

KeePassXCIPFS

Jayalshah

December 2019

CS 7960 Capstone Project

Fall 2019

Abstract

In the current digital world each and every digital account requires a credential in order to access it. Therefore it is a good security practice to use a password manager. A concern arises regarding storage of the vault on a central remote server .What if the remote server is down or has been attacked? In such cases our credentials are going to be compromised. To deal with it we will be using decentralization. Decentralization is forming the basis of large number of industries transforming and revolutionizing the world of application development. Decentralized networks provide an edge over the traditional Client-Server model, providing increased security, reliability as well as scalability. Apart from failing to eliminate censorship client-server architecture is also expensive requiring frequent maintenance. To overcome the drawbacks of centralized system we will use a decentralized storage.

Next concern arises on sharing credentials securely? One of the methods can be to encrypt the credentials with receivers public key and send it to them via secure channel and the receiver can decrypt it with their private credential. But, this creates a problem of scalability, as the solution mentioned is feasible on small scale but not in case of large number of users. In this scenario we will be making use of nucypher technology which can re-encrypt the data , saving time and resources as well as eliminating the risk of credential disclosure.

The aim of the project is to develop a decentralized password manager using keePassXC, IPFS ,NuCypher and Syncthing. IPFS acts as a decentralized storage medium, NuCypher which is used for Proxy re-encryption as well as Syncthing is used for Synchronization.

CHAPTERS

1	Introduction	5
1.1	KeePassXC	5
1.1.1	KeePassXC Community Fork	5
1.1.2	Configuration and Installation	9
1.2	Capstone project in brief	11
1.3	Programming Modules	15
1.3.1	C++	15
1.3.2	Python	15
2	Background on NuCypher and IPFS	16
2.1	IPFS Overview	16
2.2	NuCypher Overview	18
2.3	Decentralized Application introduction	20
2.4	Structure of DApp based on NuCypher	21
2.5	NuCypher Hackathon DApp Critiques	22
2.5.1	Hako	22
2.5.2	Snowden	23
2.5.3	Genobank DNA Wallet	24
3	KeePassXCIPFS	25
3.1	Decentralized and Replicated Vaults	25
3.2	Shared trust and multiple master keys	27
3.3	goals	27
4	Design of KeePassXCIPFS	29

4.1	Configuration and internals	29
4.2	sloccount, files and modules	30
4.3	Diagrams	39
4.3.1	Architecture	39
4.3.2	Dataflow	40
5	Implementation of KeePassXCIPFS	41
5.1	A 3 machine version of KeePassXCIPFS	41
5.2	Code level descriptions	41
5.3	Doxygen-ed	41
6	Evaluation of KeePassXCIPFS	41
6.1	SLOC KeePassXCIPFS	41
6.2	Static Analysis	41
6.3	Performance comparison	41
6.4	Issues Faced	41
7	Password Manager Comparison	41
8	Conclusion	47
9	Bibliography	49

1 Introduction

1.1 KeePassXC

1.1.1 KeePassXC Community Fork

Password Managers play the role of storing multiple passwords at a single place locked by a master password. There are multiple password managers available that differ from one another in the way they encrypt the data, type of storage as well as the supplementary features provided by the password manager. In our case we will be using KeePassXC version of the KeePass password manager. Lets get started with the original KeePass functions and features and then switch to KeePassXC. KeePass is the mostly widely used Open Source Software distributed under the terms of the GNU General Public License Version 2. KeePass is a small and light project. User simply needs to unpack from the Zip package. It also can be transferred in a USB stick with no additional configuration needed. KeePass is a project that once uninstalled from a computer, leaves no trace behind. So there is no way passwords and other data in the database can be found later. KeePass provides a secure, encrypted database where the user can store all of their credentials as well as notes.

The basic architecture of KeePass consists of a database, where the data of the user is divided into groups and subgroups. The unique master key or the master password can be a single string or a composition of a string and a Key file to unlock the database. Each and every subgroup consists of fields consisting of Title, Username, Password, URL as well as the recent date when the password was last modified or accessed. It is also capable for onetime key

creation using Transaction Authentication Numbers, that can be used once in the transaction. KeePass also supports random password generator as well as generating passwords requiring specific patterns. The password manager can be useful to a variety of users varying from simple end users to system administrators as well as advanced end users. KeePass has a timing requirement constrain in which a copied password remains in the memory just for 10 seconds after which the user needs to re-copy the password. KeePass makes use of two cryptographically strong encryption algorithms namely AES and Twofish having block size equal to 128 bits and a key size of 256 bits. SHA-256 is used for creating the key. Pseudo-random sources such as current tick count, performance counter, system date/time, mouse cursor position, memory status, message stack, process heap status, process startup information and several system information structures are used for generating Initialization vector as well as the Master key salt. The password manager has a very friendly user interface consisting of all the options under the menu bar. The saved data is displayed on the main database window. When a function is performed, the active window is the one performing the action and the main database window is inactive and cannot be accessed unless the current active window is closed. KeePass uses NET/Mono and Win32 (MS Windows) interfaces. If a USB containing the database is removed from a computer while changes haven't been completely saved, the database is damaged and cannot be opened. In this case the repair functionality can help by repairing KeePass database file from tools menu. In case the user forgets or loses the Master Password, the repair functionality is of no use. In case the header of the database, which is the first few bytes, is corrupted, the repair func-

tionality won't help. To avoid this kind of situations, back-ups can be done regularly.

After KeePass, let us switch to KeePassX which is a multi-platform fork of KeePass. The official KeePass on Mac and Linux requires Mono to run, But KeePassX uses Qt instead of .Net. Mono is a free and open-source project to create an Ecma standard-compliant .NET Framework-compatible software framework, including a C compiler and a Common Language runtime basically used to run Microsoft .NET applications cross-platform. On the other Qt is a free and open-source widget toolkit for creating GUI's as well as cross-platform applications with little or no change in the code. Originally KeePassX was called KeePass/L for Linux since it was a port of Windows password manager KeePass Password Safe. After KeePass/L became a cross platform application the name was recoined to KeePassX. Although both KeePass and KeePassx differ on the basis of user interface, still their databases can be used interchangeably as they are binary compatible. KeePassX can be simply installed in most of the versions of linux by simply typing `sudo apt-get install KeePassX` in the terminal, available in the form of in-built package. If it is absent in a specific version, it can be installed with the help of source code, by downloading the tarball containing the source code from the KeePassX website, followed by extracting the source code from archive, compiling the make file and then installing it. The basic features and attributes for the KeePassX are similar to the original KeePass. The password manager present in KeePassX can be setup in a way, binding the passwords with specific policies and specifying parameters for password generation. KeePassX and KeePass differ on the aspects on a large scale. The X variant appears

similar to other native programs as it is independent from mono. But like KeePass, KeePassX does not support any kind of plugins which can be a major drawback for the X variant. After going through the introduction about KeePass and KeePassX let us now look at the XC version referred to as KeePassXC, which is combined with other technologies in an attempt to decentralize it.

KeePassXC is forked from KeePassX. KeePassXC has an edge over both KeePass as well as KeePassX. KeePassXC was forked from KeePassX due to the slow-moving advancement of KeePassX as well as lack to merge many of the helpful pull requests failing to user expectation. Compared to KeePass, the XC version is written in C++ with the help of QT framework running natively across all available platforms. Some of the main features of the XC version include :

- Secure storage of passwords and other private data with AES(Advanced Encryption Standards).
- Cross-platform ability, makes it run on all available platforms without modifications.
- File format compatibility with KeePass2, KeePassX, MacPass, KeeWeb and many others (KDBX 3.1 and 4.0).
- SSH Agent integration.
- Auto-Type on all supported platforms for automatically filling in login forms.

- Key file and YubiKey challenge-response support for additional security.
- Time-based One Time Password generation.
- CSV import from other password managers, meaning that it can import to and export from programs storing data in tabular format .
- Command line interface.
- Stand-alone password and passphrase generator along with Password strength meter.
- Custom icons for database entries and download of website favicons. Database merge functionality along with automatic reload when the
- database is externally changed.
- Browser integration with KeePassXC-Browser for Google Chrome, Chromium, Vivaldi, as well as Mozilla Firefox.

1.1.2 Configuration and Installation

KeePassXC can either be directly installed as a package using the terminal based on the Linux distribution, the user is using or the zip file of the source code can be downloaded or cloned from the github repository available at <https://github.com/keepassxreboot/keepassxc>". KeePassXC has certain runtime requirements which were needed to be installed and are mentioned below in the steps I performed while installing KeePassXC as follows:

1. Downloading or cloning the KeePassXC github repository and extracting if the zip file is downloaded.
2. Now to build the source code we will be requiring cmake. If not available, Cmake can be easily installed from the terminal by typing `sudo apt install cmake`
3. Next, navigate to the KeePassXC folder followed by navigating to the directory containing CmakeLists.txt and type `cmake`.
4. As mentioned the building process does have certain runtime requirements, some of which can be directly installed from the terminal and for the rest need to download the file and install it manually. Files required during my installation of KeePassXC are mentioned below with the links from where they can be downloaded or commands to install them.
 - QT5 (`sudo apt install qttools5-dev`)
 - libgpg(<https://gnupg.org/download/index.html#libgpg-error>)
 - gcrypt(<https://gnupg.org/download/index.html#libgpg-error>)
 - Argon(`sudo apt install argon2-dev`)
 - zlib(`sudo apt install zlib1g-dev`)
 - QREencode(<https://fukuchi.org/works/qrencode/>)
 - Sodium(<https://download.libsodium.org/libsodium/releases>)
 - Qt5X11Extras(`sudo apt install libqt5x11extras5-dev`)

5. After Installing all the required libraries KeePassXC is built successfully
6. KeePassXC can now be accessed by navigating to the directory containing the built source file and running it.

1.2 Capstone project in brief

With escalation of technology and expansion of digital world brings in the need for secure online credentials. Multiple web login requires multiple credentials. Remembering complex security credentials can be quite tedious and therefore we can make use of password manager to store all of our credentials and just need to remember a single master password for the password manager or password vault. There are multiple password managers available that differ from one another in the way they encrypt the data, type of storage as well as the supplementary features provided by the password manager. Next in the queue is to overcome the drawbacks of client-server architecture as we cannot afford misuse of our credentials if the central server for password manager is breached. Although we can use an offline password manager but if the password manager works as a stand alone node we cannot access it from outside world. Therefore to overcome the limitations of centralized system we can make use of decentralization. Most of the web applications since the early days of Web are based on Client-Server architecture. But in recent times a shift has been seen from the traditional client-server model to Decentralized Web in order to overcome the limitations of the centralized model. The basic use of decentralization is to eliminate the central authority,

distributing it equally among all. Therefore, in decentralized architecture all the nodes have the equal authority unlike the client-server model in which the server is responsible for everything.

Decentralized web applications can hop the hurdles of the centralization such as eliminating censorship due to its distributed property and no central authority to censor the data, better privacy to its users as data now travels directly from one end point to another, more secure and trustable, eliminates the risk of single point of failure, along with these decentralization of the web also provides better efficiency, more reliability, cost effectiveness as well as robustness. Building Decentralized applications wraps all the perks of decentralization together accompanied by scalability and portability.

In our study we will be using KeepassXC password manager which is an open source community fork of KeePass Password Manager. The databases are encrypted using the best and most secure encryption algorithms currently known AES and Two fish. We will be using KeepassXC version of the keePass password manager which is an open source community fork of KeePass.

Current version of the KeepassXC password manager is yet to be decentralized. In our project we will be decentralizing the password manager using technologies such as IPFS(InterPlanetary File System), NuCypher(Re-encryption) as well as syncthing(Synchronization tool), which is discussed in depth in the sections below. IPFS is a protocol and network designed to create a content-addressable, peer-to-peer method of sharing hypermedia in a distributed system. IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of storage and sharing. IPFS is a p2p distributed system that seeks to connect

all computing devices . IPFS also provides the function of version control by taking files and managing them. It also stores them followed by tracking versions over time along with accounting for how those move across the network. For data to move around the IPFS network certain rules are needed to be followed. IPFS uses content addressing rather than location based addressing, which implies that content is addressed using names consisting of provider name, content name and version. As the content traverses through the network, routers and hosts either cache the location's where they can find a copy of the content or cache an actual copy of the content, enabling future requests for that content to be delivered by any node that currently holds a copy of the content. IPFS uses hash of the content to identify the content as hash can be said to be a representation of the content itself rather than creating an identifier and searching for the content, similar to search carried out in case of location based addressing. Peer-to-peer overlay in case of IPFS open the gates for super speed routing. Authentication in IPFS is supported by the Merkle DAG data structure which basically consists of tree of hashes. All the concepts are explained in more brief in the IPFS section. In our study IPFS will be used for storage of data.

In combination to IPFS we will also be using NuCypher which is a proxy re-encryption network. NuCypher is a decentralized Key Management System (KMS) that addresses the limitations of using consensus networks to securely store and manipulate private, encrypted data. It provides encryption and cryptographic access control, performed by a decentralized network, leveraging proxy re-encryption. NuCypher makes use of proxy re-encryption using which a proxy entity can change the encrypted data to receivers public-

key without getting access to the contents or plain-text of the data. The proxy re-encryption process is discussed further in the Nucypher section below. It also uses a spit-key threshold re-encryption known as umbral which eliminates the chances of loss of privacy as the data can never be decrypted and read in between. Precautions have also been taken in case of malfunctioning of the nodes or in a situation where nodes collude with each other. NuCypher consists of its own currency in the form of virtual tokens known as NuCypher Tokens which can be provided as an incentive to the miners who perform the services of re-encryption in the case of NuCypher. In our study NuCypher will also be used to support the new feature which will be added to the Decentralized application in which the user can share certain passwords with another trusted user. In conjunction we will use syncthing for synchronization purpose as once we securely share the password database we can synchronize changes thereafter.

The overall aim of the project is to develop a decentralized version of the KeePassXC password manager using decentralized technologies such as IPFS used for storage, combined with the service of proxy re-encryption provided by NuCypher as well as Syncthing as a synchronization tool. With the help of added group password function we can even share selected passwords. The technologies used and their working is given in more depth along with diagrams in the following sections.

1.3 Programming Modules

1.3.1 C++

C++ was required in modification of the password manager as KeePassXC is developed in C++ using QT framework. The new module of group password added required the following files from already existing project:

- Files from the project included - config, CustomData, Database, Group, Metadata, Passwordkey, FileDialog, KeePass2Reader(From KeePass), KeePass2Writer(From KeePass)Header and .cpp files
- Qt inbuilt modules - QApplication ,QObject , Qmessage ,Qpushbutton , QMessageBox, QitemModel

1.3.2 Python

- Python modules - os, Copy, logging, datetime, base64, maya, simpleob-server,
- NuCypher modules - Crypto, Umbral , Characters, Config, network, keystore

2 Background on NuCypher and IPFS

2.1 IPFS Overview

IPFS or the Interplanetary file system is a hypermedia distribution protocol, addressed by content and identities. IPFS enables the creation of completely distributed applications aiming to make the web faster, safer and more open. It tends to connect all the devices with same file system. Juan Benet, creator of IPFS claims it to provide high throughput content-addressed block storage model,consisting of content-addressed hyperlinks. IPFS works by providing a unique identity in the form of cryptographic hash to each and every file along with blocks within it. IPFS removes duplicacy by removing files with exact similar hashes. While searching for files, the network is asked to find the nodes storing the content that matches the hash. The content can also be retrieved in the form of human readable names using a decentralized naming system called IPNS. IPFS provides flexibility in accessing data being independent of low latency. The deduplication feature can help save lots of storage space,which can be utilized in version control to save more and more data. IPFS saves a lot of bandwidth along with providing secure content delivery. IPFS guarantees efficiency in performance at minimal cost. IPFS comprises of multiple properties combined together namely Distributed Hash Tables, BitTorrent block exchanges, Version Control Systems as well as well as Self-Certified Filesystems. Distributed hash tables play the role of distributed databases that are used to store and retrieve information based on key:value pair. Mapping is distributed among the nodes in way such as any modification results in minimal disruption. Properties such as scalability,

fault tolerance as well as autonomy makes DHT favorable. Distributed hash tables are governed by the CHORD protocol responsible for assignment of keys to the nodes on the network as well as recovering the value for a given key. In other words we can say that IPFS uses DHT for routing i.e to notify the network about the added data in addition to locating the data when requested by any node. Values smaller than 1KB are stored directly to the DHT and for the rest of the data references are stored by DHT resembling the Node Ids of the peer able to serve the block. IPFS is a single huge swarm of peers for versatile data where peers connect to share and exchange blocks they have as well as the blocks they are looking for. Next component in the row after Distributed Hash tables is BitSwap, used by IPFS as an exchange layer protocol. The basic difference between BitSwap and Bittorrent is that in BitSwap each and every node is requested for the content unlike Bittorrent where only peers looking for same file are traded with the block and all blocks being exchanged are from a single torrent. The purpose of bitswap is to request as well as send blocks to other peers in the network working as a marketplace for exchange of blocks. By marketplace we mean that Each peer participating in this marketplace has an internal strategy that they use to decide if they will exchange content (and other information) with any other peer they are contented to. Exchange of blocks by the peers is tracked in the BitSwap Ledger. IPFS also provides the ability of version control similar to Git. Version Control Systems models files that are modified over time with effective distribution of different versions. Merkle DAG object model captures the modifications to the file system tree. The versioning system basically comprises of 4 components blob,list,tree,commit. Blobs are free from

any links consisting of data generally representing files. Lists also define files but a list consists of multiple blocks. We can say that Tree plays the role of directory in IPFS. While commit refers to a snapshot of the history in a tree. The links in IPFS are in the form of hashes which means that the data is immutable. Labels or Pointers are used to point towards the immutable content. The labels in order can be used to represent the current or the latest version of the object. Therefore, version changes can only update references or add objects. The labels or the pointers are created using Self-Certified Filesystems(SFS). In other words we can say that SFS are used to address remote filesystems. Changing an object results in change of its Hash and indeed its address, so IPFS uses InterPlanetary naming System(IPNS) to provide an permanent address along with more human-readable favorable form. IPNS uses SFS in which name refers to hash of the public key while records are signed by private key and distributed which on retrieval can be decrypted with public key and authenticity can be verified. IPNS records are announced and resolved with the help of distributed hash tables. IPFS will be used as a decentralized storage in our application.

2.2 NuCypher Overview

NuCypher is a proxy re-encryption network for empowering privacy in decentralized network as well as addressing the limitations of using consensus network of manipulating private data. The network facilitates end-to-end encrypted data sharing for distributed applications and protocols. NuCypher will be an essential part of decentralized applications, just as SSL/TLS is an essential part of every secure web application. Using Umbral threshold proxy

re-encryption NuCypher provides Cryptographic access controls on the decentralized network. Before jumping on to Umbral threshold lets delve into the working of NuCypher using an example using actors. Alice as the data owner, Bob as the data recipient as well as an entity as a 3rd party.

Firstly, the data owner, grants access to her encrypted data to a 3rd party entity willing for the data by creating a policy and uploading it to the NuCypher network.

Secondly, a 3rd party entity can encrypt data on Alice's behalf using Alice's public key. The resulting encrypted data can be uploaded to a decentralized storage layer such as IPFS or Swarm.

A group of Ursulas, which are nodes of the NuCypher network, receive the access policy and re-encrypt data in exchange for payment in fees and token rewards. Proxy re-encryption makes sure Ursulas and the storage layer never have access to Alice's plaintext data.

Bob, a data recipient, sends an access request to the NuCypher network. If the policy is satisfied, the data is re-encrypted to his public key and he can decrypt it with his private key.

NuCypher uses proxy re-encryption which allows a proxy entity to transform ciphertext from one public key to another without gaining any knowledge about the underlying message. Overcoming the drawbacks of Public-key Encryption protocols, proxy re-encryption can be used for N-to-N communication with random number of data producers and consumers, where the identity of the data recipient can be unknown at the beginning. The working of re-encryption is explained in the next chapter.

2.3 Decentralized Application introduction

Decentralized applications can be broadly divided into 3 categories based on the type of blockchain model used namely Type I, Type II and Type III dApps. Type I dApps consist of their own blockchain such as Ethereum and Bitcoin. Type II includes those applications that make use of Type I applications basically consisting of protocols having tokens necessary for their functioning such as 0x protocol which uses ethereum to power applications. The last category of dApps include those applications that make use of Type II dapps such as DDEX which operates on 0x protocol.

An application needs to meet certain criteria in order to be considered as decentralized application which are as follows:

Application needs to be completely open source so that the users can trust the application and can have complete access to their data .

The data and records of the application must be stored cryptographically in order to enhance security.

The Decentralized application must have its own token system working as an internal currency in order to monetize the dApp and provide an incentive to the developer.

The Application should be self-dependent to generate its own token along with having an inbuilt consensus mechanism ,of which the smart contracts can take care of.

In order to develop a decentralized application a chain of steps need to be followed starting with publishing the white paper including the features of the dApp along with a working prototype. Publishing the paper is followed by initial token sale and spreading ownership stake of the dApp. Final

step includes investment of funds in the development of the dApp followed by deployment of the decentralized application. In this study we will be discussing some of the decentralized applications, mainly emphasizing on a decentralized password manager. Password managers add value to the security posture Password managers allow the storage and retrieval of sensitive information from an encrypted database. Users rely on them to provide better security guarantees against trivial exfiltration than alternative ways of storing passwords.

2.4 Structure of DApp based on NuCypher

A decentralized application uses NuCypher for secrets management as well as dynamic access control. With the help of NuCypher the sender or the person granting the permissions can revoke them whenever he wants. The typical structure of a DApp based on NuCypher is shown in the image below. The user creates a policy and uploads it to NuCYpher network. The user encrypts the application data and stores it on IPFS or a decentralized storage. Whenever receiver wishes to get the data they download it from decentralized storage and asks the sender to provide access. The sender then grants the receiver permission using receiver's public key. after which the receiver can ask the Nucypher nodes to re-encrypt the data base on receivers public key and can be decrypted by the user with the help of their private key.

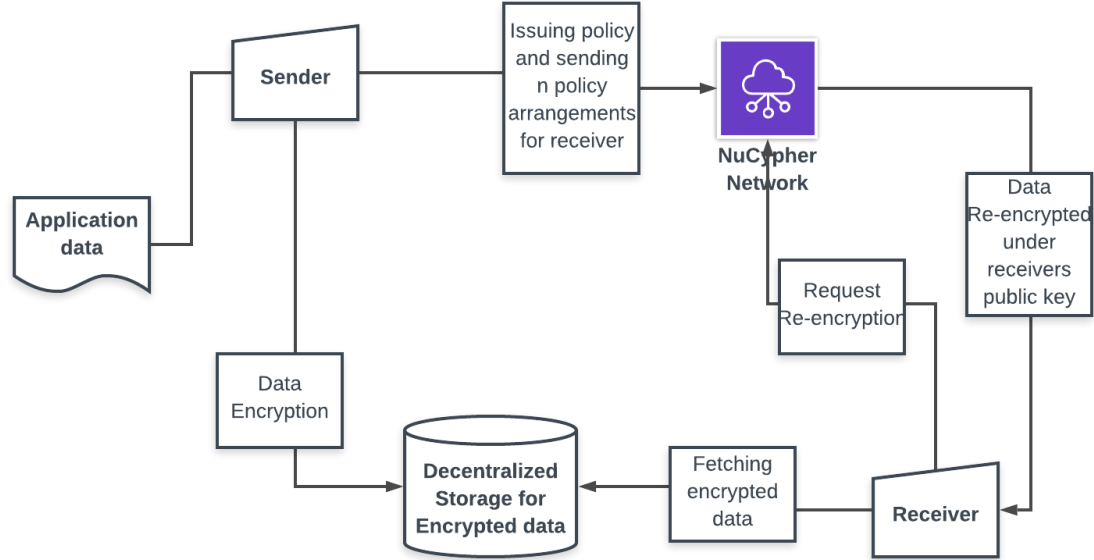


Figure 1: Structure of DApp based on NuCypher

2.5 NuCypher Hackathon DApp Critiques

2.5.1 Hako

Hako is a decentralized file sharing application powered by NuCypher and IPFS. It uses a the proxy re-encryption scheme provided by NuCypher in order to secure and share data. The good part about it is that the user is always in control of credentials unlike centralized file shraing services such as Google Drive and Dropbox. Decentralization provided by IPFS also makes it resistant to single point of failure. Usage of NuCypher also saves encryption as well as computational run time making the DApp more favorable. The hako code can still be modified by embedding the functionalities of NCIPFS library in the hako code itself making it completely independent.It basically

handles NUCID'S and futari's(Users) NCIPFS library combines CID which is the content identifier for the data, the policy key created by the user for encryption as well as filename and encryption key. Therefore, on overall basis we can say that Hako is a very useful DApp which has the power to replace centralized file sharing applications providing security and transparency to its users.

2.5.2 Snowden

Snowden is a google chrome extension which encodes the posts into emojis granting access to all your friends . Nobody except them could read and understand what you want to share in your news feed. The privacy is controlled by the user themselves. The extension makes use of the nucypher technology. The text message is encrypted in the form of emojis and with the help of proxy re-encryption the message gets re-encrypted and the receiver can then decrypt with the help of their private key. The application is is a good way to maintain privacy without getting censored but the app could have been more better if certain changes are made to the application. Firstly, using a third party platform is not a good idea as what if the platform providing server fails, instead on can use a decentralized storage from where the receivers can access the data.Secondly, the documentation of the application needs to be more clear and precise. Next certain error messages should not take place when the script is executed again such as regarding the installation of nucypher as well as moving of com.snowden.connect.json file.The instructions should include as to what changes are needed to be made in the respective files. The main bug in the application is running of chrome

extension even after terminating or stopping the services of nucypher. Overall the idea behind the implementation is pretty good but it needs a bit of improvements.

2.5.3 Genobank DNA Wallet

Genobank DNA Wallet is a Decentralised and anonymous DNA wallet. Enables users to extract DNA, establish ownership, share (P2P) control using Nucypher and Blockchain. It combines an FDA approved DNA kit with NuCypher and Blockchain to enable an anonymous and encrypted way to interact with the World's Genomic ecosystem with privacy control. The main issue with the application is that it uses a docker container which restricts it to certain configurations. The developed application should be able to work with specified configuration and configurations above them. Unlike that the rest of the application works well with easy to use user interface built using solidity.

3 KeePassXCIPFS

3.1 Decentralized and Replicated Vaults

KeePassXCIPFS is a decentralized password manager developed by combining KeePassXC, IPFS and NuCypher. It has been modified to overcome some of the limitations of older version of KeePassXC such as sharing of selected passwords as well as vault replication ,secure sharing of credentials and synchronization. In the earlier version of keepassxc, a way to access the password manager was to upload the database to a central server such as dropbox or google drive and access the data whenever needed but in this case concern related to single point of failure arises as if due to certain reasons if cloud service is not available the user would not be able to access the database. In case of the modified password manager single point of failure does not exist. The decentralization benefit provided by IPFS helps to eliminate single point of failure as multiple nodes would be consisting of copies of the password database and even if a few nodes go out of service the network will still have the database. IPFS stores the database in the form of hash as it supports content based search rather than location based search. IPFS therefore provides the functionality of decentralized storage. The data stored on IPFS is encrypted with the help of Nucypher. With the help of nucypher the data sender can grant as well as revoke access at will. The NuCypher network provides accessible, intuitive, and extensible runtimes and interfaces for secrets management and dynamic access control. Traditionally, sharing encrypted data requires the sender to trust the receiver with the decryption key. If the decryption key was leaked in any part of the data sharing process, then

anyone with the key can decrypt the files leading to data breach. NuCypher provides an encryption layer that enables users to move data storage to the cloud securely. Cloud services providers in our case IPFS only have access to the encrypted data while the decryption keys are always stored on the nodes with the users. PRE technology allows users to delegate access to both internal and external users scalable while retaining complete control of the data. The nucypher network in our project helps to share the database with multiple users securely. The sender generates a policy public key using a random label. The database is then encrypted with policy public key and uploaded to IPFS. The sender then needs to grant access to receiver with the help of receiver's public key. The sender then sends the policy info to the receiver through a secure side channel. On the receiver's side the receiver can then download the password database from IPFS. The downloaded database is still encrypted and in order to get access the receiver needs to request the nucypher network to re-encrypt the database so that the receiver can decrypt with their own private key. In this process the private credentials never enter the network at any point of time. In future if the sender needs to send the password database to another user he does not need to encrypt the whole thing again he just needs to authorize the new user by creating a new policy info with their public key and send the policy info to new receiver. The new receiver can go through the same process in order to decrypt the password database. In case of multiple users it provides a feasible solution by providing scalability as well as saves resources and processing power. In absence of Password manager the combination of IPFS and NuCypher can also provide the functionality for a decentralized dropbox. In our case using IPFS and

NuCypher can create multiple and replicated vaults over the network.

3.2 Shared trust and multiple master keys

KeePassXCIPFS provides the same functionality that the KeePassXC password manager provided but with an additional feature. The password manager was modified to share selected passwords. The sender can export a group of selected passwords locked with a new password and saved as a new password database or a file. The exported set of passwords locked with a new password can be shared with the receiver with the help of IPFS and NuCypher network described in the above section. The sender needs to share public credentials with the receiver using a secure side channel. The receiver after decrypting the re-encrypted message with their private key can import the received database. The password for the locked group of passwords will be provided by the sender which can be changed after importing it once by the receiver. The receiver can either import the received list of passwords in a new database or in an already existing password database. Therefore we can say that there will be multiple master keys for the same set of passwords for multiple users. The group password feature is as depicted in the images below.

3.3 goals

Functional behavior of KeePassXCIPFS limited to what was in KeePassXC must be identical to keePassXC.

The older version of KeePassXC password manager was modified and a new functionality was added of sharing selected user passwords. The func-

tional behavior of the password manager still remains unchanged. The events which used to occur from beginning phase of unlocking an existing database or creating a new one to the end representing the save and close state of the password manager still remain unchanged.

2. Speed of transaction as close to KeePassXC as possible.

The working mechanism as well as the speed of the modified password manager remains almost the same on the host side, but when IPFS and NuCypher comes into play it takes some amount of time for the node to discover the hash content and for the nuCypher nodes to re-encrypt the data based on receivers public key. Since NuCypher works on localhost in our project as it requires virtual currency to perform re-encryption on external network and therefore during the testing phase I was required to perform tests with combined module of KeePassXC, IPFS and NuCypher as well as isolated IPFS and the password database. In the first case when all the 3 were used together the process was completed in couple of seconds as the upload file was already present with the IPFS gateway and the ursula nodes were local to. In second case with KeePassXC and IPFS, I used SSH to connect to two remote nodes and added password databases using their IPFS gateway. When I tried to get the content using IPFS gateway of my node it took 6.4 minutes to find the first password database and 4.81 to find the second database.

3. Reliability? KeePassXCIPFS is completely reliable.

The original version of the password manager was already considered very secure as it uses AES encryption algorithm with 256 bit key. Along with that we are using IPFS and NuCypher which provide availability as well as confidentiality and

authentication respectively. Since IPFS is decentralized it eliminates single point of failure as well as server side attacks enhancing availability. With the help of Keys and encryption NuCypher protects the data as well as verify the identity of the sender. The private credentials of neither the sender nor the receiver are ever needed to leave their premises and remain with the users themselves.

4. Feature set of KeePassXCIPFS larger than KeePassXC The feature set of KeePassXCIPFS include a feature for sharing selected passwords known as group password . It also consists a remote decentralized storage layer as well as well as secure sharing of password databases. Lastly it also uses an already developed Synchronization tool known as Syncthing for Synchronization if the sender and the receiver wish to. face

4 Design of KeePassXCIPFS

4.1 Configuration and internals

The configuration needed for successfully running KeePassXCIPFS is basically divided into 4 parts. The 4 different parts include configuring KeePassXC which is our password manager, IPFS , NuCypher as well as Syncthing. Detailed configuration and steps for KeePassXC are provided in the above sections. In brief first we need to download and install required libraries and then build the program using cmake or using automatic installer provided with the password manager. Next we need to configure IPFS. The IPFS package is provided with modified libraries as the IPFSapi module was needed to be modified in order to work. Then the user needs to install and

configure go. For installing go the user needs to download official binaries (<https://golang.org/doc/install>) and then set the path as mentioned.

Place the go sources in a "go" directory inside your home directory:

```
❯ export GOPATH= /go
```

This is needed for the go toolchain:

```
❯ export GOBIN="$GOPATH/bin"
```

Make sure that our shell finds the go binaries:

```
❯ export PATH="$GOPATH/bin:$PATH"
```

Next we need to set the IPFS gateway which will be used for uploading and downloading file to decentralized storage. An IPFS Gateway acts as a bridge between traditional web browsers and IPFS. Through the gateway, users can browse files and websites stored in IPFS as if they were stored in a traditional web server. We can use the gateway after starting the IPFS daemon. Go is needed to be configured correctly as without proper configuration of Go the IPFS gateway won't work. Next we need to clone or download the NuCypher module and set up a node. We then need to connect the node to ethereum network in order to work on the external nucypher network. We just need to change the provide-uri as ethereum nodes address in the client.py file. At last we need to download the syncthing binaries and add peers the user wishes to and synchronize the sharing folder.

4.2 sloccount, files and modules

Combined Sloccount of the project

Keepassxc

sloccount keepassxc

Have a non-directory at the top, so creating directory top_dir

Adding /home/jerry/Downloads/keepassxc/CHANGELOG.md to top_dir

Adding /home/jerry/Downloads/keepassxc/CMakeLists.txt to top_dir

Adding /home/jerry/Downloads/keepassxc/CMakeLists.txt.user to top_dir

Adding /home/jerry/Downloads/keepassxc/COPYING to top_dir

Adding /home/jerry/Downloads/keepassxc/INSTALL.md to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.BOOST-1.0 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.BSD to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.CC0 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.GPL-2 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.GPL-3 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.LGPL-2.1 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.LGPL-3 to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.MIT to top_dir

Adding /home/jerry/Downloads/keepassxc/LICENSE.NOKIA-LGPL-EXCEPTION to top_dir

Adding /home/jerry/Downloads/keepassxc/README.md to top_dir

Creating filelist for build

Creating filelist for cmake

Creating filelist for docs

Adding /home/jerry/Downloads/keepassxc/release-tool to top_dir

Creating filelist for share

Creating filelist for snap

Adding /home/jerry/Downloads/keepassxc/sonar-project.properties to top_dir

Creating filelist for tests

Creating filelist for utils
Have a non-directory at the top, so creating directory src_top_dir
Adding /home/jerry/Downloads/keepassxc/src/CMakeLists.txt to src_top_dir
Creating filelist for src_Group Password
Creating filelist for src_autotype
Creating filelist for src_browser
Creating filelist for src_cli
Adding /home/jerry/Downloads/keepassxc/src/config-keepassx.h.cmake to s
Creating filelist for src_core
Creating filelist for src_crypto
Creating filelist for src_fdosecrets
Creating filelist for src_format
Adding /home/jerry/Downloads/keepassxc/src/git-info.h.cmake to src_top_
Creating filelist for src_gui
Creating filelist for src_keys
Adding /home/jerry/Downloads/keepassxc/src/main.cpp to src_top_dir
Creating filelist for src_proxy
Creating filelist for src_qrcode
Creating filelist for src_sshagent
Creating filelist for src_streams
Creating filelist for src_totp
Creating filelist for src_touchid
Creating filelist for src_updatecheck
Creating filelist for src_zxcvbn
Categorizing files.

Finding a working MD5 command....

Found a working MD5 command.

Computing results.

c_count ERROR - terminated in string in /home/jerry/Downloads/keepassxc/

SLOC	Directory	SLOC-by-Language (Sorted)
35866	build	cpp=35337,ansic=529
26267	src_zxcvbn	ansic=26267
17747	src_gui	cpp=17724,xml=23
14458	tests	cpp=13569,xml=635,javascript=254
9531	src_core	cpp=9531
5549	src_format	cpp=5549
3502	src_browser	cpp=3502
3407	src_fdosecrets	cpp=3407
3234	src_autotype	cpp=3178,python=56
3136	src_crypto	cpp=2502,ansic=634
2733	src_Group Password	cpp=2733
2354	src_cli	cpp=2354
1154	src_streams	cpp=1154
1028	top_dir	sh=1028
934	src_keys	cpp=934
705	share	xml=645,sh=60
678	src_sshagent	cpp=678
308	src_totp	cpp=308

218	utils	sh=147,python=71
179	src_proxy	cpp=179
143	src_qrcode	cpp=143
141	src_updatecheck	cpp=141
112	src_top_dir	cpp=112
29	src_touchid	cpp=29
7	snap	sh=7
0	cmake	(none)
0	docs	(none)

Totals grouped by language (dominant language first):

cpp:	103064 (77.25%)
ansic:	27430 (20.56%)
xml:	1303 (0.98%)
sh:	1242 (0.93%)
javascript:	254 (0.19%)
python:	127 (0.10%)

Total Physical Source Lines of Code (SLOC) = 133,420

Development Effort Estimate, Person-Years (Person-Months)=34.08(408.97)

(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))

Schedule Estimate , Years (Months)

= 2.05 (24.57)

(Basic COCOMO model, Months = 2.5 * (person-months**0.38))

Estimated Average Number of Developers (Effort/Schedule)

= 16.65

Total Estimated Cost to Develop

= \$ 4,603,876

(average salary = \$56,286/year, overhead = 2.40).

SLOCCount, Copyright (C) 2001–2004 David A. Wheeler

SLOCCount is Open Source Software/Free Software, licensed under the GNU

SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to

redistribute it under certain conditions as specified by the GNU GPL lic

see the documentation for details.

Please credit this data as "generated using David A. Wheeler's 'SLOCCou

Group Password

sloccount 'Group Password'

Creating filelist for Group Password

Categorizing files.

Finding a working MD5 command....

Found a working MD5 command.

Computing results.

SLOC Directory SLOC-by-Language (Sorted)

2733 Group Password cpp =2733

Totals grouped by language (dominant language first):

cpp:2733 (100.00%)

Total Physical Source Lines of Code (SLOC)

Development Effort Estimate, Person-Years (Person-Months) = 0.57 (6.90)

(Basic COCOMO model, Person-Months = $2.4 * (KSLOC^{**}1.05)$)

Schedule Estimate, Years (Months)

(Basic COCOMO model, Months = $2.5 * (person-months^{**}0.38)$)

Estimated Average Number of Developers (Effort/Schedule) = 1.32

Total Estimated Cost to Develop

(average salary = \$56,286/year, overhead = 2.40).

SLOCCount, Copyright (C) 2001–2004 David A. Wheeler

SLOCCount is Open Source Software/Free Software, licensed under the GNU

SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to

redistribute it under certain conditions as specified by the GNU GPL lic

see the documentation for details.

Please credit this data as "generated using David A. Wheeler's 'SLOCCou

Nucypher +IPFS

Creating filelist for Nucypher+IPFS

Categorizing files.

Finding a working MD5 command....

Found a working MD5 command.

Computing results.

SLOC	Directory	SLOC-by-Language (Sorted)
1694	Nucypher+IPFS	python=1694

Totals grouped by language (dominant language first):

python: 1694 (100.00%)

Total Physical Source Lines of Code (SLOC)

= 1,694

Development Effort Estimate, Person-Years (Person-Months) = 0.35 (4.17)

(Basic COCOMO model, Person-Months = $2.4 * (KSLOC ** 1.05)$)

Schedule Estimate, Years (Months)

= 0.36 (4.30)

(Basic COCOMO model, Months = $2.5 * (person-months ** 0.38)$)

Estimated Average Number of Developers (Effort/Schedule)

= 0.97

Total Estimated Cost to Develop

= \$ 46,989

(average salary = \$56,286/year, overhead = 2.40).

SLOCCount, Copyright (C) 2001–2004 David A. Wheeler

SLOCCount is Open Source Software/Free Software, licensed under the GNU

SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to

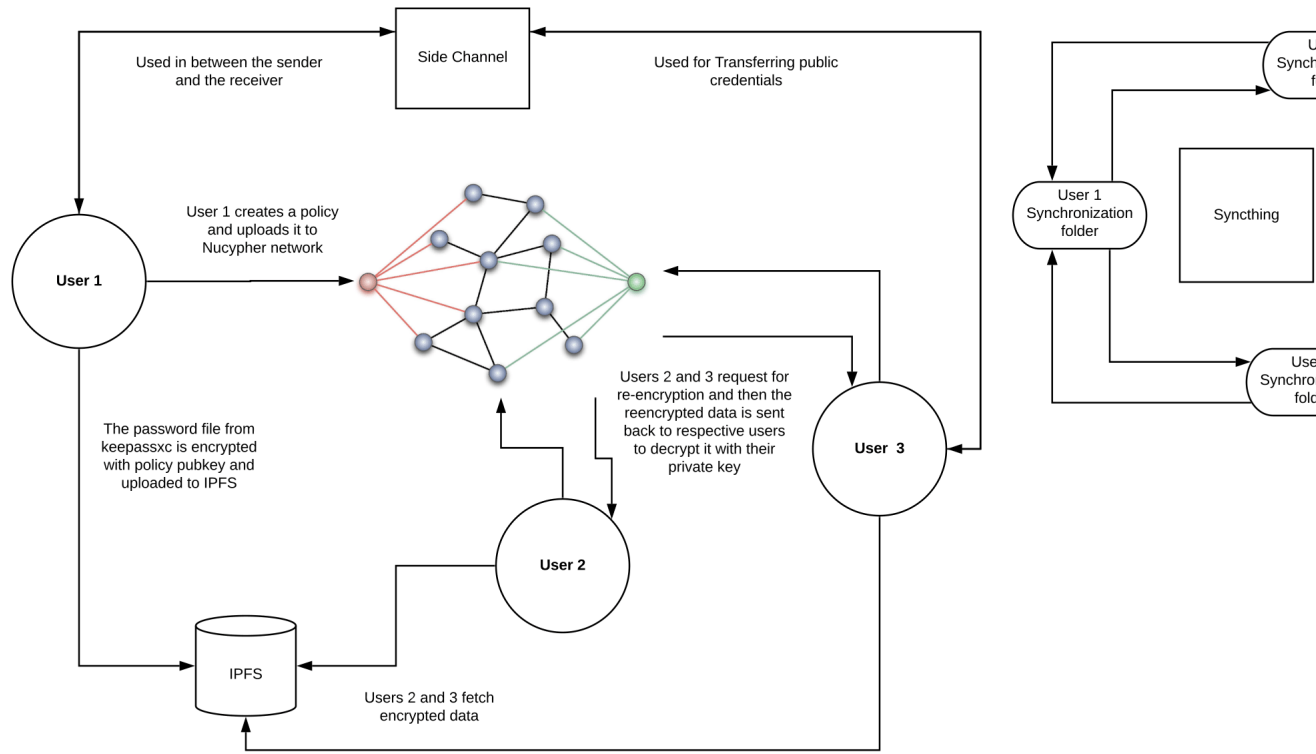
redistribute it under certain conditions as specified by the GNU GPL lic

see the documentation for details.

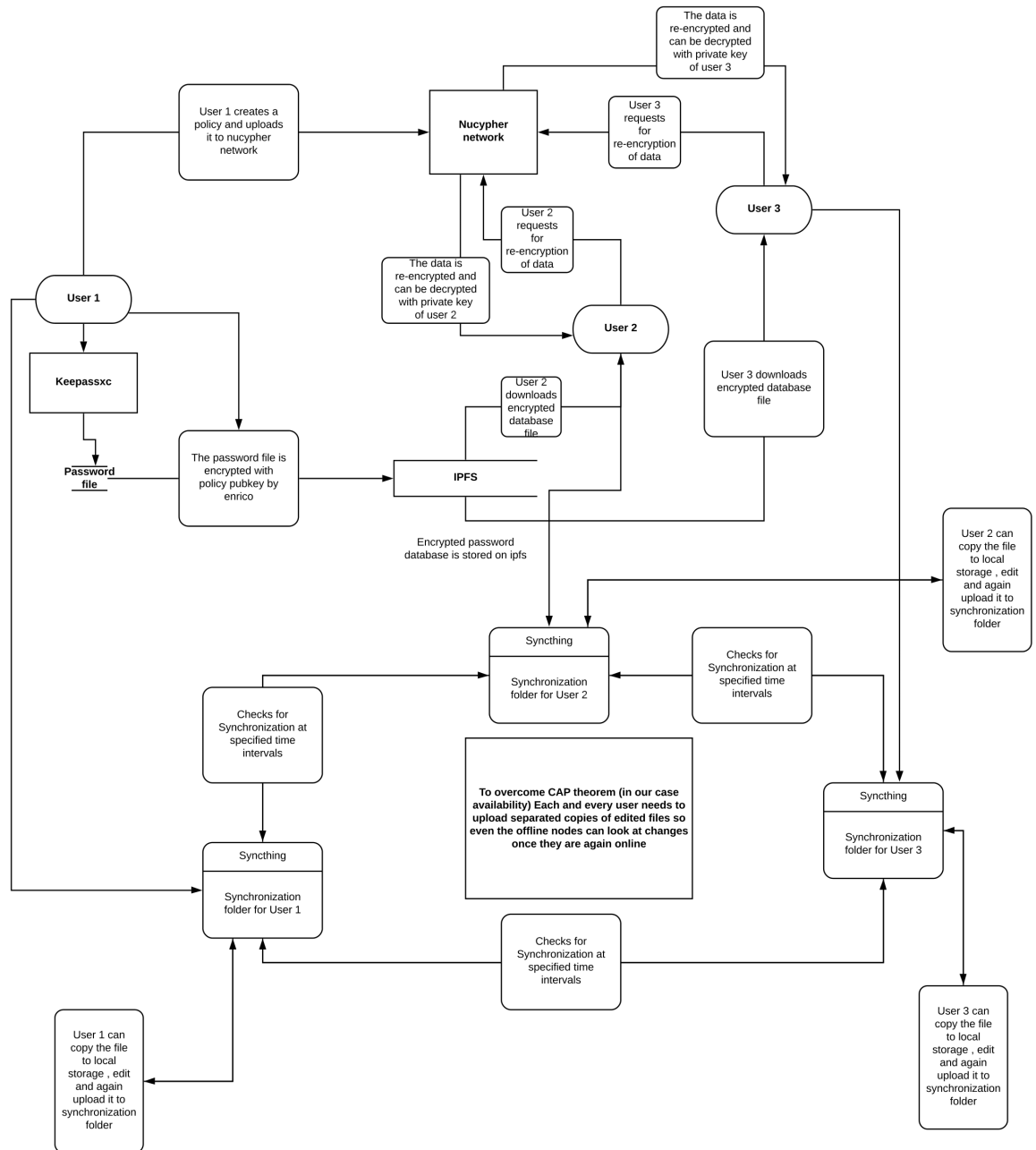
Please credit this data as "generated using David A. Wheeler's 'SLOCCou

4.3 Diagrams

4.3.1 Architecture



4.3.2 Dataflow



5 Implementation of KeePassXCIPFS

5.1 A 3 machine version of KeePassXCIPFS

5.2 Code level descriptions

5.3 Doxygen-ed

6 Evaluation of KeePassXCIPFS

6.1 SLOC KeePassXCIPFS

6.2 Static Analysis

6.3 Performance comparison

6.4 Issues Faced

7 Password Manager Comparison

LastPass

LastPass is a freemium password manager that stores encrypted passwords online. The standard version consists of a web interface, including plugins for multiple web browsers. User content in LastPass, includes passwords and secure notes protected by a master password. The content is synchronized to any device the user uses the LastPass software or app extensions on. Information is encrypted with AES-256 encryption with PBKDF2 key derivation as well as SHA-256 hashing algorithm mixed with salts . It

also has the ability to increase password iterations value. Encryption and decryption takes place at the device level. LastPass consists of following features:

- Form filler that automates password entering and form filling.
- Strong password generation
- Site sharing and Site logging
- Two-factor authentication.
- LastPass offers a user-set password hint, allowing access when the master password is missing.

```
jerry@79:~$ sloc lastpass
----- Result -----
      Physical : 15582
      Source   : 11380
      Comment  : 2311
Single-line comment : 15
      Block comment : 2296
      Mixed      : 44
Empty block comment : 0
      Empty      : 1935
      To Do     : 2

Number of files read : 70
-----
jerry@79:~$ █
```

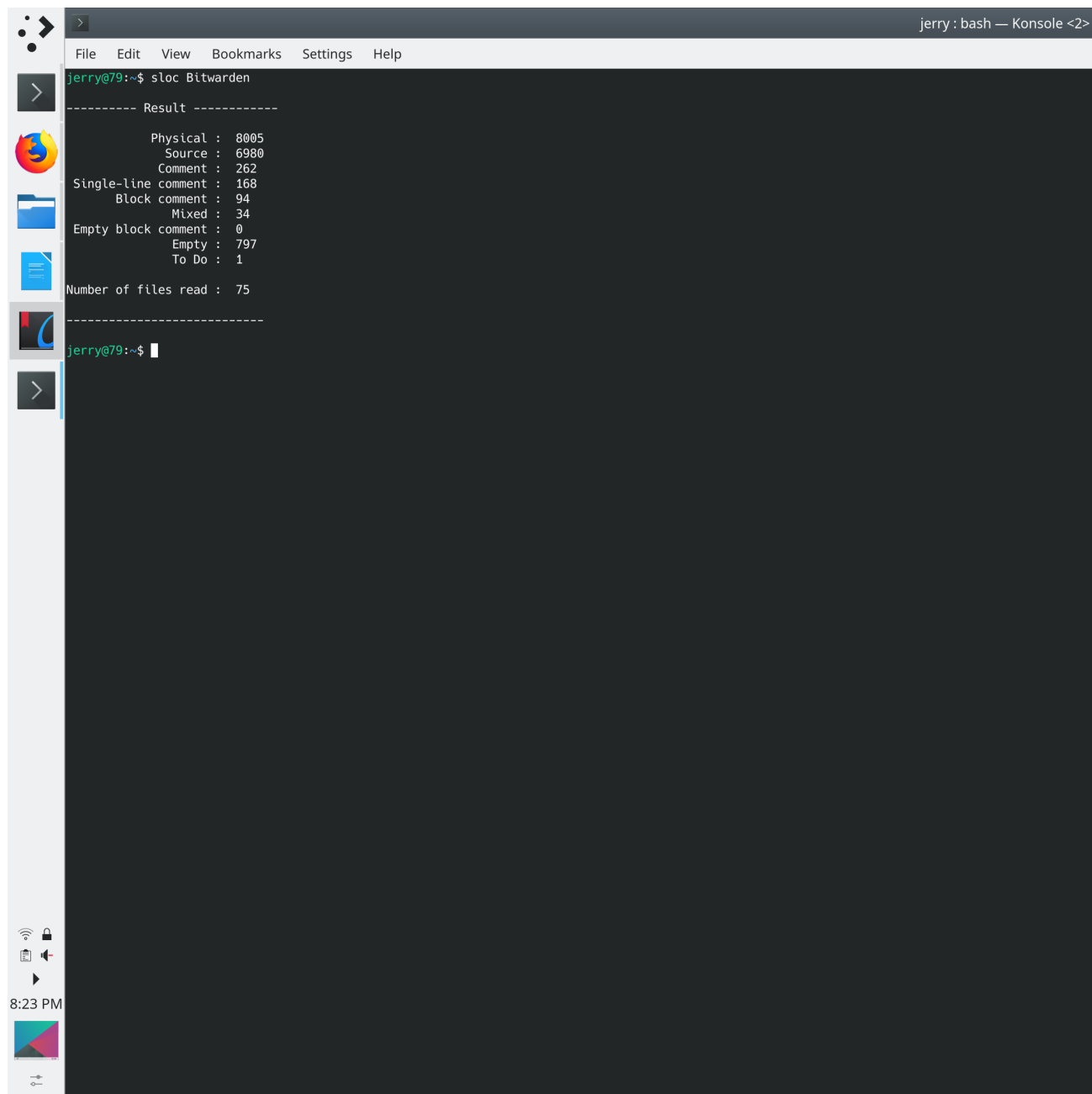
Bitwarden

Bitwarden is a free and open-source password management service that stores credentials in an encrypted vault. The Bitwarden platform offers a variety of client applications including a web interface, desktop applications,

browser extension, mobile apps as well as a command line interface. Bitwarden offers a cloud-hosted service as well as the ability to deploy the solution on host itself.

Bitwarden provides the following features:

- Cloud-synchronization.
- Items types such as Logins, Secure Notes, Credit Cards, and Identities.
- Password generator
- Two-factor Authentication.
- File attachments
- TOTP
- Data breach reports and password exposure checks through Have I been Pwned.



```
jerry@79:~$ sloc Bitwarden
----- Result -----
      Physical : 8005
      Source   : 6980
      Comment  : 262
Single-line comment : 168
  Block comment : 94
      Mixed    : 34
Empty block comment : 0
      Empty   : 797
      To Do   : 1
Number of files read : 75
-----

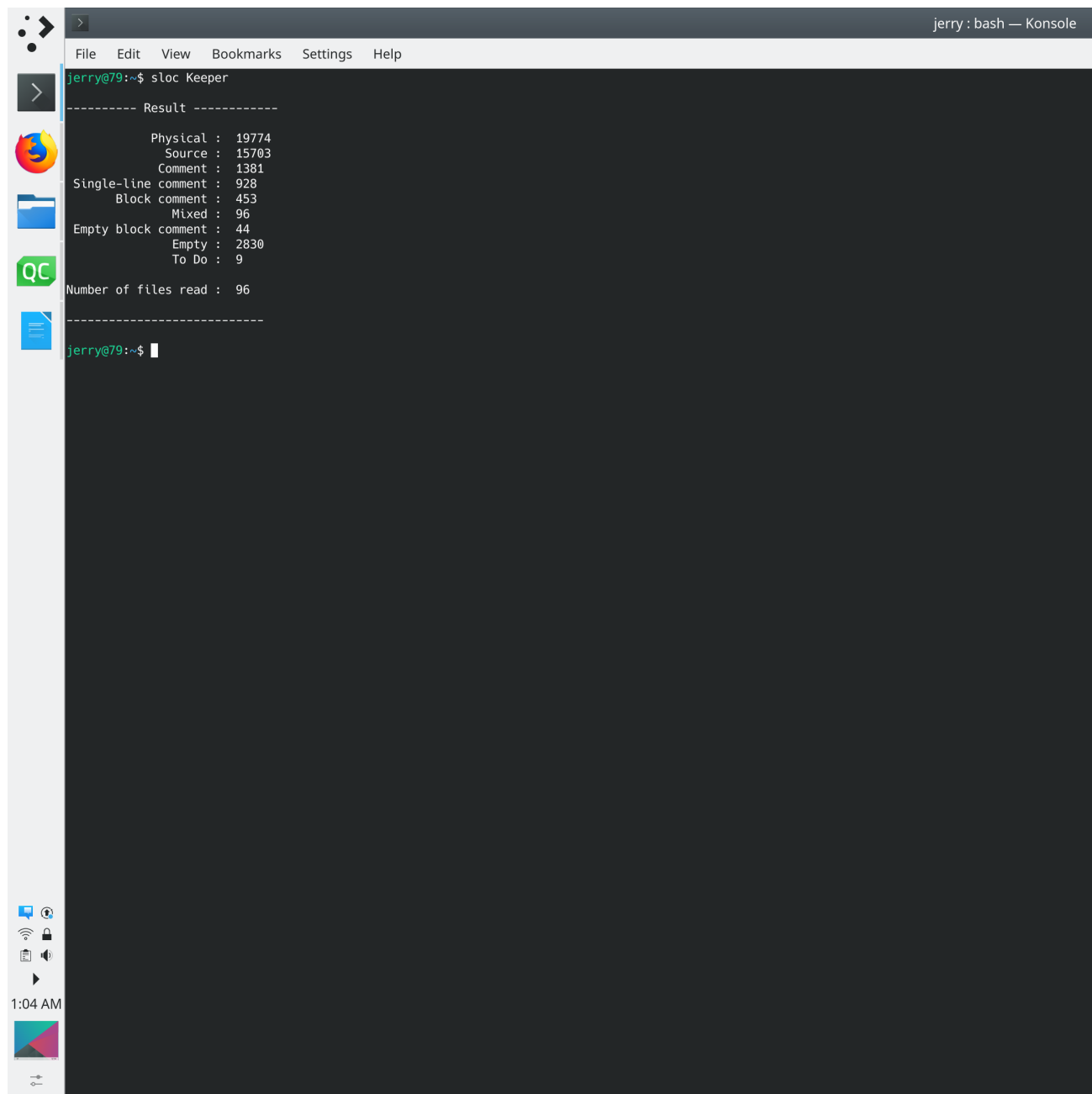
jerry@79:~$
```

Keeper

Keeper is a digital vault created by Keeper Security that stores website

passwords, financial information as well as other sensitive documents using 256-bit AES encryption, zero-knowledge architecture and two-factor authentication. Keeper's guided multi-step setup gets you up and running quickly. Keeper provides the following features:

- Cloud Synchronization
- Autofill Facility
- Secure Password Sharing
- Secure Chat facility
- Monitoring dark web for stolen passwords and notifying the user within their vault.
- Two-factor Authentication



	KeePassXCIPFS	KeePassXC	LastPass	Bitwarden	Keeper	
Properties						
Database Location	Decentralized	Local	Cloud	Cloud	Local	
Software licensing	Open Source	Open Source	Proprietary	Open Source	Opensource	
Portability	Portable	Portable	Not-portable	Not-portable	Not-portable	
Availability	Offline	Offline	Online	Offline	Offline	
Single Point of Failure	No single point of failure	Single point of failure	Single point of failure	Single point of failure	Single point of failure	
Re-encryption feature	Available	Not-available	Not-available	Not-available	Not-available	
Key Combination	Master Password and Key file	Master Password and Key file	Master Password	Master Password	Master Password	
Secure Credential Exchange	Secure	Risky	Risky	Risky	Risky	
Subscription	Free	Free	Freemium	Freemium	Freemium	
2FA	Present	Present	Present	Absent	Present	

8 Conclusion

This final chapter brings the capstone project to an end by returning to the agenda which was established in the beginning chapter. In the conclusion chapter I will discuss about some thoughts that can be implemented to further improve the project as well as certain issues faced during the capstone project. At the start of the project the main aim of the project was to develop a decentralized password manager, which included modification of the KeePassXC password manager to share selected passwords as well as integrating IPFS and Nucypher for the purpose of vault duplication as well as secure sharing of credentials. The project began with introduction to the password manager KeePassXC. It was followed by chapter 2 describing Nucypher and IPFS. Chapters 3, 4, 5, 6 described the modified password manager KeePassX-

CIPFS. They included its description, design, implementation as well as well as evaluation. Chapter 7 compares a few password managers and their features among which the decentralization and re-encryption property is just provided by KeePassXCIPFS.

The main aim of the project was to replicate the vaults as as well as sharing of selected passwords. A new feature of sharing selected passwords of a user was introduced known as group password. IPFS and NuCypher modules were combined to function more efficiently. The decentralized password manager with its improved features can prove to be very useful. Multiple users can be be organized into single database, where each user is provided a unique password to unlock the database. The database can be really useful as in an organization, if different users want to share their passwords they can simply transfer the database file with the help of nucypher and credentials through a secure channel. A future improvement which in the project can be introducing a feature in which the password is encrypted with senders private key and decrypted with the help of receiver's private key. But for that we need to eliminate the use of master password. Although Nucypher is trustworthy but its new as it has just been around a year and a half since release of its white paper and therefore it is still under further testing and development. During the testing phase IPFS was not able to work on NuCypher test network as virtual test currency was not available from the link provided in NuCypher's documentation. Therefore, it was just able to be tested on localhost. Second improvement to the project can be adding a user-interface for data storage on ipfs as well as credential transfer using NuCypher. They do have an existing UI but exist independently. KeePassXCIPFS fulfills the

CIA triad of the security world. NuCypher provides confidentiality as well as Integrity while availability is backed up by decentralization provided by IPFS. Usage of Synchronization tool makes the task a lot more easier by synchronization of edited passwords.

Having a Password manager is a recommended security practice. It helps in ensuring security for all the stored passwords within it. Its said that one must use brain for processing rather than storing and remembering. The newly developed password manager can really be very beneficial keeping in mind its multi level security, vault replication as well as user friendly User interface of the password manager.

9 Bibliography