Technical Report: Proof of Retrieve-ability

# Instructions to set up and use the Amazon account:

Please follow the steps below in order to setup your Amazon Web Services (AWS) account for use with the ‘Proof of Retrieve-ability’ (POR) system.

1. Create a new AWS account at <http://aws.amazon.com/> by clicking on **Sign Up**.
2. Follow the onscreen instructions. You will need to enter your credit/debit card and address information as EC2 requires a credit card on file for use. A $1 USD hold may occur on the card during verification. Your EC2 usage charges will be borne through this credit/debit information.
3. Once you have completed the EC2 sign up, and completed the phone verification process, wait for EC2 authorization. This can take up to one business day.
4. Note the email address you used to sign up for Amazon EC2, since the same should be used for signing into this account in the future. Also note the 12-digit AWS account number in the top right corner of the AWS ‘My Account’ page.

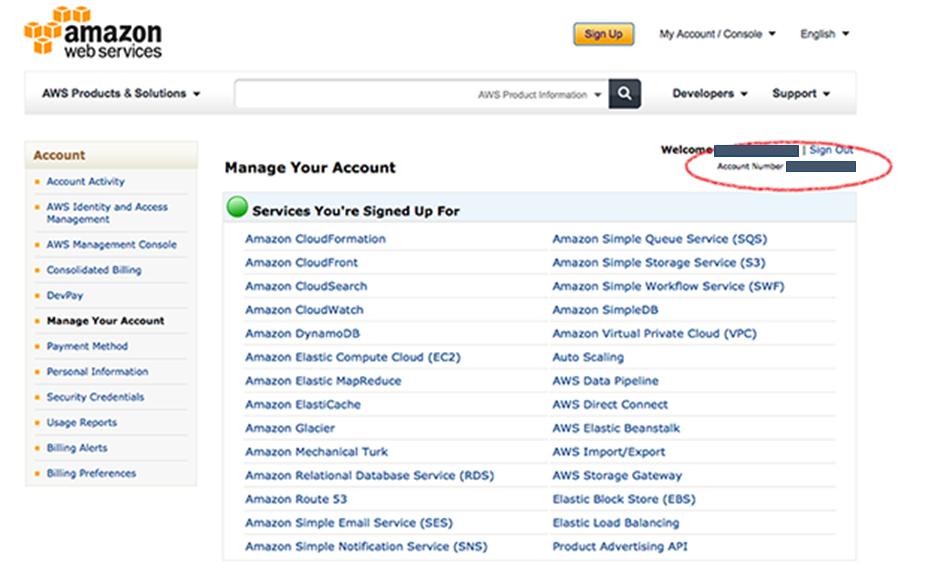


Figure 1 My Account Page - Amazon AWS

1. You should receive a consolidated billing request in your email, please click and approve the request.
2. You can now sign into your account at [http://aws.amazon.com](http://aws.amazon.com/).
3. Once you have signed in, choose **Security Credentials** from the drop-down menu on the top right corner. Under **Access Keys** click on **Create New Root Key** and note down the **Access Key ID** and the **Secret Access Key**. You will need to enter these access keys in the application when requested.

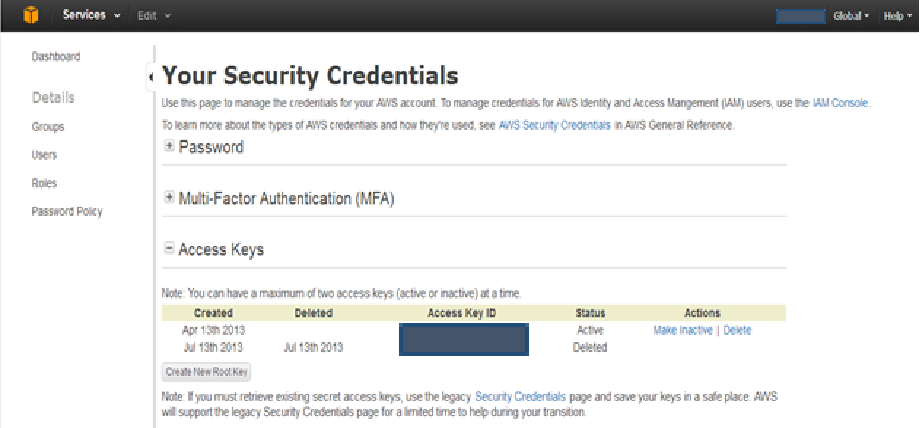


Figure 2 Your Security Credentials

1. Go to **EC2 Dashboard**, then **Key Pairs**, and then click the **Create Key Pair** button. Choose a name (it is best not to use spaces). Make sure you save the .pem file in a safe location – you will not be able to download it again later. If you do lose the .pem file or it gets compromised, you can delete the key and generate new one(s).

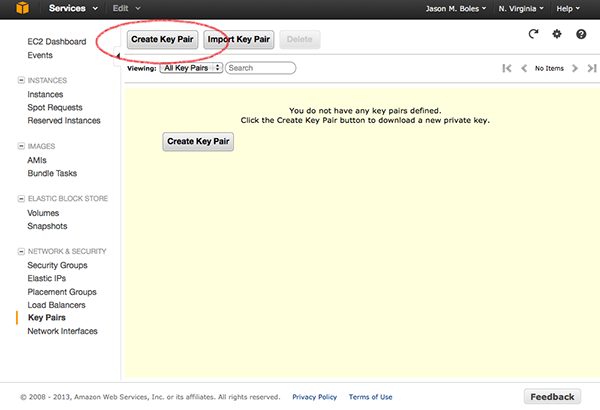


Figure 3 Create Key Pair

## Instance manipulation:

We (The Guardians) have used the following steps during development and testing in order to check the status of the Amazon EC2 instances created for deploying the system. Ordinarily, you, as a user will not need them for trying out the system yourself, but can be used when using the system for further modifications/testing.

1. Go to the **EC2 dashboard**, then **Instances,** then **‘Launch Instance’** to create a new instance. Amazon will guide you through the steps to launch your new instance. (For using small files in the POR system, a ‘t1.micro’ instance in the free tier is sufficient. For large files, an ‘m1.small’ or larger needs to be used.)
2. During its lifecycle, the following operations can be performed on the instance to change its state.
3. **Terminate:**

This will terminate your instance, such that it cannot be used again. The instance is shut-down (dead).

1. **Reboot:**

This will ‘refresh’ the instance. The instance will again be in the ‘running’ state. The state of the disks is kept intact, and rebooting does not also affect the billing of the instance.

1. **Stop:**

From this state, you can either terminate or start the instance again. This is the ‘not running’ state and you will not be charged for the time your instance is in the ‘stopped’ state.

1. **Start:**

A stopped instance can be started, such that you can use it again. The state from before you stopped it is lost. This will now be a new state.

A running instance can be terminated, rebooted or stopped; but not started.

## Using the Elastic Block Storage (EBS) to store files:

We used the EBS to store large files, such that the files could be manipulated by the various functions of the POR system on the fly.

In order to create an EBS volume for your instance, right-click on the instance and click on the ‘create image’, then on ‘EBS volume’ in the dialog box that pops up. Enter the details for your volume and click on ‘Yes, create’. You will be able to watch your new EBS volume on the EC2 dashboard, under ‘Volumes’.

**Point to note:**

It is necessary that any file to be uploaded to the Amazon cloud needs to have an access level 400, which can be specified by the command ‘chmod 400 <file-to-be-uploaded>’ in the terminal of your system.

*[Reference: 15-619: Cloud Computing at Carnegie Mellon University]*

# Setting up the workspace:

1. Install the Netbeans development tool along with Java support from the Oracle website at [http://www.oracle.com/technetwork/java/javase/downloads/jdk-7-netbeans-download-432126.htm](http://www.google.com/url?q=http%3A%2F%2Fwww.oracle.com%2Ftechnetwork%2Fjava%2Fjavase%2Fdownloads%2Fjdk-7-netbeans-download-432126.html&sa=D&sntz=1&usg=AFQjCNHe_4SoLTtz34PW6PkG81bicglADg)
2. Install truecrypt from <http://www.truecrypt.org/downloads>, choosing the .tar.gz file under ‘Linux’.
3. Install Libtom
   1. Install latex from the software center on your linux platform.
   2. Run the ‘sudo addgroup wheel’ command on the ‘terminal’ of your linux operatig system to add the default group in the installation to your system groups.
   3. Run the commands ‘make’ and ‘make install’ from inside the downloaded folder on your system terminal to install libtom.
4. In Netbeans, import the project from the folder provided. Right-click on the project, project properties, and click on ‘Add jar’ to add the following jar files:
5. **Log4j library: (apache-log4j-1.2.17)**

Download the file from <http://logging.apache.org/log4j/1.2/download.html>. The jar file is in this parent folder.

1. **AWS SDK for Java: (aws-java-sdk-1.5.1)**

Click on the “AWS SDK for Java” button on the top of the page at <http://aws.amazon.com/sdkforjava/>. Save the file when prompted. The .jar files are under the lib folder of the downloaded parent folder.

1. **Apache Commons library: (commons-logging-1.1.3)**

Download the library from <http://commons.apache.org/proper/commons-logging/download_logging.cgi>. The jar files are in this downloaded parent folder.

1. **HTTP Components: (httpcomponents-client-4.2.5)**

Download the file from <http://hc.apache.org/downloads.cgi>. The jar files are under the lib folder of this downloaded parent folder.

1. In your linux system, create a directory /media/Disk1 for use by truecrypt. Make sure that the file to be mounted for truecrypt doesn’t exist already.

## Configuring parameters in the configuration file for the entire project

\POR\Config\Config.properties contains parameters needs to be configured before running the project for the first time.

Here is the sample file:

LOGGER\_PATH = /home/poojad/POR/config/log4j.properties

AMI\_ID = ami-d0f89fb9

INSTANCE\_USER\_NAME = ubuntu

EXECUTABLE\_PATH = /home/poojad/POR/exec/

FILE\_TO\_BE\_MOUNTED = /home/poojad/POR/config/truecryptmount

MOUNT\_PATH = /MEDIA/DISK1

DEFAULT\_PARTITION\_SIZE\_MB = 1074000000

TRUECRYPT\_PASSWORD = cmuboschpor

RANDOM\_FILE = /home/poojad/POR/config/random

KEY\_FILE = /home/poojad/POR/config/truecryptkey

KEYSTORE\_FILE\_NAME = keystore.txt

CHART\_PATH = /home/poojad/POR/output

POR\_ENCODER\_EXECUTABLE = porencoder

POR\_DECODER\_EXECUTABLE = pordecoder

POR\_SERVER\_EXECUTABLE = porserver

POR\_CLIENT\_EXECUTABLE = porclient

### logger\_path

This should contain path of the log4j.properties in the system. This is required in order to set up the Apache logger.

### ami\_id

It consists of id of the Amazon machine image being used.

1. In order to select the AMI ID, sign up <http://aws.amazon.com/console/> as mentioned in cloud usage guide.
2. Go to the AWS management console
3. Find ‘Instances’ tab on the left hand side, and click on ‘Instances’ item. It will open this screen.

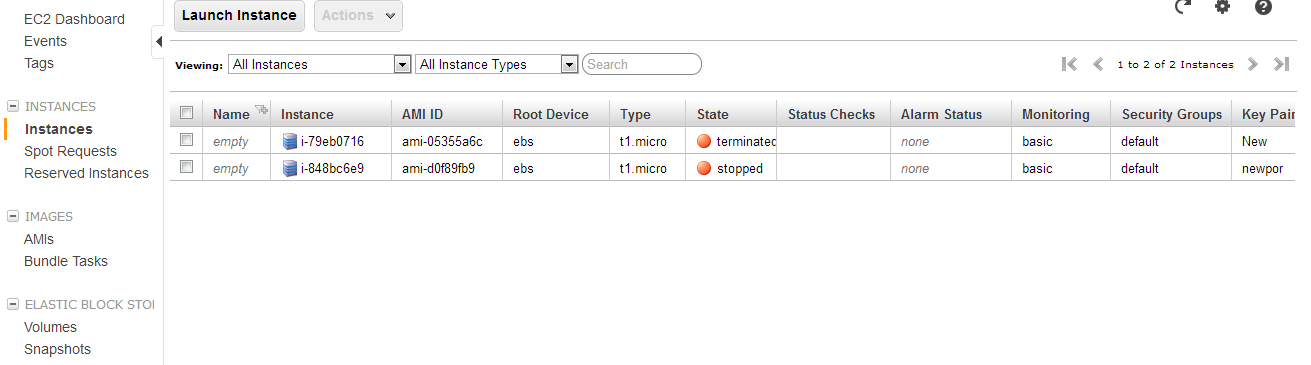


Figure 4 Launch Instance

1. Click on the ‘Launch Instance’ button. It will open this screen.

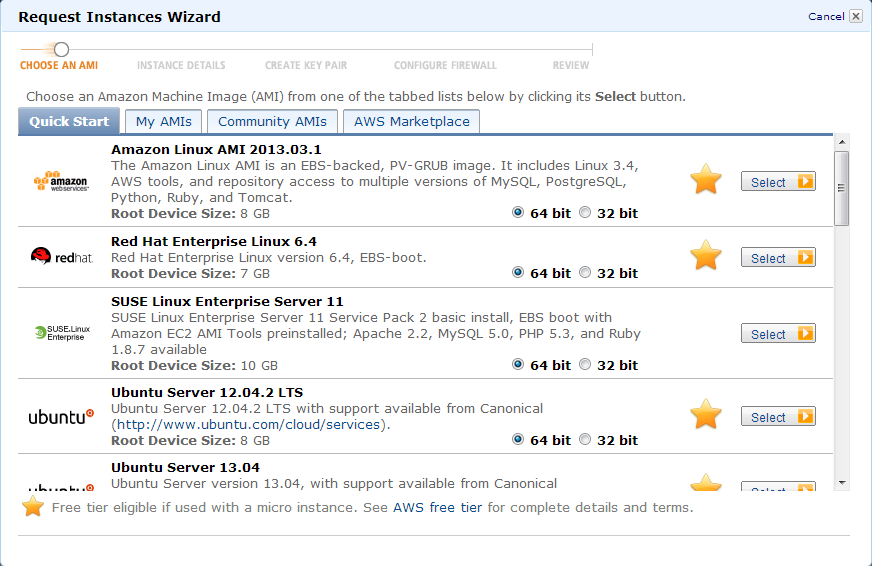


Figure 5 Request Instance Wizard

Select the type of instance depending upon your preference. We have used ‘ami-d0f89fb9’ as a default AMI. Go ahead and start the instance. Once instance is launching, you can see AMI ID here-

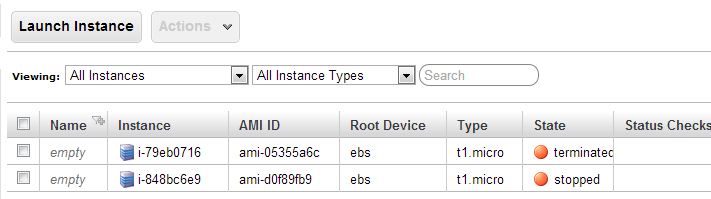


Figure 6 AMI ID

Use this id in the file.

### instance\_user\_name

This name changes depending upon the instance used. To find it, you can right click on the instance and say “connect”. The user name can be found via the ssh command provided.

### executable\_path

This should contain the path where all the ‘C’ executables are stored. Please make sure to include / at the end.

### file\_to\_be\_mounted

This is the file that needs to be mounted as a truecrypt drive.

### mount\_path

Here is the path where truecrypt volume will be mounted. Make sure this directory exists.

### default\_partition\_size

This is the size of truecrypt partition.

### truecrypt\_password

This is the password required in order to mount and dismount the truecrypt volumes. Keep this password confidential in order to protect the content of the truecrypt volume.

### random\_file

This can be any random file containing 320 characters.

### key\_file

This is the key file to be used in addition to the password for truecrypt.

### keystore\_file\_name

This file will store the user name , file name and passwords.

### chart\_path

The resulting charts will be generated to this path upon successful file upload and download.

### por\_encoder\_executable

This includes name of C executable of POR encoder. Change it only if you recompile POR encoder with different settings.

### por\_decoder\_executable

This includes name of C executable of POR decoder. Change it only if you recompile POR encoder with different settings.

### por\_server\_executable

It contains name of C executable of POR server. Change it only if you recompile POR decoder with different settings.

### por\_client\_executable

It contains name of C executable of POR client. . Change it only if you recompile POR decoder with different settings.

## Configuring log4j.properties

Please change path of the log file where content will be actually logged.

log4j.appender.A1.File=/home/poojad/config/prolog

In order to change other parameters used in this file, please refer to this link.

<http://logging.apache.org/log4j/2.x/manual/configuration.html>

# Run the System

In order to run the system, go to NetBeansProject/<ProjectName>/dist/<ProjectName>.jar

Use this path on the terminal and execute the command

Java –jar <ProjectName>.jar

This should open a welcome screen and you should be able to run the system.

In case of any errors, please check console or logs.

Note: In order to use the system with an AWS account, you need to reserve a key pair especially for working with the system. In order for the system to work, remember that you will not be able to create/have/make use of another key pair during this keypair which you have reserved for the POR system is contained by your AWS account.

# Making the system configurable:

## Configuration for different cloud providers

The system currently supports the Amazon cloud. However, in order to support configuration for different cloud providers, we have provided selection screen. You can edit the list of available cloud providers here.

1. Open up NetBeans and load the project.
2. Go to com.cmu.edu.por.CloudSelector.java.
3. Open the design tab.
4. Click on the list box.
5. Click on ‘Properties’ tab appearing on the right hand side of the screen.
6. On the tab, ‘VALUES’ edit different cloud provider names that you would like to display on the screen.

In order to modify the events those take place after selecting the cloud providers:

1. Go to design tab of com.cmu.edu.por.CloudSelector.java.
2. Click on the Select button
3. In the code, add ‘if’ condition- indicating the name of the newly added cloud provider.
4. In the body of the loop, indicate the class and method which will have logic of connecting to the new cloud provider.

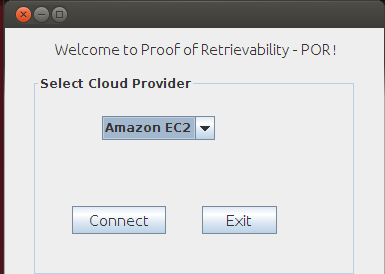


Figure 7 Cloud Selector Screen

Choosing the Amazon cloud will stake you to the following screen:

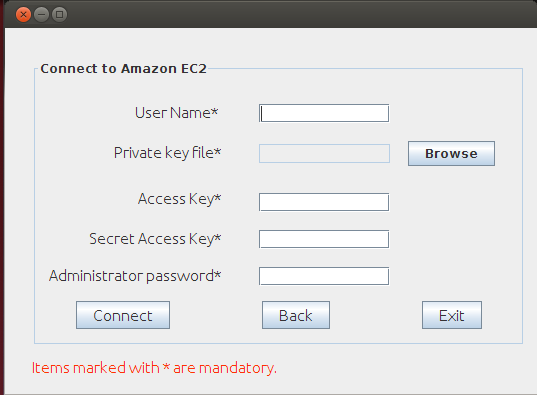


Figure 8 Connect To Amazon Screen

You need to create a similar screen for new cloud provider. It should include the parameters you need in order to connect to and use cloud instance.

Here are some of the methods that you need to include in the new class that handles connection with cloud provider

1. Launch the instance
2. Start the instance
3. File Upload
4. File Download
5. Connect to the instance
6. Stop the instance

## Configuring macros in .c files for the particular algorithm

The following parameters can be configured by changing the values of the macros in the C header file por.h:

**#define readLen 1024\*1024\*1024** *// incremental encoding read amount*

**#define alpha 10 // strength parameter for inner decoding**

**#define delta 0.25 // confidence parameter for inner decoding**

**#define BLOCK\_SIZE 32**

**#define n 255 // n,k,d is for outer layer ECC**

**#define k 223**

**#define d 32**

**#define MACSIZE 16**

**#define PORT "3490"** *// the port users will be connecting to*

**#define v 1024/32**

**#define w 4096/32**

**#define LEN 16**

**#define BLOCK\_SIZE 32**

**#define q 100**

**#define BACKLOG 10** *// how many pending connections queue will hold*

**#define n1 64** *// n1,k1,d1 is for concatenated outer code*

**#define k1 32**

**#define d1 32**

**#define n2 64** *// n2,k2,d2 is for concatenated inner code*

**#define k2 32**

**#define d2 32**

**#define MAXDATASIZE 100**

# System work-flow:

**Point to note:**

Before running the system, make sure that the properties-configuration file in the folder provided shows the correct file paths and other information specific to your system.

The following series of screen-shots will guide you through the steps of running the system through the graphical interface:

This is the initial screen – to select the cloud providers, and connect to it. It will launch the instance, if not already launched, and connect to it.

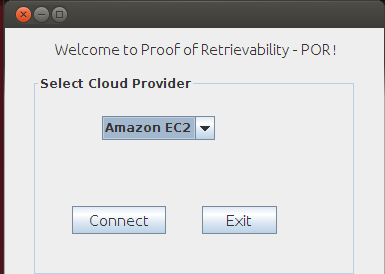


Figure 9 Cloud Selector Screen

On the next screen, enter the information required to connect to the Amazon EC2. It includes, username, <KEY>.pem file, access key and secret access key provided by Amazon and the administrator password of your operating system for mounting truecrypt.

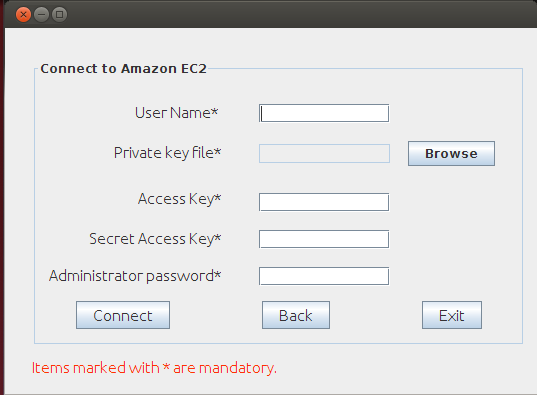


Figure 10 Connect to Amazon EC2

Once connection to cloud is successful, you will see main menu indicating three main functionalities – File upload, File Download, and checking the availability.

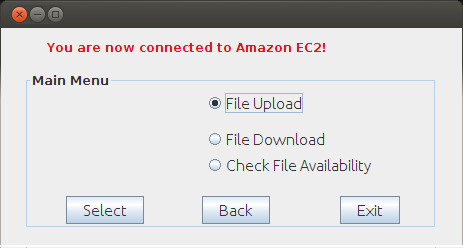


Figure 11 Main Menu

You can choose file to upload on this screen. Please enter a master key which is unique for this file. It can be any combination of letters, digits or special characters.

If the file is successfully uploaded, you will be taken to this screen. It indicates the file size before uploading the file and after, including time taken in each step.

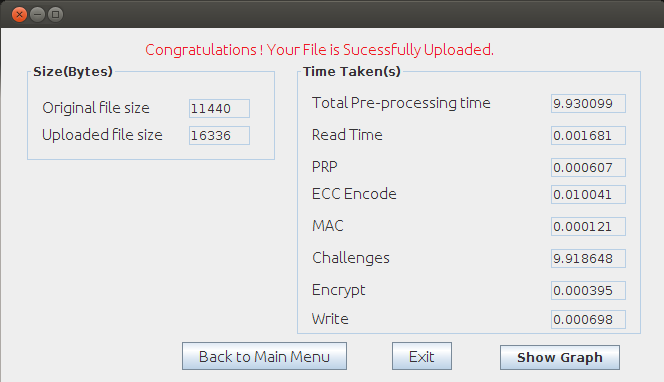


Figure 12 Upload Success

‘Show graphs’ shows the above information pictorially. It plots graph of total time taken, which gives proportion of time taken in each step.



Figure 13 File Upload chart

You can now go back to the main menu, and click on file download –

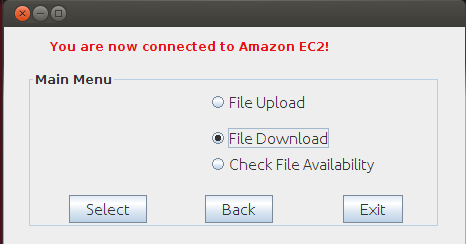


Figure 14 File Download Screen

You will see a screen, listing all the files you previously uploaded on the cloud. Select the file you would like to download and enter the master key you entered previously while uploading the file. If you enter wrong master key, you won’t be able to download the file.

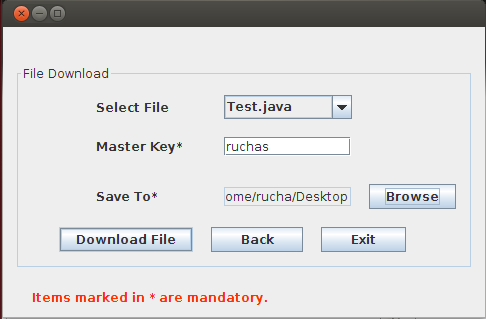


Figure 15 Select file to Download Screen

Once the file download is successful, you will see this screen – It indicates the time taken in each step to download the file completely.

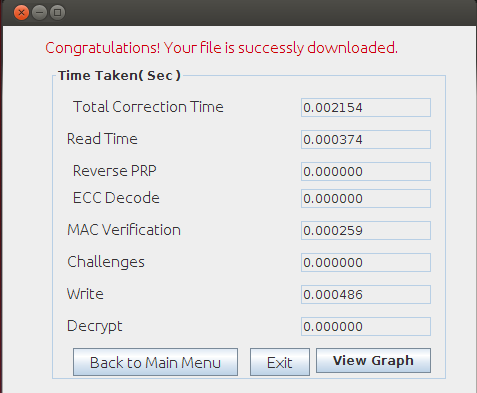


Figure 16 File Download Data

On Clicking show graph, you will see data visually, plotted as a bar chart.

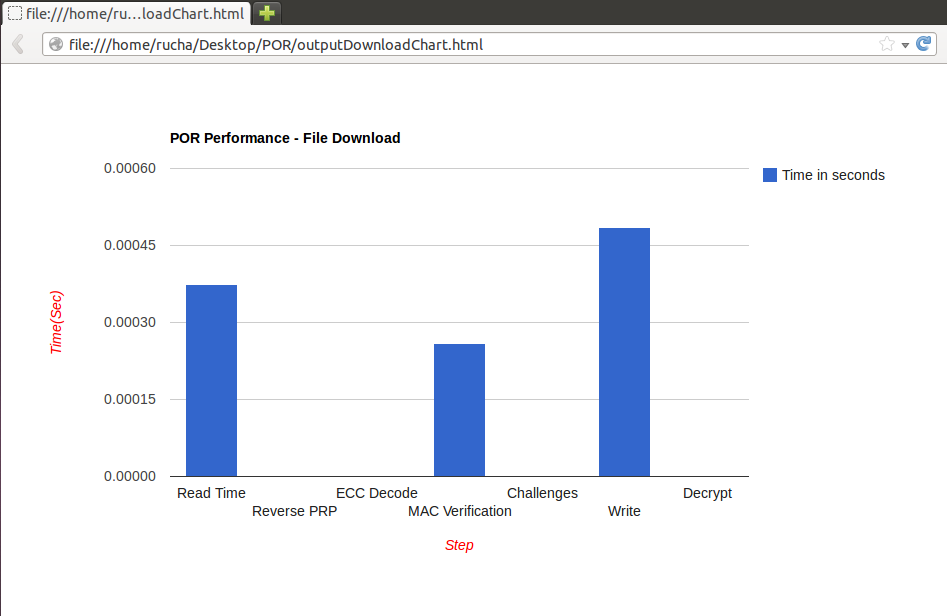


Figure 17 File Download Chart

Check the main menu, and go to ‘Check File availability’ to see if the file is available.

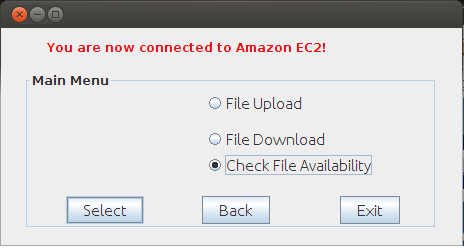


Figure 18 Check File Availability

This will show up the next screen listing the files you have uploaded previously, along with master key used to upload the file on the cloud.

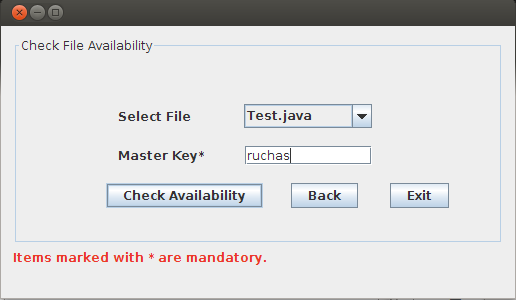


Figure 19 Check File Availability

If the file is available with the cloud provider, you will see below mentioned screen.

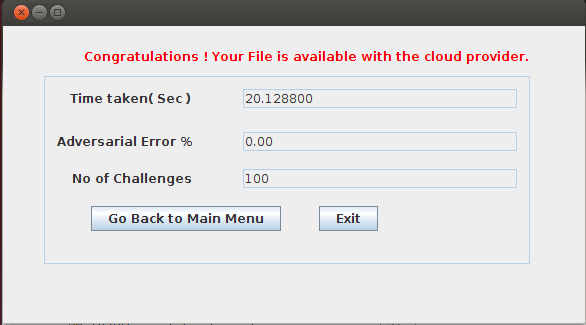


Figure 20 File is available

If the file is corrupted or the server does not respond correctly, you will see the error percentage.

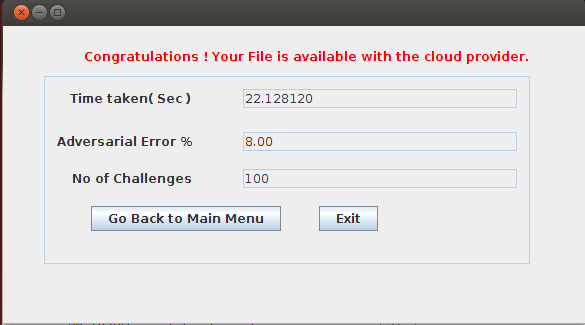


Figure 21 File is corrupted

# Code Repository Structure

Net bean project: It contains the Java GUI part of the POR system

POR folder: It contains all the C code of the POR system

Demos: It contains all the source code for experiments we have done

# Experiments Performed

## RS ECC

In order to learn and understand error correction, we performed an experiment with RSCode library. The experiment involved noting down the time required for encoding and decoding for different file sizes and understanding if it is linear.

After the experiment, we found that both encoding and decoding are linear with file sizes and decoding method takes about 20% more time than encoding.

## MAC

For creating MACs, we used libtom. We checked the performance of different types of MACs. All the time performances are linear to the file sizes. Here we used hmac with SHA-1 in our system for better performance.

## PRNG

For generating random numbers, we used libtom again. Libtome provides four different functions for generating pseudo-random numbers which involves – yarrow, fortuna, RC4, SOBER 128.

Although RC4 and SOBER-128 gives better time performance, as indicated in the libtom manual, they are weaker than Yarrow and Fortuna. Since Fortuna performs better than Yarrow on large PRNG, we chose Fortuna for our implementation.

## SOCKET

We compared the performance of TCP and UDP and found out that the transmission rate for TCP and UDP are actually very similar, however the packet loss rate for using UDP is considerable. So, for all the network transmission in our system, we chose to use TCP. Since we performed this experiment in an early stage, we do not have formal evaluation report for that.

## PRP – Ordinal, SRF, Fast

In order to do pseudo random permutation, we performed experiment with three different approaches.

1. **Six round Feistel cipher**

From here we can see that the time is linear with file sizes. But here we need to notice that when we performing this benchmark, we haven’t considered using cycle walking to guarantee the uniqueness of the PRP. So, the time here is much shorter than the one in encoding benchmarking.

1. **Ordinal Sorting PRP**

In ordinal sorting PRP, we used quick sort to reorder the random number generated for each block and perform the PRP. Since sorting is required in this PRP approach, the average algorithm complexity is O(NlogN) which is consistent with what we observed.

1. **Fast PRP**

For Fast PRP, we only did a C code implementation without using Intel assembly instruction. We found that without using assembly language to count zeros, the complexity to counting is O(N). So the algorithm is best case O(NlogN), worst cast O(N^2) depending on the distribution of 0s and 1s. Since we only implemented it for checking correctness, there’s no performance evaluation associated.

# Error Introducer

In order to introduce error at the server we wrote error introducer that can introduce errors randomly or in burst.

This error introducer takes following parameters as input

1. Error type – Burst/ Random
2. Number of errors
3. Master key for error introduction
4. File in which error is to be introduced

# Algorithms:

## POR Encoder

Here is the description of the POR encoder algorithm:

BEGIN

1. Generate Keys for
   1. File permutation (k\_file\_perm)
   2. ECC permutation (k\_ecc\_perm)
   3. ECC encryption (k\_ecc\_enc)
   4. Challenge generation (k\_chal)
   5. Random index generation (k\_ind)
   6. Response encryption ( k\_enc)
   7. Max computation (k\_mac)

We use 16 bytes for all the key sizes.

1. Compute the MAC over entire file. We are using hMAC.
2. Divide the file F into m blocks, each block of l symbols from F1, F2, … , Fm. We have used l = 32 bytes. This is a configurable parameter. Pad the last block with zeroes, if required in order to make this a 32 byte block.
3. Compute PRP over the file. We are using six round Feistal cipher as a method of pseudo random permutation. Note: We perform SRF PRP only on the block indices, and not on the actual file. This saves read time of the file.
   1. We use k\_file\_perm to generate six different seeds.
   2. These seeds are in turn used to generate six functions. Each function generation involves
      1. Each function contains random number numbers generated from the index 1 to m.
      2. Each random number is 4 byte long.
   3. After this, we calculate a and b.

a = sqrt (m)

b = sqrt (m) + 1

We need to make sure a\*b > m and b-a is very small. However, no paper mentions how to calculate a and b exactly, and what small means as a difference.

* 1. Now we apply six rounds of Feistel cipher, each round inputs – NO\_OF\_ROUNDS, a, b, number of blocks and block length.
     1. In each round, we calculate L and M.

L = m%a

M = m/a

* + 1. For i in 1 to NO\_OF\_ROUNDS
       1. Apply round function – get the number from the random tables we generated before
       2. If i is odd
          1. tmp **=** (L **+** fval) **%** b;
       3. Else
          1. tmp **=** (L **+** fval) **%** a;
       4. L = R
       5. R = tmp
    2. If NO\_OF\_ROUNDS is even
       1. Return aL + R
    3. Else
       1. Return aR + L
    4. Note: this does not guarantee that the returned number will be unique. Hence, we perform this check explicitly. We maintain an input array ‘count’. If the number is repeated, we perform cycle walking. However, in reality, we performed cycle walking and found the convergence speed is very slow especially for the last few items. So instead, we chose to do a linear probing which looks for next available slot.
    5. We check if the returned number lies within the m, number of blocks. IF it lies outside – then we perform cycle walking until we get the number inside the domain.
  1. We store the permuted block indices in the array – prpblockindices and return the same.

1. Apply outer error correcting layer on the file. We perform incremental encoding as follows
   1. We have made number of bytes to be read into the memory configurable. We call this variable as readLen.
   2. For each part of file of size readLen, do incremental encoding
      1. For each byte in the message, compute the index.
      2. Read block and byte index.
      3. If you reach end of the file, pad zeros in order to complete the message
      4. Compute the PRP index of the message
      5. We then check if this byte is read into memory, if not we load it.
      6. We then perform partial encoding on the message to get original message and the parity part.
   3. We perform PRP on the parity part.
   4. We then encrypt the parity part.
   5. Encrypted parity is then appended at the end of original file.
2. Pre compute challenges and responses and append at the end of the file.
   1. Use key k\_chal to generate key k\_ind.
   2. Generate v random indices for each of the q challenges, where each index is 8 byte long. Use k\_ind for generating the indices
   3. Use k\_ind to generate random index u.
   4. For each challenge
      1. Perform concatenated encoding to compute the response
         1. Perform striping
         2. Do concatenated outer encoding on the file. Keep the message part together and append parity part at the end.
         3. Perform concatenated inner encoding on this file.
      2. Encrypt the response and append at the end.
3. Append MAC at the end of the file

END

## POR Decoder

BEGIN

1. Open the encoded file for reading
2. Use the master key entered by client and generate all the keys
3. Read MAC from the end of the file and store it
4. Compute the MAC on the original file
   1. Read first m blocks of the file
   2. Compute the MAC on the m blocks.
5. Verify if both the MACs are same
   1. If yes, file is intact. Display the message to the user and exit
   2. If no, go ahead.
6. Perform outer decoding on the file.
   1. Read parity part from the encoded file and write into a separate file.
   2. Find out number of stripes in the parity file
   3. Calculate reverse prp for the size of stripes of parity file
      1. Perform normal prp
      2. Reverse the arrays.
   4. Find the original position of parity blocks using reverse prp table and decrypt the blocks
   5. For i in 1 to m( no of blocks)
      1. Associate each parity block with the message
      2. Once the message with parity is formed – decode the codeword to get the input message.
      3. This will correct errors and erasures.
   6. Once step (e) is completed, Compute reverse prp for the block size of 1 to m.
   7. Unpermute the file using this table, and put the decoded data in the correct format.
7. Compute MAC on the file after outer decoding and check if the file is completely recovered.
   1. If yes, file is intact. Display the message to the user and exit
   2. If no, go ahead.
8. Now, allocate memory for D- it uses format like this

*//decoding file blocks structure*

**typedef** **struct** d\_b{

**unsigned** **char** file\_blocks[alpha][32];

**unsigned** **char** frequency[alpha];

}decoded\_blocks;

1. Calculate number of challenges as q = (alpha \* t / v)
2. Populate the challenge set.
   1. Use k\_chal to generate k\_j\_c
3. For each challenge from 1 to q
   1. For u in 1 to w
      1. Execute the challenge with parameters j, k\_j\_c, u.
         1. Calculate v file indices
         2. For I in 1 to v
            1. Read the file block according to each index and form a message
         3. Perform concatenated encoding on the message
         4. Get the uth symbol from the code-word and return it.
      2. This will return part of the codeword
   2. Now we have the complete codeword to decode.
   3. Carry out inner error decoding
      1. We first perform concatenated inner error code decoding
      2. N1, k1, d1 stand for concatenated outer code and n2, k2, d2 stand for concatenated inner code
      3. Perform GMD concatenated decoding
         1. Decoding from inner code
         2. Compute the distance between codeword and message decoded, save it as delta\_dist
         3. With the probability of 2\*delta\_dist/d2, assigned the message to be an erasure, otherwise, remain the same message
         4. Decoding from outer code using the erasure information
      4. Carry out outer layer ECC decoding.
         1. Create codeword from message and parity part.
         2. Decode using c\_out
         3. Correct erasures if any
      5. Populate the Di with v file blocks
         1. In Di, we do not store actual codeword many times, we store keyword and frequency to save the space
         2. If the codeword has appeared previously, increment the frequency
         3. If not, add a new codeword and initiate its frequency as 1.
4. Now carry out maximum likelihood decoding.
5. For I in 1 to t
   1. Find out number of blocks at each Di
   2. Find out the block occurring maximum number of times.
   3. If this frequency is less than (delta \*0.5) output erasure. For erasure we use -1 in the Di
   4. If the frequency is more than (delta \* 0.5) then output that block as the value of Di
6. Perform outer decoding on the file obtained by this procedure.- EDIT
7. Calculate the MAC over the file and compare it with the stored MAC of encoded file
   1. If they are equal – output the message “ your file is recovered” with the error percentage and timing for different steps
   2. IF they are not equal – output the message “Your file cannot be recovered, please contact cloud provider.”

END

## POR Client

BEGIN

1. Generate all the keys from master key.
2. For each of the q challenges -
   1. Use k\_chal to generate k\_j\_c
   2. Use k\_ind to generate random index u.
3. Send each challenge to the server –j, k\_j\_c, u
4. Receive response from the server Qj and Mj
5. Decrypt Qj and verify if it is same as Mj
6. Keep the counter of correct challenges

END

## POR Server

BEGIN

1. Server receives each of the challenge from client containing k\_j\_c , u
2. For each challenge
   1. Read the file for computing the response
   2. Execute the challenge to calculate the response.
      1. Calculate v random indices to generate k\_j\_c
      2. Retrieve blocks corresponding to v indices from the file
      3. Get the message block, and perform concatenated encoding on the same.
         1. Perform striping
         2. Do concatenated outer encoding on the file. Keep the message part together and append parity part at the end.
         3. Perform concatenated inner encoding on this file.
   3. Send the computed response Mj and appended encrypted response Qj to the client

END

# Implementation Difficulties

## Memory Usage

Since all the implementation should be applicable on large files, the memory usage would be a major issue. For example, to perform a PRP on a 8GB file with 32 Byte each block, there are totally 2^28 blocks. To do the decoding, we need to store both PRP and reverse PRP table. So, there will be two 2^28 elements long arrays and the memory usage for that would be 2\*2^28\*8 = 4GB.

Also, for incremental encoding, to achieve linear ECC time, we need to increase the chunk size which we can read once. The ECC time would become quadratic if the file size is larger than the chunk size since we need to perform partial encoding each round when we read a chunk of the file. The fewer the chunk size is, the less efficient the incremental encoding would be.

## Uniqueness of Six-Round-Feistal PRP

Another issue we have encountered is the Six-Round-Feistal (SRF) cipher cannot generate unique mapping from one sequence to another, which does not satisfy the definition of PRP. We originally tried to use cycle walking to handle this situation so that we keep updating counters for each mapping and whenever there’s a duplicate, we perform the SRF cipher again hoping to get a different result. However, the efficiency of using cycle walking is very low especially for the last few elements. After cycle walking, there still is a high probability where the mapping is duplicated. In this case, we chose to use another method called linear probing. This is some method used in implementing hash table where hash value collision happens. So, whenever there’s a collision, we would start from that position and go to look for the next available slot. This method is also not very efficient when the table getting almost full, but the performance we evaluated is much better than cycle walking.

## Inner layer decoding

For inner layer decoding, the major difficulty is the storage of Di. Di stores every possible decoding result from inner layer decoding and thus is super large. (for alpha=10, it’s ten times as large as the file size). Also, since Di is a frequent accessed data structure, it would be too expensive to store in hard disk. To save the memory space, we used a Map-Reducer style counter for Di, where each decoded block is stored associated with a counter. Whenever we decode to get the block with the same index, we will compare with the one stored and increase the counter if it’s already there. Although this may increase the time, but it can save a huge amount of memory space required for inner layer decoding.

# Ideas for Improvements

By learning about POR protocol for a relatively long time, we have come up with some ideas to improve the performance of it.

* Parallel encoding/decoding: since the encoding and decoding procedure are not necessary to be sequential work. Also, we’ve observed that while running the encoder or decoder, for a CPU with four cores, only one of them is 100% utilized, others are all 0%. So, if parallel encoding/decoding is enabled, hopefully the performance can be four times faster than before.
* Adjusted SRF PRP: Now, the SRF PRP is not as efficient as we suppose and the major reason is that the cycle walking rate is too high to achieve uniqueness of the permutation. One reason for that would be we are not choosing the optimal value of n, a and b. Now we use the smallest possible value for those parameters to save memory space, however, to achieve higher efficiency, the tuning of those parameters is quite necessary.
* List decoding: The RScode library now can only correct up to d/2 errors for a chosen code distance. To achieve higher error correction rate than this upper bound, linear decoding would not be enough. In this case, list decoding is something more powerful to use. To use list decoding, instead of outputting only one decoded message, it can produce a list of possible messages. And then use maximum likelihood estimator to choose the most likely one.