Configuration Manual

MSc Research Project
Programme Name

Kahol Gaurav Bhuvanagiri Udayakumar
Student ID: 20157983

School of Computing
National College of Ireland

Supervisor: Dr. Arghir-Nicolae Moldovan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kahol Gaurav Bhuvanagiri Udayakumar
Student ID:	20157983
Programme:	MSc. In Cybersecurity **Year:** 2021-2022
Module:	Research Project
Lecturer:	Arghir-Nicolae Moldovan
Submission Due Date:	16/12/2021
Project Title:	Evaluation of XChaCha20-Poly1305 for Improved File System Level Encryption in the Cloud
Word Count:	1672 **Page Count:** 8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this Project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the Project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Kahol Gaurav Bhuvanagiri Udayakumar

Date: 15/12/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each Project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each Project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the Project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kahol Gaurav Bhuvanagiri Udayakumar
Student ID: 20157983

1 Summary

The research is being carried out as part of the evaluation of the XChaCha20 algorithm for Cloud file system encryption, with the goal of providing an alternate way of encryption to the AES256-bit algorithm. Encryption in the cloud may be divided into two categories.

File-System level encryption is a kind of encryption that takes place at the file system level, in which the files and directories of the user are encrypted by the Cloud vendor.

Encryption on the disc: This is a situation in which the whole storage volume is encrypted, and the encryption keys are reliant on a hardware Key Management Module, such as the TPM. The IO calls to decrypts and encrypts the file at the kernel level.

Since AES is used for both encryption techniques, the Project will first create XChaCha20 to conduct encryption on file system level and utilize generalized file formats such as multimedia, archive, and disc image as a first step toward creating an alternate algorithm for both encryption methods (ISO). Execution throughput calculations, as well as Entropy, Avalanche effect, correlation, keyspace analysis, and other metrics, are all included in the evaluation across AES software only implementation and XChacha20.

2 Technical Specification

2.1 Hardware Requirements

CPU: XEON E3 or E5-based Intel processors or higher / AMD RYZEN EPYC servers as the CPU processor
RAM: 8 GB of RAM (One of the metrics from the evaluation revealed that encryption was performed successfully with less RAM, but that the association between larger RAM and encryption tend to show way faster performance).
DISK: Solid-state drive (with a capacity of 60+ Gigabytes).

2.2 System Requirements

File System: EXT4 / NTFS / ZFS
NodeJS: V14+ (Code uses some core functionality, which is only available after Node v14)
NPM: V7.2+

Python: V3.6 and above and Google Collaboratory
Code editor: Preferably VSCode or NeoVim.

The Project is implemented in NodeJS, so it's easy to implement, and the encryption can be adopted in a wide range of distros or operating systems.

2.3 Environment setup

To implement, it's assumed the environment is Debian or Ubuntu-based distro for reproducing result.

2.3.1 Installing NodeJS:

1. Download the LTS Node binary for Linux from NodeJS official site https://nodejs.org/en/download/, copy the binary directory to Linux Path */usr/bin*
2. Check *Node -v* to see if the Node is installed
3. Along with NodeJS, its package manager NPM should be available to use, and this can be validated using the command `npm -v` command.

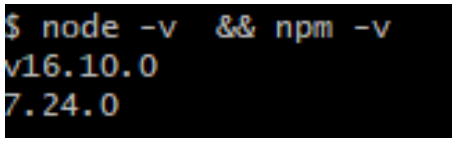

Fig 1: Node version check

2.3.2 Installing required Libraries:

1. Extract the artefacts to a directory and use cd directory to navigate to the project folder. The code repository is hosted in a personal Github repository and can be cloned from there as well.
2. Run the command *npm install* to download all the libraries mentioned in package.json.
3. Copy the test data files to respective parent directories in the Project.

2.3.3 Performing Encryption and Decryption with XChaCha20:

1. While encrypting the file, Key and nonce is configured to generate automatically and is saved to key file in JSON format in the project directory.
2. In general encryption strategy from a cloud perspective, a user should not be aware of Key, so a similar setup is stimulated here by serializing JSON file.
3. By running "npm test encrypt ${file_name}" will encrypt the file with XChaCha20 and create an encrypted file with .enc extension.
4. Example running "npm test encrypt baboon.png" will create an encrypted stream of baboon.png.enc and serialized key file.

5. By running "npm test decrypt baboon.png.enc" the Node module will generate the original PNG file by deserializing the JSON from the key file

6. For AES, the command for Encrypting and decrypting are as follows:
    - Encrypt – "npm test encryptAES baboon.png" (The command will generate an encrypted file with AES extension)
    - Decrypt – "npm test decryptAES baboon.png.aes"
7. The AES Encryption is software-only implementation [1].

2.3.4 Setting up cloud instances

All three machines use Ubuntu 18. Kernels are modified according to the cloud providers. To perform this test fairly machines with very similar clock speed is selected. One instance includes a local virtual box.

Instance 1 : The instance is categorized as M5.large in AWS with 2 Intel XEON vCPU and 8GB RAM

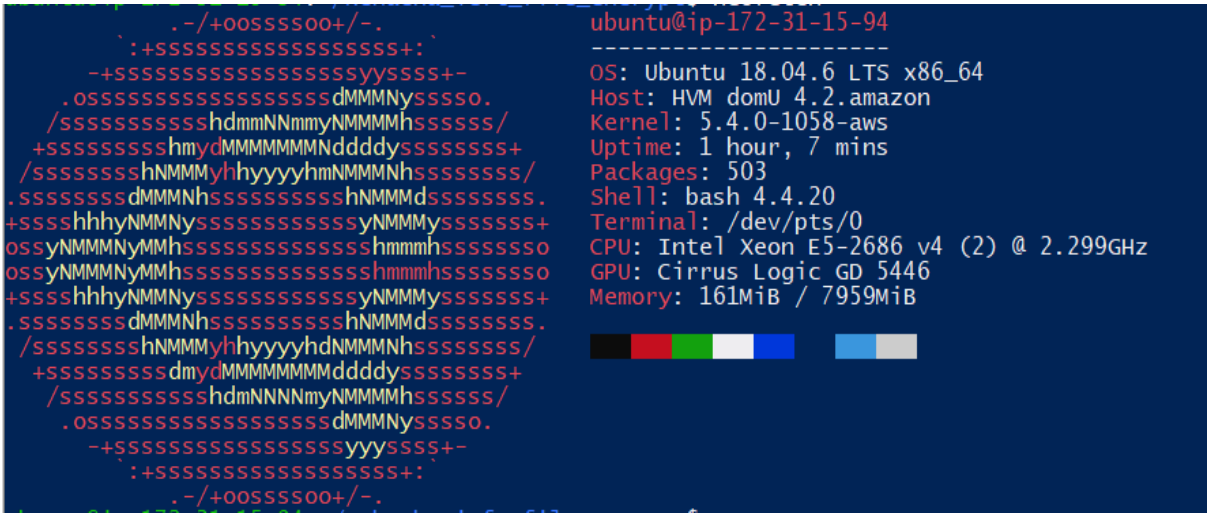

Fig 2: Neofetch result of Ubuntu 18 in AWS instance

Instance 2: The machine is categorized as D2 Version 4 in Microsoft Azure with 2 Intel XEON vCPU and 8GB of RAM

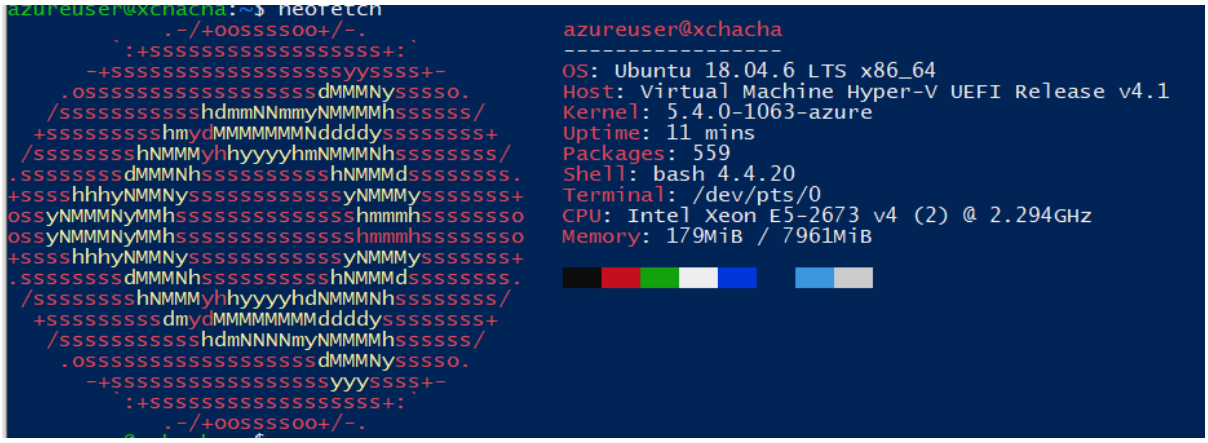

Fig 3: Neofetch result of Ubuntu 18 in Azure instance

Instance 3 : The Virtual machine is hosted in a virtual box with 2cores of i7 4th gen processors and 8GB RAM

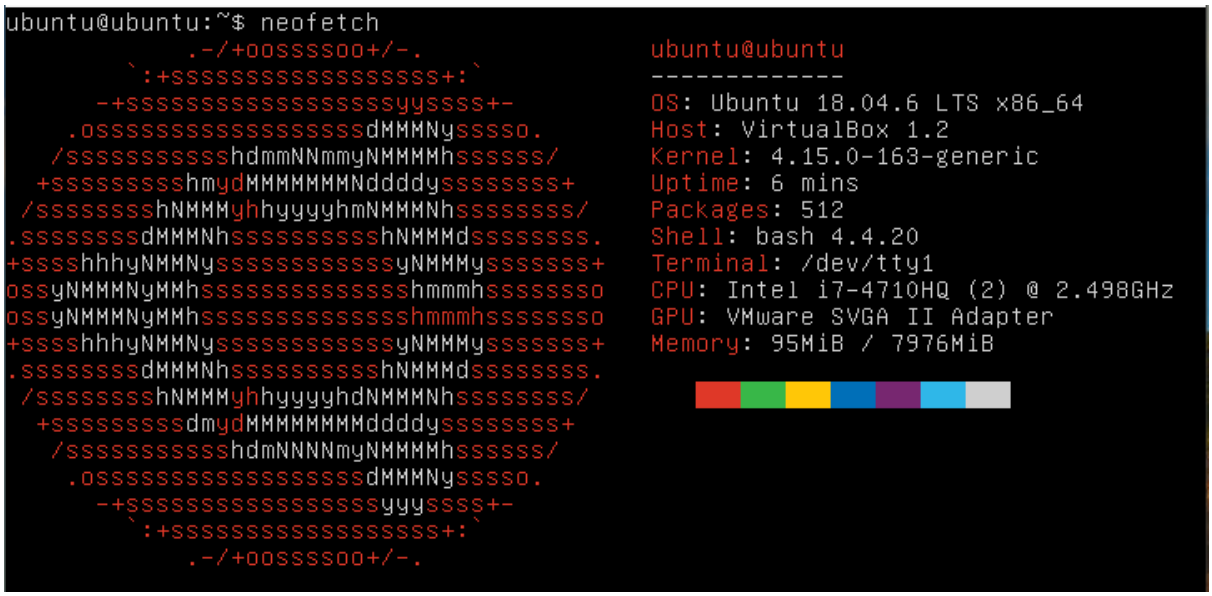

Fig 4: Neofetch result of Ubuntu 18 in Virtual box

To be a fair comparison, All three machines are running the same version of Node and use SSD based storage with a good range of IO bandwidth, though there is very minimal variation with a clock speed of CPU between cloud instances. The codebase was synced across all the cloud and local instances over SCP and SSH.

2.4 Reproduce Metrics mentioned in the Evaluation section:

2.4.1 Avalanche test:

To perform this encryption, Key must be persisted across all the datafiles, so in this case, a special function is defined inside *\src\Tests\avalanche.test.ts*. The path to the key file is defined and deserialized from prem. key and encryption is performed a byte of Key is increased by 1

bit, and another encryption is performed, and this generates a file with a completely different file hash

As a result, two files are produced for each data file, and this is done using both AES and XChacha20, with one file containing the original Key and the other holding an updated key. The purpose of an avalanche test is to identify the bits that are common and flipped between two files as a result of a key change. Consequently, a calculation function based on the ResearchGate debate is implemented in the research IPYNB notebook, where both data are uploaded as buffers, and the avalanche impact is determined [2].

2.4.2 Entropy:

1. An open-source codebase is used in the calculation of entropy.
2. The [3] code has been modified to display the average entropy value.
3. Entropy test is performed for three file category
    - Baboon
    - Linux kernel
    - Big Bunny

4. The pre-requisite library to generate the entropy in python3 are:
   "pip3 install numpy==1.17.2 Pillow==8.2.0"

5. The command to calculate entropy are as follows - "*python3 entropy.py ${filename}"*
   Example: *python3 entropy.py baboon.png*

6. Entropy is computed for both encrypted and unencrypted files.

2.4.3 Execution Time:

1. Throughput and normalization computations are performed across a variety of platforms, including Amazon Web Service (AWS), Microsoft Azure, and VirtualBox.

2. A benchmark. sh script was built to sequentially execute the tests in Linux, one after another. Throughput is computed, the runtime is measured and saved to a CSV file, and this script is provided within the artefact.

3. To execute the script, make sure all four unencrypted files is stored in the parent folder, and the following command must be executed "*bash benchmark.sh*", from the project directory

4. The output of the test is saved to a CSV file in the same directory

5. All three CSV files are converted to Python pandas data frames in an IPYNB file, which is then used to perform normalization and compare the results against one another.

```
for i in {1..5};do
filesToProcess=("baboon.png" "big_bunny_bucks.mp4" "linux-kernel.zip" "ubuntu-20.04.3-desktop-amd64.iso")
modesToProcess=("encrypt" "decrypt"  "encryptAES" "decryptAES" )

  for files in "${filesToProcess[@]}" ; do
    for modes in "${modesToProcess[@]}" ; do
        if  [[ $modes == de*AES ]] ;

        then
          processingFile="$files.aes"
        elif [[ $modes == de* ]] ;
        then
          processingFile="$files.enc"
        else

          processingFile=$files
        fi
        echo "Processing $processingFile with $modes";
        eval  "npm test $modes $processingFile" >> "test.csv"

    done
  done
done
```

Fig 5. Snippet of Benchmark script

6. The results and findings from the tests are Interpreted in the report section and IPYNB notebook.

2.4.4 Correlation

1. To generate a co-relation table for Ubuntu, Google Colab did not have sufficient RAM to compute correlation because of the file size, So Google Colab pro was used to generate a correlation table

2. Numpy is used to read the bytes from the file as it is faster to compute and can handle huge array size as shown in figure 6.

```
text1Buffer = n.fromfile('/content/drive/Shareddrives/Hoarding_Disk/temp3/Unecrypted_file/big_bunny_bucks.mp4',dtype="uint8")
text2Buffer = n.fromfile('/content/drive/MyDrive/Enc_files/big_bunny_bucks.mp4.enc',dtype="uint8")
text3Buffer = n.fromfile('/content/drive/MyDrive/Enc_files/big_bunny_bucks.mp4.aes', dtype="uint8")
```

Fig 6. Read Encrypted and unencrypted file from storage

3. Correlation is the linearity of two variables. To generate a correlation table, pandas are built-in. corr() is used to calculate the correlation between AES, XChaCha20 and Unencrypted file on Linux kernel, Baboon and big bunny buck. The code and table is shown in figure 7.

```
df = pd.DataFrame({'xchacha20':text2Buffer})
df['aes'] =  text3Buffer
df['unencrypted'] = text1Buffer
```

```
df.corr()
```

	xchacha20	aes	unencrypted
xchacha20	1.00000	-0.000280	-0.000330
aes	-0.00028	1.000000	-0.000092
unencrypted	-0.00033	-0.000092	1.000000

Fig 7. Correlation table with pandas

4. To generate graphs, it is important that the environment where the code is executed, like Jupytr notebook or Google collab has Matlotlib library installed.

5. A hexbin plot is plotted where the y-axis is the bytes of unencrypted file and x-axis is the bytes of the encrypted file,

6. The plots include XChaCha20 vs Unencrypted and AES vs Unencrypted for baboon and big bunny buck

2.4.5 Run time Analysis

1. This a metric used to evaluate CPU and memory usage during encryption/decryption, In order to perform this test, we need to enable a profiler on the Process ID of Node during encryption or decryption

2. Syrupy is an open-source GNU licensed python-based profile to export the CPU and memory usage to Log file, which can be imported to Excel for plotting the runtime.

3. To perform this test, A limited amount of memory and CPU is allocated to the Virtual box setting panel, and overall run time was calculated to see how the overall run time performed with:

4 CPU cores, 8GB RAM	3 CPU cores, 8GB RAM	2 CPU cores, 8GB RAM
4 CPU cores, 6GB RAM	3 CPU cores, 4GB RAM	2 CPU cores, 2GB RAM
4 CPU cores, 6GB RAM	3 CPU cores, 4GB RAM	2 CPU cores, 4GB RAM
4 CPU cores, 2GB RAM	3 CPU cores, 6GB RAM	2 CPU cores, 2GB RAM

References

[1]"A pure JavaScript implementation of the AES block cipher and all common modes of operation for node.js or web browsers.", GitHub, 2021. [Online]. Available: https://github.com/NikitaCartes-forks/aes-js. [Accessed: 16- Dec- 2021].

[2] "What is the avalanche effect in cryptography? How can we measure it ?", Researchgate, 2021. [Online]. Available: https://www.researchgate.net/post/What_is_the_avalanche_effect_in_cryptography_How_can_we_measure_it. [Accessed: 15- Dec- 2021].

[3]"Binary file entropy visualizer written in Python", GitHub, 2021. [Online]. Available: https://github.com/gcmartinelli/entroPy. [Accessed: 16- Dec- 2021].