ECDA29B4DECA7B930F4"}
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
4
Corrupt the Blockchain
Enter block to corrupt
0
Enter new data for block 0
Eve Begins with $1000.00
Block 0 now holds Eve Begins with $1000.00
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit

2
Verifying
Improper hash on node 0 Does not begin with 00
Chain verification: false
Total execution time to verify the chain is 0 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
5
Repair the Blockchain
Repair complete
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
Chain verification: true
Total execution time to verify the chain is 0 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
3
View the Blockchain
{"chain" : [{"index" : 0,"time stamp " : "2018-02-23 20:33:08.822","Tx ": "Eve Begins with $1000.00","PrevHash" : "","nonce" : 121,"difficulty": 2},
{"index" : 1,"time stamp " : "2018-02-23 20:34:03.281","Tx ": "Alice pays Bob $500.00","PrevHash" : "00F0F50C240AE316FD063F8466F993234AE79D48C09BCEF1BF41AD7C01B5AEB9","nonce" : 5610,"difficulty": 3},
{"index" : 2,"time stamp " : "2018-02-23 20:34:38.929","Tx ": "Bob pays Eve $100.00","PrevHash" :

"0005742159F19992E7D6D8D26117EBC0163630CAD5B7B3C875A1281D2E9B3F
60","nonce" : 17341,"difficulty": 4}
 ],
"chainHash":"0000BC3BF69FE0BFDA7B771A3437240E7D6A1E8423CFFAD7173E08
67E770F157"}
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
4
Corrupt the Blockchain
Enter block to corrupt
2
Enter new data for block 2
Alice pays Bob $120.00
Block 2 now holds Alice pays Bob $120.00
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
..Improper hash on node 2 Does not begin with 0000
Chain verification: false
Total execution time to verify the chain is 1 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
4
Corrupt the Blockchain
Enter block to corrupt
1
Enter new data for block 1

Mike pays Joe $34.99
Block 1 now holds Mike pays Joe $34.99
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
..Improper hash on node 1 Does not begin with 000
Chain verification: false
Total execution time to verify the chain is 0 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
5
Repair the Blockchain
Repair complete
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
2
Verifying
Chain verification: true
Total execution time to verify the chain is 1 milliseconds
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
3
View the Blockchain

{"chain" : [{"index" : 0,"time stamp " : "2018-02-23 20:33:08.822","Tx ": "Eve Begins with $1000.00","PrevHash" : "","nonce" : 121,"difficulty": 2},
{"index" : 1,"time stamp " : "2018-02-23 20:34:03.281","Tx ": "Mike pays Joe $34.99","PrevHash" : "00F0F50C240AE316FD063F8466F993234AE79D48C09BCEF1BF41AD7C01B5AEB9","nonce" : 4518,"difficulty": 3},
{"index" : 2,"time stamp " : "2018-02-23 20:34:38.929","Tx ": "Alice pays Bob $120.00","PrevHash" : "000AE1B36A1C1A2A6370D3605A3112C307FDC4D15F137BD4086D7E54AE4EBC31","nonce" : 43914,"difficulty": 4}
],
"chainHash":"000073F4DFC9DEF02A1BF438704287A90D45593178E2A3E0032F5CD4CE76BEEC"}
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by re-computing hashes.
6. Exit
6

Task 0 Grading Rubric 25 Points

See the Javadoc for a description of the main routine and the difficulty levels.

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works	19	17	10	0
The main method of the BlockChain class is documented and describes behavior with difficulty levels of 4 and 5 (see Javadoc)	4	3	2	0
Separation of concerns & good style	1	1	0	0
Submission requirements met	1	0	0	0

Task 1 Execution

In Task 1 we will deploy a set of methods from our Task 0 Blockchain.

The execution of Task 1 will look almost exactly the same as the execution of Task 0. However, our users will be remote and we are not allowing remote users to corrupt the blockchain or repair it. So, we will have only four options (rather than six) for our remote users. While the interaction will appear local, the Task 1 client will make SOAP requests to your newly written blockchain API. In addition, since this client may be remote, all transaction data will be signed. A signature will accompany every call to the service that modifies the blockchain. No signature is used for calls that only read the blockchain's data. Anyone may read, only special clients may write. (See the rubric below if you choose not to sign the transactions.)

Your client will need a full set of RSA keys (e, d, and n). These will be provided below.

These keys may be hard coded into the client side code. The server will need a copy of the client's public RSA keys (e and n) and these will be used to verify the signature.

Your task is to design and implement the API. You need to come up with your own method names and signatures. You may use any of the code from Task 0 that you need. Your client needs to make calls to the API so that it has the same execution as Task 0 (without the Task 0 options 4 and 5). An example remote execution appears next. Note that the value after the octothorp character (#) is the signature. Also note that the octothorp character is never included in a transaction entered by a user. It acts as a special symbol to delimit the transaction and its signature.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Exit

1

Enter difficulty

4

Enter transaction

Joe pays Mike 3400 USD

Total execution time to add this block 992 milliseconds

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Exit

3

View the Blockchain

{"chain" : [{"index" : 0,"time stamp " : "2018-03-09 16:56:40.31","Tx ": "Genesis","PrevHash" : "","nonce" : 261,"difficulty": 2},

{"index" : 1,"time stamp " : "2018-03-09 16:56:59.295","Tx ": "Joe pays Mike 3400 USD#7375638973577832424423670517331496029881604206185247046954104653810956417638337181645316313589775663019113638880945599048792125987413256062987096910139634497115868431926607477043355040280808754270650902459200509066674814941002493473982357 51","PrevHash" :

"0098B4C373CE65F4DE895FC5693508B18FE034AF5F075D2B96A7821004CFBBC2","nonce" : 112513,"difficulty": 4}

],
"chainHash":"0000FE321FB46F53C3BFA77AF24F055B5970C422AA0DC014067F0B2C463752F3"}

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Exit

4

Task 1 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works but with no signing	19	17	10	0
Works and the transaction data is signed and verified by the server	23	22	22	0
Separation of concerns & good style	1	1	0	0
Submission requirements met	1	0	0	0

Task 2 Execution

The execution of Task 2 will look exactly the same as the execution of Task 1. Behind the scenes, however, the Task 2 client will make SOAP requests to your blockchain API. This differs from Task 1 in that we are using a single message argument style

design. The blockchain service will need to be redesigned. See the Service Design Style videos. In addition, since this client may be remote, all transaction data will be signed. A signature will accompany every call to the service that modifies the blockchain. No signature is used for calls that only read the blockchain's data. Anyone may read, only special clients may write. (See the rubric below if you choose not to sign the transactions.)

In this case, the signature will be placed in an XML message of your own design. This same XML message will hold the other data elements and the requested operation as well – a single message design. Your API on the server side will only provide one operation. That operation will extract necessary information from the XML and decide what internal operation to perform.

Your client will need a full set of RSA keys (e, d, and n). These will be provided below.

These keys may be hard coded into the client side code. The server will need a copy of the client's public RSA keys (e and n) and these will be used to verify the signature.

Again, there is detail here that you need to work out.

Task 2 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works but with no signing	14	12	8	0
Works and the transaction data is signed and verified by the server	18	17	16	0
Single message design used by client and server & good style	6	3	1	0
Submission requirements	1	0	0	0

met				

Task 3 Execution

The execution of Task 3 will look exactly the same as the execution of Task 1. Behind the scenes, however, the Task 3 client will make REST or resource style requests to your blockchain API. This differs from Task 2 in that here we are using a REST style design. The blockchain service will need to be redesigned for REST. See the Service Design Style videos. In addition, since this client may be remote, all transaction data will be signed. A signature will accompany every call to the service that modifies the blockchain. No signature is used for calls that only read the blockchain's data. Anyone may read, only special clients may write. (See the rubric below if you choose not to sign the transactions.)

Your client will need a full set of RSA keys (e, d, and n). These will be provided below.

These keys may be hard coded into the client side code. The server will need a copy of the client's public RSA keys (e and n) and these will be used to verify the signature.

You need to think about what is actually a resource and what methods are appropriate to interact with it. You will also need to consider the format of the messages that are passed back and forth. Make reasonable decisions.

Task 3 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works but with no signing	14	12	8	0
Works and the transaction data is signed and verified by the server	18	17	16	0
REST design is used for client and server & good style	6	3	1	0

Submission requirements met	1	0	0	0

General Notes on Signing – applies to all signing work.

For signing and signature verification, you will use the following keys:

BigInteger e = new BigInteger("65537");

BigInteger d = new BigInteger("339177647280468990599683753475404338964037287357290649639740920420195763493261892674937712727426153831055473238029100340967145378283022484846784794546119352371446685199413453480215164979267671668216248690393620864946715883011485526549108913");

BigInteger n = new BigInteger("268852025517901502623747873143657162103121815451557296872758837706559866377091251333301800665424865065625091311087483660777796686710629019261833666084998095639973296736997628150027028645031319958686197762350334823785557943447125197765366255 3");

Key management is always tricky. In this simple example, it is fine to store the full key set in the client side code – representing a user holding its own keys. On the server, you need to store the public keys (only e and n) of the user. Again, it is fine to store these keys in the server side code. The value of d, however, should not be stored in the server code.

The computation of the signature string must be done as follows:

0) The string to be signed is the string entered by the user.
   For example, the string “Joe pays Mike 1000 USD” would require a signature.
1) Get the bytes from the string to be signed.
2) Compute a SHA-256 digest of these bytes.
3) Copy these bytes into a byte array that is one byte longer than needed.
   The resulting byte array has its extra byte set to zero. This is because RSA works only on positive numbers. The most significant byte (in the new byte array) is the 0'th byte. This byte must be set to zero.
4) Create a BigInteger from the byte array.
5) Encrypt the BigInteger with RSA d and n.

6) Return to the caller a String representation of this BigInteger. Pass this signature, along with the data, to the server. Use an octothorp (#) as a separator. In the example above, we would pass to the server one string. "Joe pays Mike 1000 USD" + "#" + E(SHA256("Joe pays Mike 1000 USD"),K_{priv}).

The verification of the signature must be done as follows:

On the server side:

0) Decrypt the encrypted hash to compute a decrypted hash. That is, compute D(E(SHA256("Joe pays Mike 1000 USD"),K_{priv}),K_{pub}).

1) Hash the arriving message using SHA-256 (be sure to handle the extra byte as described in the signing method.) That is, hash the message received from the caller "Joe pays Mike 1000 USD".

2) If this new hash is equal to the decrypted hash then the signature has been verified. If it is not equal, the data may have been tampered with. Do not add this transaction to the blockchain. Return information to the caller that lets her know that this transaction has not been accepted. Make reasonable decisions.

Notes on Task 2 API Design - Single Message Style

In this task, you will pass a single message (containing a single String holding an XML message) to a single JAX-WS service operation. The service will examine the message and will check the signature and if the signature is valid, will store the transaction on the Blockchain. If the message is a query message then no signature needs to be checked.

In this Task, you will need to design your own message format in XML. Your client will need to write that message and your server will need to parse it. Use a parser only when reading XML. There is no requirement to design a return message format. For those operations returning string data, a simple Java string will be fine.

For parsing the XML, you may use Java's parser.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
Document sensorMessage = null;
builder = factory.newDocumentBuilder();
```

```
blockChainMessage = builder.parse( new InputSource( new StringReader(
                                        xmlString ) ) );
```

Once the xmlString is converted to a Document object, we can read data from the Document object with code like the following:

```
blockChainMessage.getDocumentElement().normalize();
System.out.println("Root element :" +
blockChainMessage.getDocumentElement().getNodeName());
```

This gets you started. There are still issues needing thoughtful work. Make reasonable decisions.

Notes on Task 3 API Design – Resource or REST style

In this style, you will make good use of HTTP verbs, status codes and URL's.

You will not be using JAX-WS or JAX-RS for this task. Instead, use a plain servlet.

You still need to sign messages coming from the client. The signatures need to be verified on the server.

The HTTP status codes may be set on the server side by using the setStatus() method of the response object. To learn what representation the client requires, use the getHeader("Accept") method of the request object.

The client side code (standard Java SE) will perform communication using the URL class and the HttpURLConnection class. For example, in order to make a call to the get method, your code might look something like this:

```
URL url = new URL(http://localhost:8080/Project3Task3/BlockChainService");
HttpURLConnection con = (HttpURLConnection) url.openConnection();
con.setRequestMethod("GET");
con.setRequestProperty("Accept", "text/xml");
```

The client side code can use the con.getResponseCode() method to examine the HTTP response code. It can also use the con.getInputStream() method to read the server's response body.

The client side code will use a proxy design. In other words, all of the communications related code will be isolated within proxies. Your client should provide a demonstration of each operation (as in Task 1).

Hint: You may use StackOverflow for examples on how to use the HttpURLConnection. If you use any code from StackOverflow, be sure to cite it in your comments.

Note: It is a bad idea to write your own cryptographic software – unless you are an expert with years of experience. We do this exercise only to understand and explore some major issues in computer security.

Note: This is not all of blockchain. There is more to learn. Real blockchains include peer to peer communication and many replicas of the blockchain. The blocks themselves include Merkle Trees and real transactions are more carefully crafted. This assignment will certainly provide a nice foundation to build on.

Questions

Questions should be posted to Piazza. If there are missing pieces in this assignment write up (we are sure there are) then make reasonable decisions. Try to build a system that is well thought out and keep things as simple as possible.

Project 3 Submission Requirements

Documentation is always required.

Remember to separate concerns.

The Netbeans projects will be named as follows:

- Project3Task0 (You need to zip this folder)
- Project3Task1Server (You need to zip this folder)
- Project3Task1Client (You need to zip this folder)
- Project3Task2Server (You need to zip this folder)
- Project3Task2Client (You need to zip this folder)
- Project3Task3Server (You need to zip this folder)
- Project3Task3Client (You need to zip this folder)

You should also have four screen shot folders:

- Project3Task0-screenshot (Do not zip)
- Project3Task1-screenshot (Do not zip)
- Project3Task2-screenshot (Do not zip)
- Project3Task3-screenshot (Do not zip)

For each Netbeans project, use “File->Export Project->To Zip”. You must export in this way and NOT just zip the Netbeans project folders. In addition, zip all the Netbeans export zips and the screenshot folders into a folder named with your andrew id.

Zip that folder and submit it to Blackboard.

The submission should be a single zip file. This file will be called YourAndrewID.zip.

Submission file structure:

- YourAndrewID.zip
  - Project3Task0.zip
  - Project3Task1Server.zip
  - Project3Task1Client.zip
  - Project3Task2Server.zip
  - Project3Task2Client.zip
  - Project3Task3Server.zip
  - Project3Task3Client.zip
  - Project3Task0-screenshot
  - Project3Task1-screenshot
  - Project3Task2-screenshot
  - Project3Task3-screenshot

To add .jar to your project so that it is included when you zip the project

To ensure that the .jar that you have included under your Libraries folder in the NetBeans Project is also zipped along with your project, follow these steps:
1. From Windows File Explorer, create a folder named lib under your project folder. i.e., if this is you project folder path, C:\Users\username\Documents\NetBeansProjects\Project1Task2 then this is where you will create the lib folder. (this is an example path, it will change according to your system)
2. Within the lib folder, copy and paste your downloaded .jar files.
i.e., your .jar files go here: C:\Users\username\Documents\NetBeansProjects\Project1Task2\lib (this is an example path, it will change according to your system)
3. In NetBeans, go to the Libraries under your Project1Task2 and delete the existing .jar file that you had earlier added.

4. Right click Libraries and choose Add JAR/Folder
5. Browse to the lib folder you created in step 1. The .jar you added to the lib folder (step 2) will be visible there. Click on that .jar Ensure that 'Reference as': is set to the radio button 'Relative Path'
6. Click Open and you're done!
7. Repeat steps 4-6 for any more .jar