KeePassXCIPFS

Jayal shah

August 2019

Abstract

In the era of censorship, decentralization is forming the basis a large number of industries transforming and revolutionizing the world of application development. One of the best known example of decentralization is BitTorrent used for sharing of files. Peer-to-Peer 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. The problem of single-point of failure in the client-server architecture causes delay and sometimes total break down of the system blocking hundreds of clients from accessing and working with data on their applications. Decentralization provides favorable conditions for businesses as well as application development. The purpose of the study is to focus on some of the decentralized applications along with emphasis on already existing KeepassXC password manager which currently lacks decentralization. The password manager is decentralized using IPFS which acts as storage medium along with providing synchronization among the nodes as well as NuCypher, which is used for Proxy re-encryption helping in sharing of selected passwords.

Chapters

1 Introduction

With escalation of technology and expansion of Web application,Web development is becoming more and more popular. 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 client-server model. The basic use of decentralization is to eliminate the central authority, distributing it equally among the 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.

The whole idea of Decentralization is based on Peer-To-Peer protocol. Decentralized web applications can hop the hurdles of the client-server such as getting rid of the 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 effective as well as robust. Building Decentralized applications wraps all the perks of decentralization together accompanied by scalability and profitability. 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 flow.
- 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 ccurrency 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 ex-filtration than alternative ways of storing passwords[2]. In our study we will be using Keepass password manager which is a well known open source password manager. KeePass is a free open source password manager, which helps you to manage your passwords in a secure way. You can put all your passwords in one database, which is locked with one master key or a key file. So you only have to remember one single master password or select the key file to unlock the whole database. The databases are encrypted using the best and most secure encryption algorithms currently known AES and Twofish[3]. We will be using KeepassXC version of the keePass password manager which is an open source community fork of KeePass. KeePassXC uses a database format that is compatible with KeePass Password Safe[4].

Current version of the KeepassXC password manager is yet to be centralized. In our study we will be decentralizing the password manager using technologies such as IPFS(InterPlanetary File System) and NuCypher, 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 file system. IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing. IPFS is a p2p distributed file system that seeks to connect all computing devices with the same system of

files[5]. IPFS also provides the function of version control by taking files and managing them. It also stores them somewhere and then tracks versions over time along with accounting for how those files 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[6]. 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 as well as will also play the role for synchronization of the the nodes.

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[7] . NuCypher makes use of proxy re-encryption using which a proxy entity can change the encrypted data from one public key to another without getting access to the contents or plaintext 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 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.

The overall aim of the study is to develop a decentralized version of the KeePassXC password manager using decentralized technologies such as IPFS used for storage and synchronization, combined with the service of proxy re-encryption provided by NuCypher. We will be modifying the password manager and adding a new feature of selected password sharing by the user. The technologies used and their working is given in more depth along with diagrams in the following sections. First chapter gave a glimpse of the term paper. Chapter 2 and chapter 3 explains IPFS and NuCypher respectively. Chapter 4 introduces with KeePass password manager and its different versions including basic details about the password manager. Chapter 5 briefs about some of the important concepts useful for development of the Decentralized password manager namely Decentralized and replicated vaults,shared passwords as well as the idea behind multiple master keys. Chapter 6,7,8 are completely based on the decentralized password manager which will be elaborated in depth in the capstone project report.

2 IPFS

2.1 Introduction to IPFS

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[8]. 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[9]. 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.

2.2 Technologies involved

Distributed hash tables play the role of distributed databases that are used to store and retrieve information based on key:value pair[10]. 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[11]. In other words we can say that IPFS uses DTH 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 NodeIds 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[12]. 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. These strategies are not necessarily fixed, and can be designed to do things like incentivise data duplication, or uptime, or punish leechers[13]. 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 versoning system basically comprises of 4 components blob,list,tree,commit[12]. Blobs are free from any links consisting of data generally representing files. Lists also define files but a list consists of multiple blobs. 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. The Diagram given below explains the layered structure of IPFS.

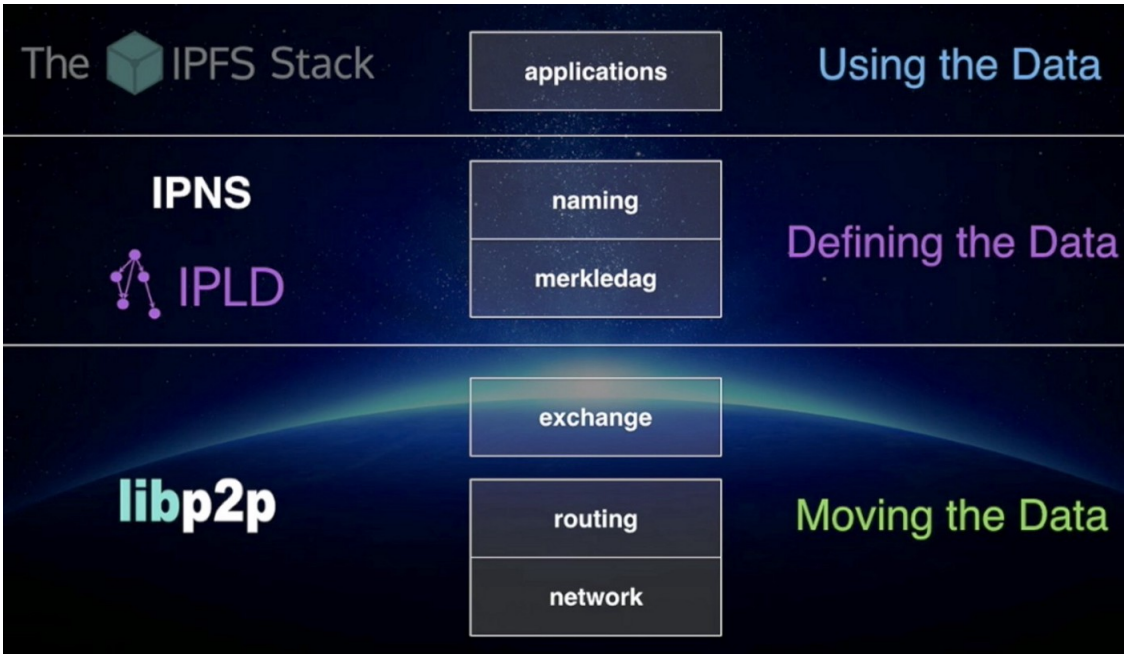

Fig 1:IPFS Layered Structure[14]

2.3 IPFS protocols

IPFS can form the basis for developing and deploying distributed applications along with versioning of large data. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects representing files and other data structures. The IPFS Protocol is divided into a stack of sub-protocols integrated together being responsible for different functionality as follows:

- Identities – Identities manage and perform node identity generation and verification.Each and every node is identified by a unique NodeID represented by a hash of the public key. At the time of connection, exchange of public key takes place between peers and verification is carried out whether the hash of public matches the NodeID or not. In case it fails to match the connection is terminated.

- Network – With the help of various network protocols controls connection with peers. IPFS can use any of the available transport protocol providing reliability using SCTP. The network also maintains integrity using hash checksum and authenticity using HMAC with sender's public key.

- Routing – Used for locating specific peers and objects. IPFS achieves this with the help of distributed hash tables.

- Exchange – As mentioned earlier IPFS uses Bitswap for exchange of blocks.

- Objects – A Merkle DAG links content-addressable immutable objects. It is used to represent arbitrary datastructures, e.g. file hierarchies and communication systems. Merkle DAG grant IPFS with certain properties such as Content-addressing by uniquely identifying content uisng multihash checksum, Tamper resistance as well as deduplication or removal of redundancy.

- Files – IPFS defines a set of objects for modeling a versioned filesystem on top of Merkle DAG.

- Naming – IPFS provides with Self-Certifying mutable name system known as IPNS which helps in assigning human-friendly names.

3 NuCypher

3.1 Introduction to NuCypher

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[15]. 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.

3.2 Proxy re-encryption

Proxy re-encryption lies in the heart of the NuCypher technology. Proxy re-encryption is a type of public-key encryption that 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. Proxy re-encryption goes hand-in-hand with decentralized applications as the token can be created and applied irrespective of time.

Let's use the same actors from the previous example to better understand proxy re-encryption. We all know the basic Public-Key encryption where the data encrypted by Alice's public key can be decrypted only by Alice using her private key. But in case of re-encryption, suppose a 3rd party entity encrypts a message m , with Alice's public key(PkA), resulting in ciphertext (Ca). Alice is now willing to provide access to message m to Bob , having the key pair as (PkB,SkB), for which Alice creates a re-encryption key as given below:

$$RkAb = Rekey(SkA, PkB); \tag{1}$$

Thus after generation the re-encryption key can be used along with generated Ciphertext to transform it into a version of cipher text(Cb) which can be decrypted by Bob using his own secret key(SkB).In the above equation private key of the sender is used but we don't want the receiver to have the senders private key, therefore to overcome this problem a transitory key pair Ske/Pke is generated. The node providing the re-encrytion service works as follows:

$$Ske = random(); \tag{2}$$

$$RkAe = rekey(SkA, Ske); \tag{3}$$

$$Sk'e = encrypt(PkB, Ske); \tag{4}$$

$$RkAb = (RkAe, Sk'e); \tag{5}$$

The re-encryption node uses RkAe to re-encrypt any ciphertext cA (whose underlying message is m) so it can be decrypted by Ske. Since the receiver needs Sk'e to decrypt the re-encrypted ciphertext, this is attached to the re-encryption result. Therefore, the re-encryption process is as follows:

$$Ce = reencrypt(RkAe, Ca); \tag{6}$$

$$Cb = (Ce, Sk'e); \tag{7}$$

The decryption by the receiver takes place as follows:

$$Ske = decrypt(SkB, Ske); \tag{8}$$

$$m = decrypt(Ske, Ce); \tag{9}$$

Proxy re-encryption can be further divided into two types namely Interactive and Non-interactive. In the case of Interactive re-encryption the key is computed using two secret keys, while on the other hand the key is computed using owners private key and delegate's public key. The proxy re-encryption algorithm can be further classified on the basis of directionality as well as based on the number of hops. In case of directionality the algorithm can be unidirectional as well as bi-directional. In uni-directional it is not possible to compute RkA B from RkB A ,while in bi-directional it can be calculated. On the basis of number of hops the re-encryption can be either based on Single-hop or Multi-hop. In case of Multi-hop , suppose we have Rk A B and Rk B C , it is possible to compute Rk A C. In case of Single-hop , it is not possible to re-encrypt it further. Several protocols are adopted by NuCypher in order to enhance security as well as prevent possible threats. Some of which are discussed below.

3.3 umbral

NuCypher uses a split-key threshold re-encryption scheme to decentralize the trust among multiple miners on the network known as Umbral. The figure given below shows the working of Umbral threshold and how splitting of keys take place. It provides enhanced trust to the users.

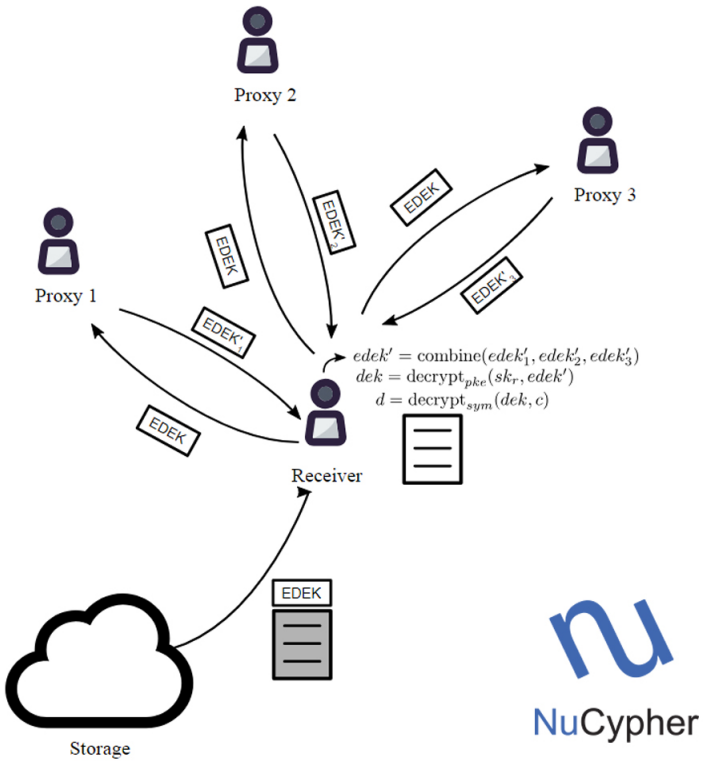

Fig 2:Split Key Re-encryption[17]

To overcome the problem of nodes malfunctioning NuCypher uses a challenging protocol which eliminates the risk of miners providing random data instead of accurate re-encrypted data. Fake re-encryption keys are designed to challenge the miner and if the node malfunctions or the miner cheats, the data along with the key are irrelevant to any private data. If there is no relevancy receivers can prove that the re-encryption result should be different. NuCypher also supports pseudo-anonymity, as the participating nodes are unaware of the re-encryption data as well as they do not store identities of any participant. NuCypher has multiple number of use-cases such as Sharing encrypted files using Decentralized Dropbox, End-to-end encrypted group chat along with electronic patient-control health record and many more. NuCypher can be used to create secure applications in different areas by bringing private data sharing and computation to decentralized ledger. In out project we will use NuCypher for users to share a handful of passwords to other users willlingly.

4 KeePassXC

4.1 KeePass and its community forks

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[18]. KeePass is a small and light project. User simply needs to unpack from the Zip package. It can be transferred also 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 the passwords, usernames, email accounts, URLs 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 Keyfile used 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 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 functionality won't help. To avoid this kind of situations, backups can be done regularly[18].

After KeePass lets 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[19]. On the other Qt is a free and open-source widget toolkit gor creating GUI's as well as cross-platform applications with little or no change in the code[20].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[21]. 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 inbuilt 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++ running natively across all available platforms. Certain features lacked by KeePassX are present in KeePassXC which were included in the new versions of the native KeePass password manager.

4.2 Configuration and installation

KeePassXC in new and improved version including new features and bug fixes making the conditions more user-friendly. Some of the main features of the XC version include are as listed below[23]:

- Secure storage of passwords and other private data with AES(Advanced Encryption Standards), Twofish or ChaCha20 encryption schemes.
- 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 externallly changed.
- Browser integration with KeePassXC-Browser for Google Chrome, Chromium, Vivaldi, as well as Mozilla Firefox.
- In Updated versions of KeePassXC KeePassHTTP has been replaced by KeePassXC-browser as KeePassHTTP has a security flaw in which an attacker can decrypt the passwords, if they manage to intercept the communication between KeePassHTTP server and KeePassHTTP-connector over wired connection[24].

KeePassXC can either be directly installed as a package from 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:

- STEP 1: Downloading or cloning the KeePassXC github repository and extracting if the zip file is downloaded.
- STEP 2: Now to build the source code we will be requiring cmake. If not available, Cmake can be easily installed form the terminal by typing "sudo apt install cmake"
- STEP 3: Next, navigate to the KeePassXC folder followed by navigating to the directory containing CmakeLists.txt and type cmake.

- STEP 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 form where they can be downloaded or commands to install them.

    1. QT5 (“sudo apt install qttools5-dev”)
    2. libgpg(“https://gnupg.org/download/index.htmllibgpg-error”)
    3. gcrypt(“https://gnupg.org/download/index.htmllibgpg-error”)
    4. Argon(“sudo apt install argon2-dev”)
    5. zlib(“sudo apt install zlib1g-dev”)
    6. QREencode(“https://fukuchi.org/works/qrencode/“)
    7. Sodium(“https://download.libsodium.org/libsodium/releases”)
    8. Qt5X11Extras(“sudo apt install libqt5x11extras5-dev”)

- STEP 5: After Installing all the required files KeePassXC is built successfully as shown

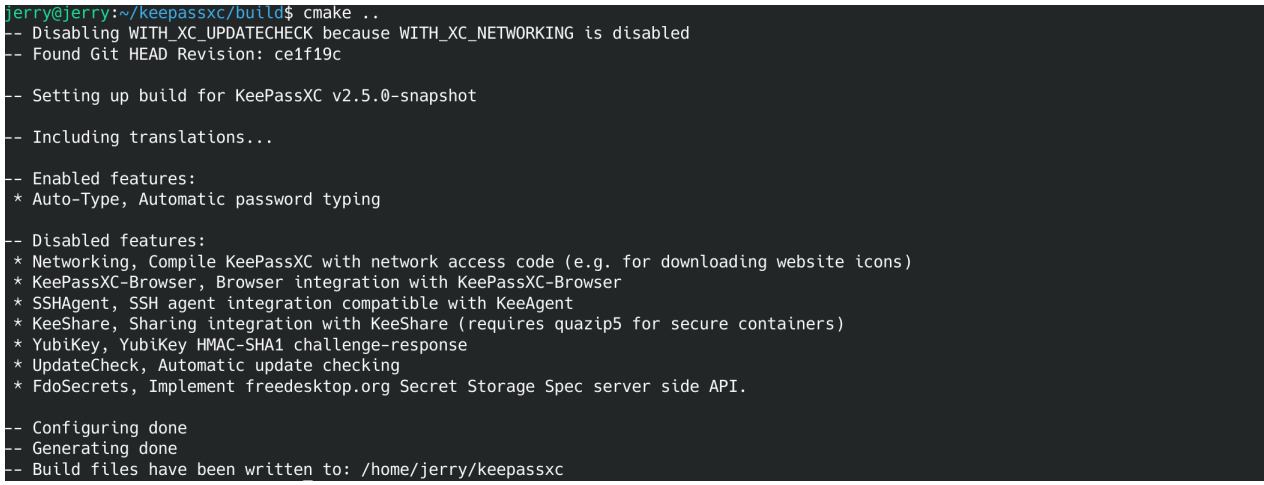

Fig 3: KeePassXC successful built

- STEP 6: KeePassXC can now be accessed by navigating to the directory containing the built source file and running it.

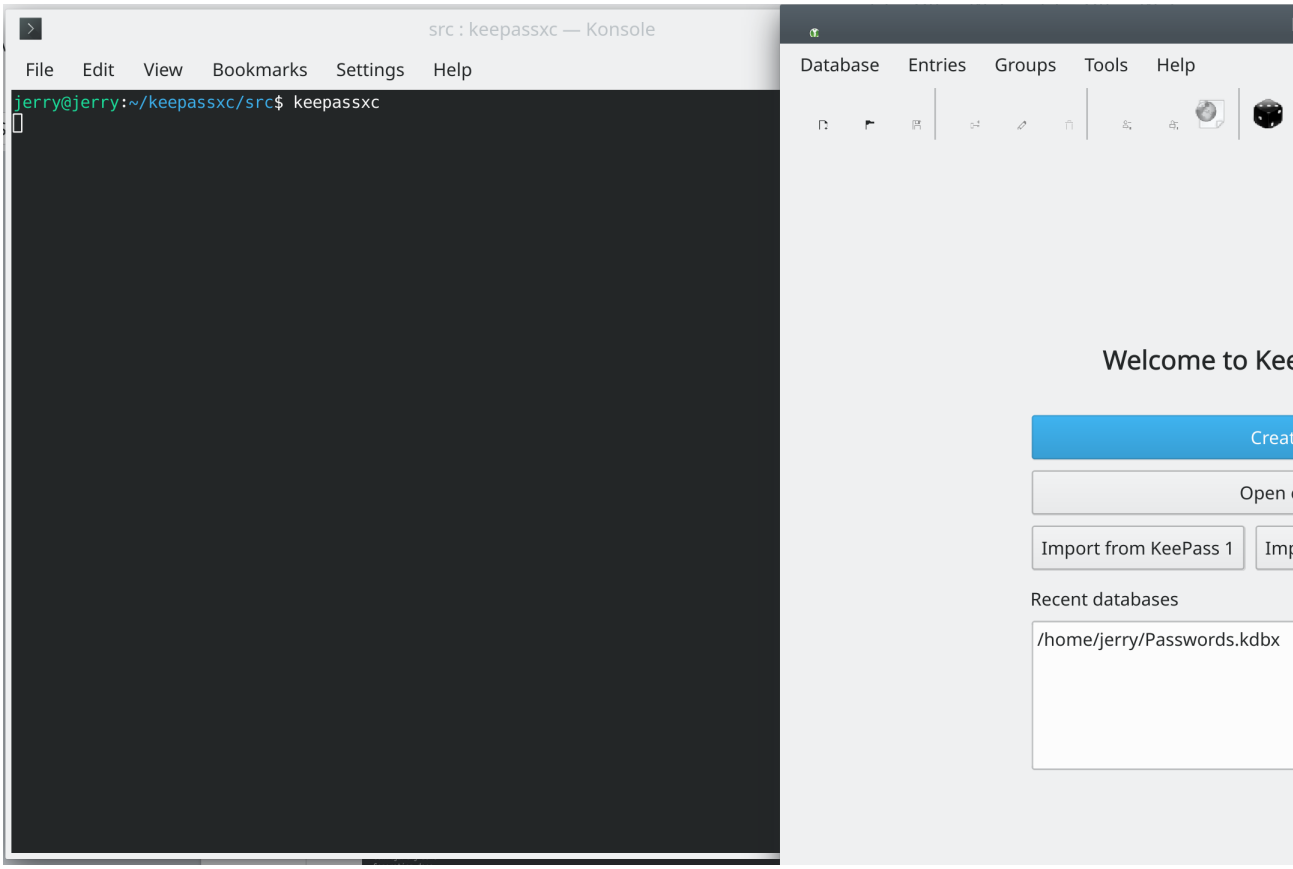

Fig 4: KeePassXC running Successfully

After successful installation of the KeePassXC a slocount was performed on the source code. The sloccount is used as software metric and it's result will be useful in development of decentralized password manager. It automatically estimates the effort, time, and money it would take to develop the software, using the COCOMO model or user-provided parameters. The result is as shown below.

SLOC Directory SLOC-by-Language (Sorted) 26267 $\mathrm{src}_z xcvbnansic = 2626720725 src_k eepassx_c ore_a utogencpp = 2072518753 testscpp = 17864, xml = 635, javascript = 25416803 src_g uicpp = 16780, xml = 239168 src_c orecpp = 91685538 src_f ormatcpp = 55383849 src_a utotypecpp = 3793, python = 563416 src_b rowsercpp = 34163402 src_f dosecretscpp = 34023136 src_c ryptocpp = 2502, ansic = 6342750 src_k eesharecpp = 27501713 src_c licpp = 17131354 CMakeFilescpp = 852, ansic = 5021154 src_s treamscpp =$

$11541125 src_autotype_autogencpp = 1125947 top_dirsh = 947832 htmljavascript = 832810 src_keyscpp = 810678 src_shagentcpp = 678598 sharexml = 538, sh = 60542 src_qrcodecpp = 542394 src_zxcvbn_autogencpp = 394291 src_totpcpp = 291179 src_proxycpp = 179147 utilssh = 147142 src_updatecheckcpp = 142135 src_top_dircpp = 13529 src_touchidcpp = 297 snapsh = 70 build(none)0 cmake(none)0 docs(none)0 latex(none)0 src_CM$

Totals grouped by language (dominant language first): cpp: 93982 (75.26ansic: 27403 (21.94xml: 1196 (0.96sh: 1161 (0.93javascript: 1086 (0.87python: 56 (0.04

Total Physical Source Lines of Code (SLOC) = 124,884 Development Effort Estimate, Person-Years (Person-Months) = 31.80 (381.54) (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05)) Schedule Estimate, Years (Months) = 1.99 (23.93) (Basic COCOMO model, Months = 2.5 * (person-months**0.38)) Estimated Average Number of Developers (Effort/Schedule) = 15.94 Total Estimated Cost to Develop = $4,295,105$ ($average salary$ =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 GPL. SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to redistribute it under certain conditions as specified by the GNU GPL license; see the documentation for details. Please credit this data as "generated using David A. Wheeler's 'SLOCCount'." **sloccount keepassxc Have a non-directory at the top, so creating directory top**$_dirAdding/home/jerry/keepassxc/CHANGELOGtop_dirAdding/home/jerry/keepassxc/C$
$1.0 top_dirAdding/home/jerry/keepassxc/LICENSE.BSDtop_dirAdding/home/jerry/keepas$
$2 top_dirAdding/home/jerry/keepassxc/LICENSE.GPL-3 top_dirAdding/home/jerry/keepa$
$2.1 top_dirAdding/home/jerry/keepassxc/LICENSE.LGPL-3 top_dirAdding/home/jerry/ke$
$LGPL-EXCEPTION top_dirAdding/home/jerry/keepassxc/Makefile top_dirAdding/home/$
$tool top_dirCreating file list for share Creating file list for snap Adding/home/jerry/keepassxc/so$
$project.properties top_dirCreating file list for tests Creating file list for utils Creating file list for s$
$directory at the top, so creating directory src_top_dirAdding/home/jerry/keepassxc/src/CMakeLists$
$keepassx.h to src_top_dirAdding/home/jerry/keepassxc/src/config-keepassx.h.cmake to src_top_dirC$
$info.h to src_top_dirAdding/home/jerry/keepassxc/src/git-info.h.cmake to src_top_dirCreating file l$
$terminated in string in /home/jerry/keepassxc/src/gui/AboutDialog.cpp$

SLOC Directory SLOC-by-Language (Sorted) 26267 src$_zxcvbn ansic =$
$2626720725 src_keepassxc_core_autogencpp = 2072518753 testscpp = 17864, xml =$
$635, javascript = 25416803 src_guicpp = 16780, xml = 239168 src_corecpp =$
$91685538 src_formatcpp = 55383849 src_autotypecpp = 3793, python = 563416 src_browsercpp =$
$34163402 src_fdosecretscpp = 34023136 src_cryptocpp = 2502, ansic = 6342750 src_keesharecpp =$
$27501713 src_clicpp = 17131354 CMakeFilescpp = 852, ansic = 5021154 src_streamscpp =$

$11541125 src_autotype_autogencpp = 1125947 top_dirsh = 947832 htmljavascript = 832810 src_keyscpp = 810678 src_sshagentcpp = 678598 sharexml = 538, sh = 60542 src_qrcodecpp = 542394 src_zxcvbn_autogencpp = 394291 src_totpcpp = 291179 src_proxycpp = 179147 utilssh = 147142 src_updatecheckcpp = 142135 src_top_dircpp = 13529 src_touchidcpp = 297 snapsh = 70 build(none)0 cmake(none)0 docs(none)0 latex(none)0 src_CM$

Totals grouped by language (dominant language first): cpp: 93982 (75.26ansic: 27403 (21.94xml: 1196 (0.96sh: 1161 (0.93javascript: 1086 (0.87python: 56 (0.04

Total Physical Source Lines of Code (SLOC) = 124,884 Development Effort Estimate, Person-Years (Person-Months) = 31.80 (381.54) (Basic COCOMO model, Person-Months = 2.4 * (KSLOC1.05)) Schedule Estimate, Years (Months) = 1.99 (23.93) (Basic COCOMO model, Months = 2.5 * (person-months**0.38)) Estimated Average Number of Developers (Effort/Schedule) = 15.94 Total Estimated Cost to Develop =** $4,295,105 (average salary =$ **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 GPL. SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to redistribute it under certain conditions as specified by the GNU GPL license; see the documentation for details. Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."**

4.3 Comparison of Password Managers

Multiple Password managers are available in the market these days varying from each other based on price, features,compatibility, security as well as based on their user-friendliness. We even have browser-based password managers but they are less recommended as browser's have multiple other tasks and responsibilities which eliminates password management from their priority list and therefore using a password manager which has a dedicated goal of password management leading to better security should be used. Most password managers are systems rather than a single thing consisting of apps or browser extensions for each of your devices, having tools to help you create secure passwords, safely store them, and evaluate the security your existing passwords. All that information is then sent to a central server where your passwords are encrypted,

stored, and shared between devices[24]. They also manage compromised passwords along with searching through passwords database to ensure compromised codes are not reused. After going through basics of KeepassXC and its working, Now let's have a look at some of the other popular password managers and compare them with KeePassXC. Popular password managers include LastPass, 1Password and DashLane. Different aspects of the password managers have been compared in the comparison table below[26].

	KeePassXC	LastPass	1Passwo
Properties			
Availability	Offline	Online	Offline
Two-Factor Authentication	Present	Present	Not-Pres
Browser Integration	Possible	Possible	Possibl
Portability	Portable	Not-Portable	Not-Porta
Mobile Application	Available	Available	Availabl
Adding Plugins	Support not available	Support available	Support ava
Pricing Model	Opensource	Free/Paid	Paid

5 KeePassXC on IPFS + NuCypher

5.1 Combined Technologies

In the earlier sections we saw how Nucypher can be used in vast variety of applications that are needed to share private and encrypted data. We even saw how IPFS can behave as a storage medium in the decentralized world. On combining both of them files can be encrypted on client-side and stored in decentralized medium. After which the files can be easily shared with trusted third-parties, where a re-encryption token is provided on the basis of 3rd party's public key. The third-party's permission access can be easily removed by removing the re-encryption token from the network. We can understand the combined working of IPFS and NuCypher with an Example. Suppose I want to upload a 10 TB file and allow access to 5 of my friends. Traditionally I would ask all my 5 friends for their public keys ,encrypt the data for all of them with their respective public keys and store the encrypted

data on IPFS for them to retrieve. However NuCyphers proxy re-encryption allows me to encrypt the data once, then re-encrypt new keys that will allow all the 5 parties access the data. In this system I only encrypted the 10 TB once along with storing it just once on IPFS, this saves storage space, encryption time, and encryption computations along with making it more scalable. The combination would really be helpful when the scale for number of receivers is large a scenario where the number of receivers is unknown. It would combine all the benefits of NuCypher and IPFS making it more secure. Another example use case can be Medical Records. Medical records require private use of data.Several blockchain based projects are aiming to replace centralized medical records providers, like Epic, with a decentralized service giving the patient ownership and control over their own medical data[27]. For such service to be feasible the data needs to remains properly encrypted at all times but still shareable with multiple parties, including medical providers, applications, insurance companies, medical researchers, or even other patients. MediBloc is another example of a patient-controlled electronic health record system that will ne using NuCypher to encrypt medical records on IPFS [27].With NuCypher, patients using MediBloc could encrypt and store each record only once with their own encryption keys and then upload their encrypted records to IPFS. When the patient wants to grant access to a service provider, the patient can create a new re-encryption key using the service's public key and issue it to the NuCypher network. The network will use the re-encryption key to transform the encryption on the medical records so the service provider can decrypt them using their own private key. The patient has the ability to revoke access to that service provider at any time by issuing a revocation request to the NuCypher network. At that point, the service provider would no longer be able to decrypt the encrypted records, although they would continue to have access to any previously saved decrypted records.Regardless of the size of the underlying encrypted data, the cost to re-encrypt files with NuCypher remains constant. In the figure given below another example is depicted which shows how Alice uses NuCypher to create a policy public key for the Heart Monitor to use, so she can read and delegate access to the encrypted data as she sees fit. The Heart

Monitor uses this public key to produce a file with some amount of encrypted heart rate measurements which is then uploaded to decentralized storage service. Alice can now share the data with other people by obtaining their public key, followed by creating a policy. After this, other people can read the file with encrypted dataand request a re-encrypted ciphertext for each measurement, which can be opened with their respective private key.

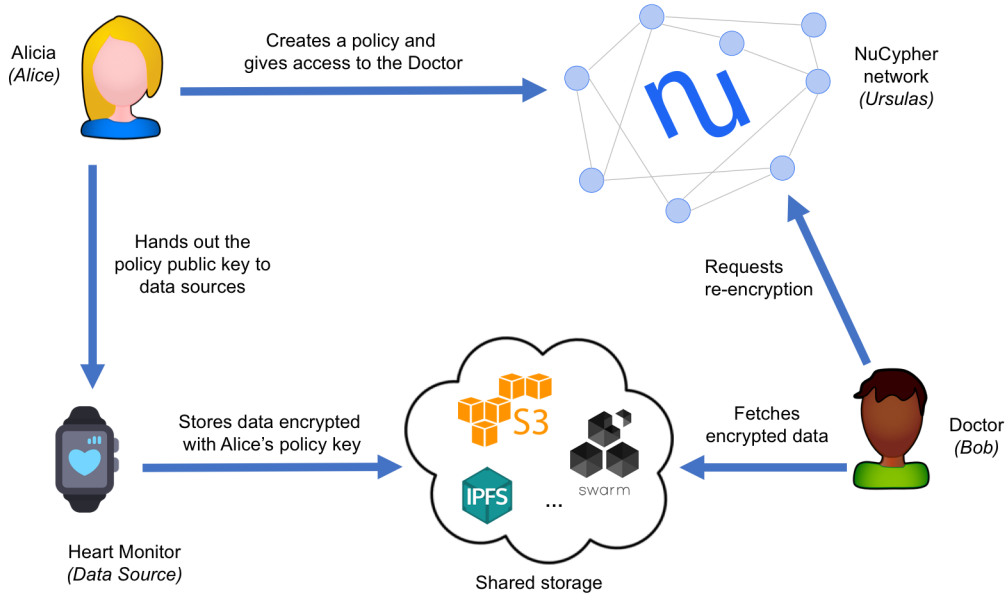

Fig 5: NuCypher + IPFS[28]

5.2 Decentralized and Replicated Vaults

A vault in physical world is used to store priceless or critical items necessary for the working of an organization. Losing or exposure of these vital items can possibly disrupt the enterprise's business. We will be talking in context of digital vaults since majority of the information in today's world is stored digitally. A digital vault is used to store valuable digital information. A digital vault provides certain standard services to ensure that its content is protected, acting as a long-term repository, highly secured regardless of overall network security and regardless of the physical topology of the network. It offers an effective way to protect and control critical

information allowing an organization to focus its defense resources to a vault at any location. The Vault protects the information residing inside the vault from all major security risks over a network environment[29]. A digital vault also create and manages strong passwords along with sharing confidential information between multiple users or among the teams. In our case we will be using KeePassXC password manager as our digital vault. The Password manager consists of all almost all the features digital vault consists of providing equal security.

In the last section we gained a little idea about digital wallets and their functioning. Now lets learn about decentralized version of the digital vault. Te best way for storing passwords on IPFS is storing them in at a single place, for which nothing can be better than a password manager. The password vault or the password manager database can than be moved onto IPFS. We know that for every change in content of the file, the hash is updated. For every change or update in the password manager the hash will be changed,which in turn will change the address. Interplanetary Naming system will be helpful in this case which can be used to point to the latest version of the file. IPNS is a simple service that uses your peer ID to point to a particular hash. This hash can change, but your peer ID doesn't. That means that you can point to content in IPFS that may also change, and people can still access it without needing to know the new hash before hand.It updates the pointer to the content .Updating IPNS requires access to a private key; which can also be stored in the Password vault, thus assuring one can always update the IPNS record if they have access to the Password Safe and the master password.IPFS seems to mostly be about making data available widely and permanently. Data in IPFS is protected by the obscurity of the link and any encryption you provide. Although IPFS provides functionality for pointing towards new versions, at the same time IPFS lacks Dropbox-like cloud functionality with actual synchronization. It is only possible to mirror a certain state of a directory at any given time, from one node to another. To synchronize a new version of the directory, you would first have to delete the old directory its contents from the receiving IPFS node, then pin the new directory from the sending node on the receiving node. IPNS would not really be

of any help overhere, as the only difference is that you now have a shared secret IPNS publishing key and an IPNS hash known to both nodes, but you'll still not be able to actually synchronize the shared directory's contents; you still need to perform a handful of tasks such as deleting the old files from one node followed by getting the new/edited files from the other node using the IPNS hash. One way we can synchronize the files is by clustering the nodes who want to synchronize in a private swarm, which would be dedicated swarm for just the vault directory, then use ipfs-cluster, meaning that all changes to one node will be mirrored on the other nodes automatically. Another option is to use BitTorrent sync also known as BTsync which lets sync file across devices or share them with friends using peer-to-peer file sharing technology. Files are also encrypted in transit, and since there's no real cloud service there are no storage limits beyond the hard drive size of your devices[30].It can also be used for tracking down document versioning as well sharing documents with other users in read-only mode. Although BitTorrent sync has certain downsides such as users cannott sync specific files with specific devices. With BTSync, user can either sync a folder among devices or not. Devices also need to be powered on and signed in for BTSync to work; the BitTorrent protocol relies on both devices being active to share files. Even with the mentioned drawbacks BitTorrent Sync is a terrific option for synchronizing data without relying on cloud services and third-party servers.Both the method can be used as a work-around for one-way node-synchronization but we still need something solid for two-way synchronization. For the purpose of two-way synchronization or replication we will use a tool called brig. Brig is a distributed secure file synchronization tool with version control. It is based on IPFS, written in Go.It can be thought of as a swiss army knife for file synchronization or as a peer to peer alternative to Dropbox[31].We will be using brig for two-way synchronization of the vaults on the ipfs nodes. Brig provides the following features as mentioned below[32]:

- The main feature, which is most useful to us is that it can handle two-way synchronization of moved files as well as empty directories.

- **Encryption of entire data during storage and transport using AES-256 in GCM mode.**
- **Brig provides the feature of strong and easy version control.**
- **It provides auto-updating facility which synchronizes any kind of alteration.**
- **Simple user identification and discovery.**
- **FUSE filesystem that makes all files seem like a normal directory.**
- **Gateway to share normal HTTP/S links with other users.**

5.3 Password Sharing Across Trusted Users

Although sharing passwords is not a good security practice but in certain situations we need to share the passwords with other users. In the current Version of the KeePass Password manager the only way for sharing new passwords is creating a whole new database. We went through the basics as well as working of NuCypher technology. The main idea behind using NuCypher was to add a feature to the password manager in the capstone project. The new feature added will help the user to share selected passwords with other users. Password managers such as Lastpass and 1Password have the facility to share passwords. In both of them instead of creating a new database. The user can create a new group and share the group with the user he/she is willing to share the password. But in my scenario of the capstone project, using NuCypher I'm trying to develop a way in which the user will be able to share selected passwords from the same database and from the same group. We can use the concept of proxy re-encryption in order design a new feature and add it to working of KeePassXC. The process for doing so is still in process and the advances will be reflected in the capstone project report.

5.4 Multiple "Master Keys"

A password manager or a vault with multiple master keys would really be helpful in certain scenario's. For example, In a corporate environment 2nd master key would really be helpful for the sysadmin to open an employee's password manager or the vault with a password or a key file when the employee has stored important corporate information and has been terminated or in not available for a certain period of time. Certain password managers other than KeePassXC have facility where it is possible to encrypt the Master Key using multiple different passphrases, each different than the actual database Master key using plugins. In the case of multiple master keys, when we have a look at it from decentralization point of view it is again somewhat similar to the previous section of sharing passwords across trusted users. Suppose a user wants to share the vault with a single user he can easily the key by public-key encryption I.e encrypting the key with other user's public key and then transferring the key, where the other user can simply decrypt it with his/her own private key and get the master password to the vault. But what in case of hundred's of users, it wont be feasible to encrypt the key and send it to all the users. In such cases proxy re-encryption can be used which will help in generating unique master keys for all the users willing to get the master key from the original user. In another situation, it is also possible that user does want to share the vault with other user but is unwilling to share the master key as he uses the same master key for another vault to. In this case proxy re-encryption can really be useful as the nodes present on the NuCypher network can perform re-encryption which the receiver can decrypt using their own private key.

6 Design of KeePassXCIPFS

KeePassXCIPFS will be a combined module consisting of multiple binded together. IPFS will form the basic layer on top of which will be NuCypher technology. KeePassXC password manager will be on top of the combination at the application level. The pass-

word manager will be useful in sharing as well as accessing vault irrespective of the location. Majority of the benefits of the decentralization technology are available to KeePassXCIPFS. NuCypher Technology with its re-encryption capability will be useful in generation of multiple keys as well as will form the basis for addition of new feature to the password manager. A basic diagram is as shown below. The design of the password manager is still in progress and more detailed structural diagram as well as detailed design layer descriptions will be made available in the capstone project report. From the Independent study point of view all the technologies were implemented individually and were working successfully.

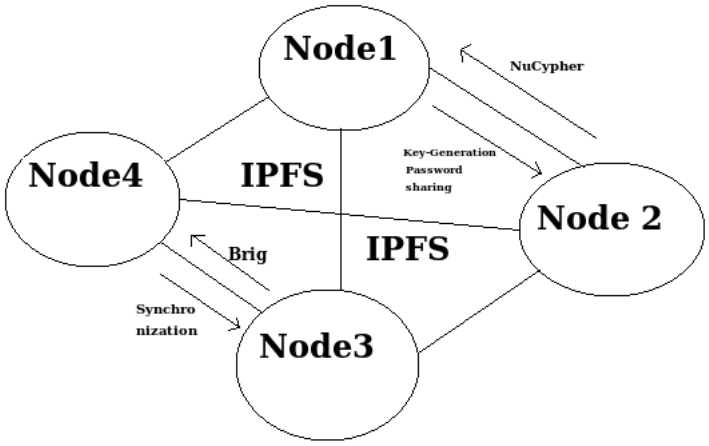

Fig 6: KeePassXCIPFS Basic Working

7 Implementation of KeePassXCIPFS

The term paper just includes a basic idea about the architecture and design of KeePassXC. The design of the project is still in progress and therefore the implementation is also in progress. All the technologies used are open source and therefore the source code is easily available for all the technologies used. The source code for KeePassXC is available in c++ while NuCypher is in python. IPFS is written in Go but it has libraries written in python which will be useful in our implementation. The challenge lies in combining

all the codes together in order to develop a decentralized password manager. The code level descriptions , pre and post assertions as well as doxygen expectations will be available in the project paper.

8 Evaluation of KeePassXCIPFS

The Decentralized version of the KeePassXC password manager, which will be developed as KeePassXCIPFS is supposed to have enhanced efficiency as compared to KeePassXC as well as other centralized or stand-alone password managers. The key plus point for KeePassXCIPFS is added advantage of decentralization. For now we will focus on evaluating the password manager on basis of security and performance.

8.1 Security Evaluation

Security Evaluation The IPFS version of KeePassXC would be free from all threats that the client-server model possess. IPFS encryption adds an extra layer of security to our password management. At the same time KeePassXC uses secure encryption algorithm. Combination of NuCypher protects the privacy of the users as they don not need to share any kind of keys with other users.

8.2 Performance

Performance: Speed and Memory The Newly developed password manager will be much more efficient as compareed to previous one as it has the properties of decentralization which increases the bandwidth for the use, reducing the time for the process and providing faster speed. Along with proxy re-encryption, the user can save a lot of time in case of multiple users by avoiding the traditional public-key encryption. The memory in case of decentralization depends on the available memory of the node user.

8.3 SLOC

3. SLOC Comments on Source lines of code is possible only after its successful implementation and will be elaborated in the documentation of the capstone project based on implementation of KeePassXCIPFS.

9 Conclusion

Having a Password manager is a recommended security practice. It helps in ensuring security for all the stored passwords within it. Considering the benefits of decentralization various decentralized technologies useful in development of a decentralized password manager were discussed. IPFS was selected as a storage medium combining it with NuCypher's proxy re-encryption technique helpful in Sharing of keys without losing privacy over the decentralized network. For the porpose of synchronization of the nodes on the ipfs network a tool named brig can be used. Although the term paper lacks implementation of the combined technologies to develop KeePassXCIPFS password manager, which will be continued in the capstone project. All the combining technologies when tested individually were working fine but the real challenge lies in combining them to develop a working decentralized password manager. After the development of the decentralized password manager certain aspects which were not elaborated in the term paper can be explained in more depth. Architectural diagrams, code level descriptions along with assertions as well as evaluation based on multiple factors will be carried out in the project paper after successfully implementing the decentralized password manager.

10 Bibliography/ References

1. Paralkar Keyur. Apr-2018."Decentralized Web Application Using Ethereum Blockchain." International Research Journal of Engineering and Technology,5(4),489-492.

2. "Password Mangers:Under the hoods of Secrets Management". February 19, 2019 Online. Independent Security Evaluators. July14 , 2019. Available:https://www.securityevaluators.com/casestudies/passwo manager-hacking/.

3. Reichl Dominik. "KeePass Password Safe" May,2019. Available:https://keepass.info/.

4. Reichl Dominik. "KeePass Password Safe" May,2019. Available:https://keepass.org/.

5. Wikipedia contributors. (2019, July 16). InterPlanetary File System. In Wikipedia, The Free Encyclopedia. June, 2019, from https://en.wikipedia.org.

6. Mort R. June-2012. "Content Based Addressing: The case for multiple Internetservice providers"[PDF file]. Available from https://pdfs.semanticscholar.org/6bc5/c4d755f86a278fd42b4c04f58ea857f48b2f.p

7. Egorov M. June-2018."NuCypher: A proxy re-encryption network to empower privacy in decentralized systems"[PDF file]. Available from https://www.nucypher.com/static/whitepapers/english.pdf

8. Pors Mark. Aug-2017."The IPFS White Paper: IPFS Design." "Blochchain Train journal,part-5.

9. "ipfs is the distributed Web".2014.Online.Github. June, 2019. Available https://github.com/ipfs/ipfs.

10. Benet Juan. "IPFS - Content Addressed, Versioned, P2P File System"[PDF file]. Available from https://github.com/ipfs/papers/raw/master/i cap2pfs/ipfs-p2p-file-system.pdf

11. Dufel Michael.Dec-2017. "Distributed Hash Tables And Why They Are Better Than Blockchain For Exchanging Health

Records" June, 2019. Available at https://medium.com/@michael.dufel 10220/distributed-hash-tables-and-why-they-are-better-than-blockchain-for-exchanging-health-records-d469534cc2a5.

12. Khan Farhan. Sep-2018. "Chord: Building a DHT (Distributed Hash Table) in Golang" June, 2019. Available at https://medium.com/techlog/chord-building-a-dht-distributed-hash-table-in-golang-67c3ce17417b.

13. Farmer Carson. Sep-2018. "Swapping bits and distributing hashes on the decentralized web" Jun, 2019. Available at https://medium.com/textileio/swapping-bits-and-distributing-hashes-on-the-decentralized-web-5da98a3507.

14. Krstonic Nemanja. Jun-2018. "A Closer Look at the Inter-Planetary File System". Jun,2019. Available at https://medium.com/mvp-workshop/a-closer-look-to-the-inter-planetary-file-system-b3f3af31a3c7.

15. "NuCypher".Jun, 2019. Available at https://docs.nucypher.com/en/latest/.

16. "NuCypher : Future of Blockchain". Stake Zero Ventures.July, 2019 Available at https://www.futureofblockchain.co.uk/nucypher

17. "NuCypher ICO review". Nov-2018. PRIMEICO. Jul, 2019. Availble at https://primei.co/nucypher-ico-review/

18. Kouzari Eli.Feb-2008. "Software Requirements Specification for KeePass Password Safe"[PDF file]. Available at https://keepass.info/extensio KeePass-1.10.pdf

19. Wikipedia contributors. (2019, August 2). Mono (software). In Wikipedia, The Free Encyclopedia. Retrieved 19:56, August 5, 2019, from https://en.wikipedia.org/w/index.php?title=Mono(software)o

20. Wikipedia contributors. (2019, July 30). Qt (software). In Wikipedia, The Free Encyclopedia. Retrieved 19:57, August 5, 2019, from https://en.wikipedia.org/w/index.php?title=Qt(software)oldid=90

21. KeePassX community. Jul, 2019. Available at https://www.keepassx.org/ t

22. Tiwari Nitish. 2015. "Secret Stash". Linux Magazine (173/2015).Available at http://www.linux-magazine.com/Issues/2015/173/KeePassX.

23. KeePassXC Community.Jul, 2019. Available at https://keepassxc.org/project/

24. "Keepass2 vs KeepassXC - KeePass-Http connector - Mono in Linux" Decc-2019.Information security.Available at https://security.stackexchang vs-keepassxc-keepass-http-connector-mono-in-linux

25. Gilberson Scott. May-2019." THE 4 BEST PASSWORD MANAGERS TO SECURE YOUR DIGITAL LIFE" Online. Jul, 2019 . Available at https://www.wired.com/story/best-password-managers/

26. Fitzpatrick Jason. May-2018. "Password Managers Compared: LastPass vs KeePass vs Dashlane vs 1Password".Online. Available at https://www.howtogeek.com/240255/password-managers-compared-lastpass-vs-keepass-vs-dashlane-vs-1password/

27. S+C intelligence. Jul, 2019 . Available at https://www.smithandcrown.com/asset

28. "Heartbeatdemo".Jul, 2019 Available at https://github.com/nucypher/nucypher

29. Macleod Calum.Oct-2005."What Are Digital Vaults".Online.Available at https://www.helpnetsecurity.com/2005/10/11/what-are-digital-vaults/

30. Ian Paul. May-2014."Deep inside BitTorrent Sync's cloudless file syncing" Online. Available at https://www.pcworld.com/article/2152444/dee inside-bittorrent-syncs-cloudless-file-syncing.html

31. "brig - decentralized secure synchronization".Online.Aug, 2019.Available at https://brig.readthedocs.io/en/latest/

32. "brig - Features".Online.Aug, 2019.Available at https://brig.readthedocs.io/en/la page