CMP6200 Individual Honours Project – Matthew Potts – Development Logbook

# 27/09/23

**Basic market research:**

Given my project is based on a cryptographic receipt generator, I set about finding similar systems online that operate similarly to this. A predominant market leader I found was AWS Blockchain (Amazon Web Services, n.d.), Amazon produces several cloud-based services that can be implemented in various enterprise environments.

A leading example of this is Amazon Quantum Ledger Database, or QLDB. A ‘fully managed database that provides a centralised, immutable, and cryptographically verifiable transaction log.

# 29/09/23

In terms of QLDB, examples of data that often use blockchain architecture like this include “credit & debit transactions, insurance claim history, supply chain asset tracking, vehicle records, and more’ (Amazon Web Services, n.d.). In another diagram provided, data can be cryptographically choreographed in two interlinked ways, individual files can be stored in a prese nt state and a historical state (e.g. the current state of a bank account, and its history). Similarly, AWS provides a journaling functionality. An immutable sequenced journal storing cryptographically verifiable entries of every change made. Every change is registered as a ‘block’. These changes can be verified with client-side cryptographic software provided by AWS.

Upon further inspection, a file or a folder’s change history is cryptographically hashed (w/ SHA-256) into a ‘digest’. A client-side QLDB API can verify data transactions to ensure integrity (Amazon Web Services, n.d.). AWS claims it’s “serverless”, instead storing QLDB data in a database instead of a traditional blockchain framework, calls can be made with QLDB API in the AWS CLI. It integrates with a tool called CloudWatch, a tool that produces real-time metric logs for all operating software on numerous clouds (AWS, 2018).

# 04/10/23

A similar, open-source tool that operates like AWS QLDB is Trillian (transparency.dev, n.d.), a tool that claims one can “cryptographically prove that data hasn’t been unexpectedly changed” – like AWS, Trillian provides a log that is “immutable”, “accurate” and “verifiable”. Trillian is an open-source software – and can be found on GitHub, under Google’s repository (GitHub, 2023). It can be used to “monitor third-party ecosystems”, with an API that can be operaated on by a “Financial Ledger” application, which generates a “Trillian” app. This can be utilised to ensure “regulatory compliance”, and retain a permanent log of data that can be examined for any malicious foul play. “Certificate Transparency” is a standardised certificate that can be attributed to different ecosystems, that can act as a deterrent to any troublemakers.

# 05/10/23

“Transparency.dev” does have limitations of its own. It claims that “it doesn’t protect against malicious code”, claiming that if a user was to create a new version, and change its ID – it could still be injected into a workspace (transparency.dev, .n.d.). QLDB also has a few limitations of its own – QLDB enforces quotas, for example – only 5 ledgers can be used in one ‘Region’ (enterprise) at any one time, and only 20 tables can be used at any one time. Kinesis Data Streams are also limited to 5 (docs.aws.amazon.com, n.d.).

# 10/10/23

During my initial literature search during my proposal, I discovered an open-source Github repository for transparency.dev (GitHub, n.d.). Upon completion of the proposal, I continued my literature search, and found a few more past papers I used as inspiration for my project. Starting off with Mathieu Quinou’s “Blockchain” (Quiniou Matthieu, Blockchain), and then another publication called “Blockchain in payment card systems” (Godfrey-Welch, Darlene; Lagrois, Remy; Law, Jared; Anderwald, Russell Scott; and Engels, Daniel W, 2018). Two student publications I used were called “Using Blockchain as Secure and Immutable Storage” by Amritpal Singh Sandu (Sandhu, A. “Using Blockchain as Secure and Immutable Storage”, 2018) and “Speechless: A Virtual Personal Assistant “ by Villiam Langsch (Langsch, V. “Speechless a Virtual Personal Assistant.”, 2018).

# 30/10/23:

Having taken a little break from the project, I’m returning to begin a thorough literature search on my project. Having spoken with my supervisor, they suggested trying to narrow down the general scope of my project. At the moment, the general premise of it revolves around a “blockchain-like” framework, to which cryptographic receipts, logs, and malicious checks are operated on a range of specific files, directories; whilst being flexible in how it can be used, either as a local-side, or a server-side application. A predominant issue with this is how broad a project like this would be. If I had a greater temporal scale, manpower – it’s reasonable, but it must be scaled down for the sake of maximising quality over quantity. At first thought, I’d consider creating a server-side only application, or slimming down some of the anti-virus algorithms.

Having also done a basic literature search beforehand, I also came to discover that a few particular words, such as ‘blockchain’ are not documented entirely well regarding cryptographic use – on sites such as BCU Search, Scholar, and IEEE Explore. To help my case, i’m using similar references from projects mentioned within my Proposal Report to jumpstart my literature search.

# 16/11/23;

Upon inspection of ‘Using Blockchain as Secure and Immutable Storage’ (Sandu, A. “Using Blockchain as Secure and Immutable Storage” , 2018) – and its references list, Sandhu’s work has provided me a little bit of inspiration as for the kind of keywords associated with a more intricate search into the subject. A few examples of these keywords include “Privacy”, “Cryptographic Transactions”, “Integrity”, “Accountability”, “Validation”, and “Protocols”.

Likewise, I took a dive into similar papers I recorded for further inspiration, using Godfrey-Welch’s “Blockchain in Payment Card Systems” (Godfrey-Welch, Darlene; Lagrois, Remy; Law, Jared; Anderwald, Russell Scott; and Engels, Daniel W, “Blockchain in Payment Card Systems, 2018); we can also utilise keywords such as “Digital Time Stamping “.

# 22/11/23:

Using ChatGPT, I’ve inputted questions to help generate a few more ideas for my research, by finding new terms I can search pages such as Google Scholar, IEEE Xplore, and BCU Search (OpenAI, 2023). A few terms that it provided included “Decentralised File Receipt Systems”, “Tokenised File Receipts”, “File Integrity Proof with Cryptography”, “File Transaction Records on Blockchains”, “Digital File Receipts with Cryptography”, and finally “Cryptographic File Receipts”.

Garg, Rishabh describes how a blockchain fundamentally operates, claiming that a “transaction is accepted only if the majority of the nodes agree on its validity. When all nodes reach a consensus, transactions are recorded on a new block and added to the existing chain. This work is known as ‘mining’” (Garg, Rishabh, 2023). Similarly, following on from this; a few firms have implemented bespoke proprietary systems of their own accord – that utilise blockchain-like systems to record data modifications, and introduce a new system that “propose a blockchain-based reliable traceability system for telecom big data transactions using smart contracts and the InterPlanetary File System” (Pang, Yue et al, 2022). Although drafted, this doesn’t showcase any practical case studies – this is entirely theoretical. Albeit, Pang et al. did utilise Ethereum as an example blockchain to highlight its efficiency, and its superiority to current implemented solutions.

Furthermore, Pang et al. describes how “Hyperledger Fabric v0.6 were discussed to provide insights for further development” (Pang, Yue et al, 2022). In regard to this, I decided to look further into Hyperledger Fabric. Hyperledger Fabric is described as a ‘platform for distributed ledger solutions underpinned by a modular architecture”, where it provided immense “confidentiality, resiliency, flexibility, and scalability” (wiki.hyperledger.org, n.d.). Hyperledger Fabric is also “open-source”; and operates in tandem with Ethereum, a “peer-to-peer network” that “securely executes and verifies application code, hitherto called smart contracts”, and “allows participants on the blockchain to transact with each another without a trusted central authority” (Garg, Rishabh, 2023).

An important part of this though is understanding fundamentally how a Blockchain operates. Personally, there seems to be many similarities between a Blockchain (i.e. interpretation of Distributed Ledger Technology), and a cryptographic receipt tool. Files may be tokenised, and hashed to construct Blockchain-like links. Since a blockchain is inherently a Distributed Ledger Technology (DLT), I want to look into how they operate, and a few real-world examples of how they’re used.

Although the idea of a distributed ledger technology can be applied to my cryptographic receipt software, given I manipulated it into a server-side based software. However, since for the sake of maintaining a reasonable temporal scale I’m limiting this project to a user-side software/framework; I’m more so intrigued into the operational layout of a DLT, and how it provides a cryptographically verifiable ledger a structure to operate upon. At the very lowest level of a ledger, there are ‘transactions’ – a transaction has a ‘sender’, a ‘signature that authenticates the source, and content (data)’. Transactions are then ‘gathered into blocks, and blocks are chained to each other creating a graph without cycle’ (Qingyi Zhu et al. 2019).

# 23/11/23

Furthermore, upon a little more digging into DLT intricacy, and how they operate, I came across a few more terms – these being a “nonce”, a “hash”, and “mining”. Although other formats as to how a cryptographic ledger could be created, a blockchain-like structure personally portrays itself as more logical, and secure. As I delved further into the field surrounding cryptographic ledging software, and generic file ledging software at the basic level; it became increasingly apparent that these kind of software’s are solely designed for enterprise, and server-side operation.

As a result of this, it’s shifted my thinking as to whether a software like this should perhaps be shifted into a more LAN-side form of application. More specifically, for environments in which closely-controlled, confidential data is operated on. Data, that if mishandled maliciously or without due diligence, can lead to significant ramifications. This puts a spanner in the works as regards to implementation, as Sarmah writes: “No confidential information is transferred in a blockchain network and all the transactions are visible to every node in the network” (Sarmah, S.S., 2018). However, links to a ledger as to whether a transaction has “authenticity” is very important for this. “Blockchain has a property of a database except the fact it stores the information in the header and data is stored in the form of a token or a cryptocurrency”. This just emphasises the kind of structure a receipt must have for it to be within a ledger, data such as a “header”, “hash”, “nonce”, all combine to provide both a unique identifier and a trace of file history. If it’s been modified in secrecy this can easily be revealed.

If I’m going to tone my software towards confidential usage, encryption is vital. Sarmah also describes that as a blockchain doesn’t have a central point of authority, “it can withstand any security attack”, “provide(s) transparency and immutability”, and “sensitive business data can be protected with end to end encryption” (Sarmah, S.S., 2018). This, providing food for thought, has led me into theorising whether a LAN-based application, specialising in end-to-end encryption can provide a blockchain-orientated ledging application specifically designed for small, individual teams operating with sensitive/confidential data. Sarmah continues to reinforce my theoretical basis, claiming that “Blockchain are resilient to cyber-attacks due to peer-to-peer nature and network would operate when some of the nodes are offline or under security attack”.

# 06/12/23

Furthermore, after looking further into the topic and the progress of my project as a whole, it also dawned on me that an ethics form (as required by BCU) was integral. I deduced that only a single document had to be completed, this being the “Student Project Declaration”. I completed this tonight, and I intend to show this to my supervisor when possible.

# 07/12/23

After finishing Ethics, turning it in, and updating my Mahara entry, I started working on my methodology and literature review while continuing to search literature. I chose to develop an XMind diagram grouping each literature source into corresponding topics in order to better organise my literature search into a logical structure that would be easier to follow when I return to the project after a break.

After finishing my xMind diagram, it became more and more clear to me how my study was influenced by a variety of themes and how I could broaden it to offer a more critically acclaimed view of the field. This will be really helpful to me in visualising the goal of my software. An updated view of the new xMind diagram can be viewed below:

A diagram of a project

Description automatically generated

As a result of this, I’ve determined that more research into general cryptographic methodologies, malware risks, disciplinary action, and how Artificial Intelligence can be utilised (to a reasonable extent) within my software. I currently have 19 sources in my bibliography, I intend to gather at least 25, or 30, before I begin a draft writeup of my “Literature Review and Methods” assessment report.

Looking into Cryptography more specifically, I wanted to start by researching how a selection of cryptographic techniques are utilised in different enterprise environments. As a first, I started off with “Use of distributed ledger technology by central banks: a review”, by (Del Río, C. 2017); to which Del Rio writes: “DLT arrangements use cryptography for several purposes, such as identity verification and data encryption”. Furthermore, a collection of users can either utilise public-key encryption, or private key, methodology to transfer data; though the latter is more often used. “A participant may create a digital signature with its non-shared cryptographic credential called a private key”, other participants of this arrangement will then “act as validators of transactions” and “verify authenticity of the ledger entry by decrypting it with a mathematical algorithm” (hash) “…and the asset owner’s publicly available public key” (Del Río, C. 2017).

Although not too specific, this provides an extra bit of content for thought. A software like mine, with LAN technology, could use a securely maintained list of private keys to periodically assess the authenticity of a file in relation to a hash algorithm in use. If this is wrong, a flag can be raised to investigate the discrepancy. A few similar ideas to mine have been published in the past, Anthony Harrington and Christian Jenson write how “keys” can be used to restrict file access, and then a “log-structured file system” can be used to manage false entries, and detect any malicious activity (Harrington, A. and Jensen, C, 2003). However, an issue that they come to realise is that “asymmetric cryptography is too slow – several orders of magnitude too slow”; as such, a new mechanism on which a symmetrically modified version of an algorithm is instead provided. A Log system like this requires that “the client signs log entries”, with digital “signatures”. They claim that they’re using a new “extension that uses symmetric cryptography to encrypt and decrypt data, whilst asymmetric cryptography is reserved for generation and verification of digital signatures” (Harrington, A. and Jensen, C, 2003).

The use of a “digital signature” is integral to the functionality of their proposed system, as a signature guarantees authenticity, and aids a server in detecting any false or malicious log entries; whilst using the speed of a symmetric encryption medium for decent network performance. I noticed going back through this that this methodology of encrypting files within a logged-based file system, and the constant, integral usage of digital signatures mimicked significant similarities to modern blockchain implementation. A blockchain instead using a “nonce” to verify individual blocks as signatures.

After a brief investigation (as a foundation into how cryptography can be used within a ledging technology), I’ve decided to investigate further into the malware risks of a blockchain, or a Distributed Ledger Technology (DLT). This can give me a basis to which I can formulate how I can design my software to protect itself against similar attacks.

# 08/12/23

In general, blockchains at a fundamental level are designed in such a manner that malicious entry, modification, is quite tough in comparison to standard database storage, and ledgers. Firstly, I had to get an idea of a few malicious attacks that blockchains are vulnerable to; including the likes of identity theft, and the “51% attack”, I got such information from a web article titled: “Are blockchains immune to all malicious attacks?” (Xu, J.J, 2016). Starting a bit of research into what exactly both a “51% attack” and “identify theft” was, in the context of a blockchain, let me to find that this is a situation within a blockchain whereby “a group of miners control more than 50% of the network’s mining hashrate, or computing power” (Ye, Congcong et al. 2018). As a result, a state like this can pose quite a few risks towards the overall stability of a network. “Attackers can complete proof-of-work quicker than honest miners”. To add on to this: “The attackers would be able to prevent new transactions from gaining confirmations”. This final point is without a doubt the most significant of all those made here, if a situation like this were to arise in a LAN-based ledger like mine; file editing transactions could be malicious modified to hide changes, or scapegoat others; or even entirely remove any trace of a file being changed at all.

Another issue that I found can also be a bit of a nuisance for blockchain’s, and awful for more widespread decentralized chains is “double spending”. Given that a blockchain is a decentralised system, ledging can’t be stored on a singular device, “so that one party cannot take control of all transactions in the network” (Aggarwal, S. and Kumar, N. 2021). This is a pertinent issue for me now, if a program of mine is going to store a ledger in a graphical interface (which can be moderated by those of a higher authority), the ledger, as a blockchain, must be distributed across all LAN-connected devices. This means all compliant devices must have my software installed, and running, all the time in the background to operate correctly.

Given that a blockchain is also a decentralized system, we must also enforce a level of access control to ensure attackers are blocked, or removed, as quickly as possible. Either this can be accomplished with scripting within the program a user is going to use (i.e w/ account, password), or with blockchain-based access control, which has been proposed. Although I’m not entirely sure as to the scale and difficulty of developing a blockchain for my ledger, let alone an access control system. For this reason I’ll stick with a secure account, password style login. These two variables of data can also be securely encrypted with AES-256, a modern encryption standard.

# 11/12/23

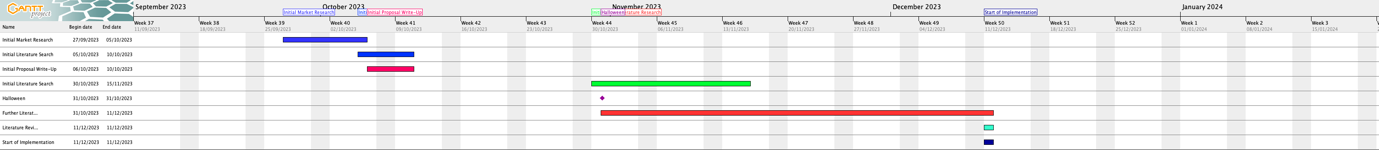
At first thought, ledging software could theoretically be compromised by spyware, a ransomware of sorts; or even my program could be disguised as a trojan (and manipulated by a third-party to operate an unintended script of commands). Network transfer data can be encrypted, and decrypted, as it’s received on different nodes in a LAN. A private key stored on one end of a network can decrypt files. How can my software be protected against trojans though? A thought that springs to mind is that an external verification tool can be downloaded, and used to authenticate the installed program. If It’s different in any way – a user can reinstall their software.

Furthermore, this is a newly updated version of my xMind diagram: “indi\_projectTHEMES.xmind”:

A diagram of a project

Description automatically generated

Likewise, I’m also going to update a Gantt chart of mine that I’ve had for a few months - to provide a visual time line of the progress I‘ve made:



With this all complete, I can finally set about getting a start with the Literature Review and Methods draft. Alhough the hand-in isn’t a requirement until Janaury in the new year, I would like to have a presentable document by the end of this week. I would like to show my supervisor a draft of it, and show other work I have done (including my research), to ensure that I am on the right path.

A fairly immediate issue that also dawned on me as I began a draft writeup of my A2 assessment is that I didn’t actually have a name set in stone for my software. With a slight hand from ChatGPT, I’ve decided on the name: “AudiCrypt” (OpenAI, 2023). In terms of a descriptor to go with it, I have a few ideas in mind:

* “A cryptographically verifiable ledging software based upon blockchain technology”
* “A blockchain-based ledging software, with cryptographic verification features, for use in local sensitive environments.”

# 12/12/23

I’ve made a decision on a descriptor I am going to use for the project, this being one with a hint of inspiration from my initial idea:

*“AUDICRYPT: A CRYPTOGRAPHICALLY VERIFIABLE LEDGING SOFTWARE FOR USE IN LOCAL SENSITIVE ENVIRONMENTS, WITH BLOCKCHAIN TECHNOLOGY”*

# 13/12/23

I’ve continued writing up my Literature Review and Methods document over the last day, having reached a point where my report now has an introduction, a collection of both aims and objectives, and a section detailing how I’ve employed a thematic approach into my research. A few examples of themes I have implemented have been detailed in numerous time-stamped sections above. Given the Literature Review is quite a bulky section, and I haven’t yet completed my UX Design assessment – I’m going to save this work until either this weekend, or next week, where I can also begin a design of the software implementation ahead.

# 19/01/23

After taking quite a long break away from the project, I’ve decided to make a start again with my Lit Review document.

# 17/03/23

I’ve made a return to this Project, deciding to begin simultaneously writing “Design/Implementation” segments (i.e. bc “Artefact” hand-in is 8th APR); going to get Figma design out the way, initial Python (or other languages(?)) researched, before basic skeleton coded. As part of research, I came across “cryptidy” (Jong, N.-O. de, n.d.) – claiming it makes “encryption/decryption of any Python object as simple as possible”, allowing both symmetric and asymmetric encryption techniques. An added feature I particularly liked was one claiming it could “unload AES key from memory as soon as possible”, to reduce risk of “memory attacks”, this library also enforces my AES-256 requirement.

Example of code provided to ‘pip’ includes:

A screenshot of a computer code

Description automatically generated

Upon testing, I found “cryptidy” to be somewhat rudimentary. Its “PyPi’, and “GitHub” pages are exactly the same, and don’t intricately detail anymore functions apart from those detailed in the Figure above, therefore I feel this isn’t a wise choice. Another library I had found was one called “Fernet”; described as a library that “guarantees…a message encrypted using it cannot be manipulated or read without the key” (Cryptography.io, 2014). At first glance, it looks quite a barebones library; though it Implements both symmetric, asymmetric encryption. However, some pages do claim, that on the flip side, that “Fernet” is actually a somewhat effective module (Stack Overflow, n.d.). That same forum describes the use of a “hazmat” module, using “GCM” to produce AES-NI keys up to “gigabytes” a second faster than those like “Fernet”.

Upon further digging, a clear consensus between a majority of those using Python cryptographically, is that a library called “pycryptodome(x)” is the best solution for most. On the page’s “GitHub” site (Eijs, H. (2022), “pycryptodome(x)” is described as a “self-contained Python package of low-level cryptographic primatives”. On It’s “PyPI” page (Eijs, H, n.d.), it claims it also has support for modules (i.e “GSM, CCM, EAX”), AES-NI, and support for other various hashing algorithms (e.g. “SHA-3”). A vast majority of those using cryptography within Python claim that this is one of the best libraries for this purpose. A dilemma I have though is combining this with LAN functionality (securely).

This also made me contemplate (in more detail) the anatomy of my App. To make my mind up on how a LAN in my program should be constructed, I decided to evaluate both advantages, and disadvantages of various LAN-based topologies (LAN Topologies Figure 2-1 Unicast Network Server Client, n.d.). It seems that a “Unicast” system, organised into a “Star” topology. This raised a new question, how would one node take charge of a LAN’s switch, to record new data changes? This node will act almost as a server but is cryptographically locked with a mixture of keys from all different devices.

# 24/03/24

Well, before I get into how a server should be organised in a network like this; I’m aware of my use of FTP, IP, and TCP (described as most common in LAN) (StudySmarter UK, n.d.). FTP has a few Python implementations, i.e. ‘pyftpdlib’; “Extremely fast and scalable Python FTP

server library” (Rodola, G, 2024). I soon had another dilemma I had to resolve, this was whether to transform this into a client/server -> web app; or retain a native app but integrate TCP/IP functionality into it. Hassan (and co.) describe in their Paper, that “asyncio”, the “standard library module” works with both “UDP and TCP” protocols, albeit not as advanced as other libraries like “Twisted”, it may still be ideal for my project (Hassan, G. M., Hussien, N. M., & Mohialden, Y. M, 2023).

With a clear plan going on in my head – I needed to deduce how I’d visualise my software. Although a majority of my past experience has been with those like “Tkinter”, I’ve personally found it to be somewhat rudimentary, but as of furthcer research; it turns out it has modern GUI capabilities, and can be customised to maximise aesthetics. Other libraries include “PyQt5”, described as a library that uses the “Qt” framework, originally “written in C++”. It ensures a user can use Python and still feel the speed of C++ (build-system.fman.io, n.d.); ‘PyPi” describes “PyQt5” as a “comprehensive set of Python bindings for Qt v5” (Limited, R.C, n.d.). Similarly, there is also another library called “Kivy”; based off a similar programming language (i.e. Python-interpreted); “GeeksForGeeks” describes it as an ”opensource multi-platform GUI Development library for Python” that also has capabilities to run on “iOS, Android, Windows, OSX, and Linux” (GeeksforGeeks, 2020). “Kivy” does provide a very modernised approach to UI Design, and has vast scope for development; much more so than similar alternatives, like “Tkinter”. I plan on testing my design with a few libraries, than making my final choice – though “Kivy” does look like a strong contender”.

A similar problem I had thought of whilst writing this was whether a user wanted to share a directory in the Cloud. Although this isn’t a direct requirement of my task, it’s a thought that could be implemented into post-release/maintenance steps. AWS, and Google, use API’s that can be configured with Python for use of Cloud storage. I’ll go back to this further in development if time permits.

Once I had an idea for this in place, I started making a Figma design. I intend on producing mid-high fidelity (i.e. basic animation); no executions, just showcase. I got started on a Low-Medium Fidelity prototype to get myself going (can be found in “/Design/Low-Mid Fidelity”.

# 27/03/24

Furthermore, it also dawned on me that a primitive Figma design is only useful in terms of how GUI will be portrayed. To showcase how basic CLI, source code would operate I’d have to use tools like “draw.io” to draw out HTA, and flowchart, diagrams. Although HTA and flowcharts are similar, I’ll employ the former more so to design tasks function purposes individually, and the likes of a flowchart for how a function can be used to manipulate data.

# 30/03/24

My Figma Design has been updated, with greater fidelity and more features; portraying basic home screen layout. I also looked in retrospect towards my initial project objectives to get a perspective as to a potential layout I could use for my Artefact. First and foremost, it became evident that an administrator/client system must be implemented in terms of software design; two versions of one program. FTP protocol must also be administered to ensure server connectivity, i.e. server connection where database can be stored, modified, and protected. In future hypothetical iterations Cloud connections can be formulated.

It also dawned on me that I required a set of libraries capable of finding changes to a filesystem. After some research, I discovered a few (though differentiated by OS type); i.e. “FSEvents” for macOS (ideal for me with one); though Windows and Linux use their own variants; like “inotify” (Linux), and “FileSystemWatcher” on Windows. A few libraries do exist (e.g. “entr” (Radman, E, 2024))j and “fswatch” – which seems like a far more exceptional choice in comparison to the former; using all trio of API’s in one library, it can be used cross-platform, it also contains native Python support (Crisostomo, E.M, 2024). Examples of its code being implemented can be visualised in its “PyPi” page, as seen below:

A screenshot of a computer

Description automatically generated

Another point I wanted to examine within Python is the use of a digital signature; though this is more so covered in detail in my cryptography section earlier. Then I also researched how a “hybrid database” could be implemented; such a database can be described as a combination of a blockchain and a database, a “revolutionary type of blockchain that mixes both worlds, both private and public blockchain” (GeeksforGeeks, 2023). “Private” nodes are utilised to implement validation, and other forms of encryption; “Public” nodes are in charge of managing communication with members of a network and other users who aren’t in direct connection with a LAN established by the software.

With a bit of research, it seemed apparent that some form of direct Pythonic implementation did not exist; rather it can be manifested with a mixture of other Python libraries.

A site that caught my eye during research into how a hybrid blockchain-like distributed database can be implemented within Python was a site called “BigChainDB”; coining itself as the “blockchain database” (BigchainDB, n.d.). A few of the key features it claims to implement include: “decentralisation”, “query”, “BFT”, “low latency” and “rich permissioning”. It seems like a coherent mixture of both a relational database and a fully-functional blockchain system. It seems it can be used in conjunction with tools like “fswatch” to record and store cryptographically-verifiable receipts in chronological order to ensure changes to files can be traced, without the worry of malicious database tampering. It also provides documentation for a “Driver” that can be used for its implementation into a Python script (docs.bigchaindb.com, n.d.). it seems mostly straightforward, though since a number fo other nodes in a LAN are going to have to use a “root URL” to connect to one node holding a database, I’m hoping such functionality is available through other libraries researched already, such as “asyncio” for instance.

A screenshot of a computer

Description automatically generated

***BigChainDB site features:***

Before getting into the design of a GUI in Figma, I had to ensure I had a comprehensive palette system going before I delved into manifesting different objects, and functions. Not only that, I have to think about animations, accessibility measures, simplicity, and consistency throughout my entire app. In reference to earlier, “Kivy” does integrate “.animation()” methods into “Widgets” (kivy.org, n.d.).

Another issue I had to encounter was that of a malware system in my software. With a bit of research, I found an open-source library called “clamd” (Grainger, T, n.d.); this library is based on another software, and a daemon used in the background, called “ClamAV” – or more specifically its malware and antivirus engine. On its website, ClamAV is described as having a few features that can be used in a range of software, as its engine can be imported into a wide array of languages (docs.clamav.net, n.d.).

i

A screenshot of a computer

Description automatically generated

***ClamAV feature site:***

# 05/04/24

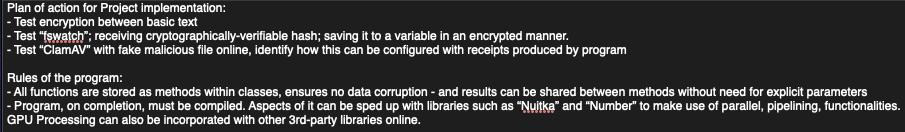
Going further into design, I looked into creating a flowchart for how a user could go about implementing a LAN within the Python client. However, it must also be recognised that this could either be an Admin user or a standard client. Another process includes establishing cryptographic contact with all other nodes in a LAN, and then establishing a combination of FTP, SSL, etc.. contact from all nodes in star topology.

# 09/04/24

Going back to my project, I had a decent Low-Fidelity graphical prototype for my software; but, in-line with my Spiral regulation of my project, I don’t’ intend on modernising this prototype too much yet – and only prioritise a matter like this only if I ever get to a stage of creating a GUI at some point during the development cycle. At this rate, that doesn’t look too likely.

# 11/04/24

With my “draw.io” designs getting better, I’ve decided to make a start on my implementation – I collected all my final libraries into one document, and I set about making a plan of action of how this project will make it from a “draw.io” flowchart into an usable program. My initial points began to look like this:



I wanted to, first and foremost, I had a perfectly working set of libraries in my collection; to fully comprehend how they work with one another, and whether an alternative might be necessary.

During a few installs of libraries I had listed, I encountered an error creating “BigChainDB”; some “.json” download errors arised, preventing “BigChainDB”s wheel being constructed; providing a fatal error to “pip”.

A screenshot of a computer program

Description automatically generated

This is a preview of the error I had.

After a bit of Googling, I decided that trying to install this on Mac became too much of an issue, I wasn’t entirely sure what the error was – even after following a segment of the documentation I hadn’t noticed before, in “cryptography.io”’s guide, referencing in their “Building cryptography on macOS” section, that a segment of code as shown below:

A green rectangle with black text

Description automatically generated

..is a prerequisite to installing “BigChainDB”. After further testing, I still fell into the same problems. I’ve decided to use a Ubuntu VM to continue development, in the hope of more success (cryptography.io, n.d.). After configuring my new VM with a new Python virtual environment, I tried installing all packages I had listed earlier. Again, another attempt at an install didn’t work. At this stage I was worried if there was a problem at all with “BigChainDB”; and moreso, its dependencies – and whether they were Python3.12 compatible.

# 13/04/24

At this stage, I was struggling to get into a start for my implementation because of a few issues I had with the “BigChainDB” package; I gave it another go with the installation, but this time trying with [Python@3.10](mailto:Python@3.10) (i.e. older Python version) to see if there was any difference. Unfortunately, this resulted in another error. After a bit of digging for some closure, it soon dawned on me that this may be as a result of a lack of contribution to the Python version of the project on GitHub by their developers for the last few years (GitHub, n.d.). I set about finding an alternative to this. I knew it wouldn’t be an exact replica, but I felt that even if it had come to a stage where I had to use a pretty common library – I’d still be able to use my cryptographic libraries to manipulate data in such a way that I could still chronologically trace data with cryptographic proofing capabilities.

I started researching into “PostgreSQL”; and compared this with similar alternatives on the market like “MongoDB”. PostgreSQL seemed quite comprehensive, and therefore a good option for me to take forward into my project.

# 14/04/24

I started a project on “PyCharm”, and pushed it into a GitHub repository that will serve as a version control mechanism throughout this project. I first made a start on a “Plan of Action” document, that can be seen below:

A screenshot of a computer

Description automatically generated

Adding another section in the middle will further describe a methodology I will employ during development. This section can be found below:

A screenshot of a computer

Description automatically generated

At a glance of the last line, I soon realised that I had a new dilemma that I had to get a grip of, and that was how I could use both “pgcrypto” as a tool within my project. I will use a combination of both public, private-key cryptography on both server/client; I feel I could use private hash of admin user with public hash of server to ensure only admin’s can create receipts; ensuring culprit can’t tamper with produced receipt. Question was how to stop a user from editing a receipt stored on database; to do this – I theorise on storing on public server a combination of both public server hash (i.e. SHA-512 or SHA-256) w/ private hash of admin user. Admin user doesn’t have access to database either, will be completely read-only.

With this in mind, it got to a stage where I know two programs have to be made – one where a server is established on a device. This will act as a service that can constantly be contacted by other nodes in the LAN. Another client will also be distributed that will act as both an admin and a standard user login.

So I set about on creating a new script, that simultaneously establishes both a PostGresSQL database, and a server with the “twisted” library, that can accept connections, and listen to any requests made. These requests will be hashed, timestamped, data transactions made on a given directory in the server. This can be set as the entire server directory.

For the purpose of testing, I intend on utilising a new Virtual Machine installed with “Ubuntu Server’

# References:

**Legend**:

Green = included in “indi\_projectTHEMES.xmind”. (i.e used in Lit Review, Methods doc.)

Red = used in further Dissertation (A3 Dissertation.docx document.)

Amazon Web Services, Inc. (n.d.). Blockchain on AWS - Amazon Web Services. [online] Available at: [https://aws.amazon.com/sblockchain/](https://aws.amazon.com/blockchain/).

Amazon Web Services, Inc. (n.d.). Amazon QLDB. [online] Available at: <https://aws.amazon.com/qldb/?c=bl&sec=srv>.

Amazon Web Services, Inc. (n.d.). Amazon QLDB Features. [online] Available at: <https://aws.amazon.com/qldsb/features/?pg=ln&sec=hs>.

AWS (2018). Amazon CloudWatch - Application and Infrastructure Monitoring. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/cloudwatch/>.

transparency.dev. (n.d.). An open-source append only ledger | Trillian. [online] Available at: https://transparency.dev [Accessed 4 Oct. 2023].

GitHub. (2023). Trillian: General Transparency. [online] Available at: https://github.com/google/trillian [Accessed 4 Oct. 2023].

docs.aws.amazon.com. (n.d.). Quotas and limits in Amazon QLDB - Amazon Quantum Ledger Database (Amazon QLDB). [online] Available at: https://docs.aws.amazon.com/qldb/latest/developerguide/limits.html [Accessed 8 Oct. 2023].

transparency.dev. (n.d.). Trillan helps you reliably log all actions performed on your servers | Trillian. [online] Available at: https://transparency.dev/application/reliably-log-all-actions-performed-on-your-servers/#limitations [Accessed 8 Oct. 2023].

GitHub. (n.d.). transparency-dev. [online] Available at: https://github.com/transparency-dev [Accessed 10 Oct. 2023].

Quiniou, Matthieu. Blockchain : The Advent of Disintermediation, John Wiley & Sons, Incorporated, 2019. ProQuest Ebook Central, <https://ebookcentral.proquest.com/lib/bcu/detail.action?docID=5781105>.

Godfrey-Welch, Darlene; Lagrois, Remy; Law, Jared; Anderwald, Russell Scott; and Engels, Daniel W. (2018) "Blockchain in Payment Card Systems," SMU Data Science Review: Vol. 1: No. 1, Article 3.

Sandhu, A. “Using Blockchain as Secure and Immutable Storage.” Faculty of Computing,

Engineering and the Built Environment, 2018. Print.

Langsch, V. “Speechless a Virtual Personal Assistant.” Faculty of Computing, Engineering and the Built Environment, 2018. Print.

OpenAI (2023). ChatGPT. [online] chat.openai.com. Available at: <https://chat.openai.com>.s

Garg, Rishabh. Blockchain for Real World Applications. Hoboken, New Jersey: John Wiley & Sons, 2023. Print.

Pang, Yue et al. “Blockchain-Based Reliable Traceability System for Telecom Big Data Transactions.” IEEE internet of things journal 9.14 (2022): 1–1. Web.

wiki.hyperledger.org. (n.d.). Hyperledger Fabric - Hyperledger Fabric - Hyperledger Foundation.

Qingyi Zhu, Seng W. Loke, Rolando Trujillo-Rasua, Frank Jiang, and Yong Xiang. 2019. Applications of Distributed Ledger Technologies to the Internet of Things: A Survey. ACM Comput. Surv. 52, 6, Article 120 (November 2020), 34 pages. <https://doi-org.bcu.idm.oclc.org/10.1145/3359982>

Sarmah, S.S., 2018. Understanding blockchain technology. *Computer Science and Engineering*, *8*(2), pp.23-29.

Del Río, C. (2017). Use of distributed ledger technology by central banks: A review. Enfoque UTE, 8(5), p.1. doi:https://doi.org/10.29019/enfoqueute.v8n5.175.

Harrington, A. and Jensen, C., 2003, June. Cryptographic access control in a distributed file system. In *Proceedings of the eighth ACM symposium on Access control models and technologies* (pp. 158-165).

Xu, J.J. (2016). Are blockchains immune to all malicious attacks? Financial Innovation, 2(1). doi:https://doi.org/10.1186/s40854-016-0046-5.

Ye, Congcong et al. “Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting.” 2018 5th International Conference on Dependable Systems and Their Applications (DSA). IEEE, 2018. 15–24. Web.

Aggarwal, S. and Kumar, N. (2021). Chapter Twenty - Attacks on blockchain☆☆Working model. [online] ScienceDirect. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0065245820300759

Troy, S. (2021). distributed ledger technology (DLT). [online] SearchCIO. Available at: <https://www.techtarget.com/searchcio/definition/distributed-ledger>.

Jong, N.-O. de (n.d.). cryptidy: Python high level library for symmetric & asymmetric encryption. [online] PyPI. Available at: <https://pypi.org/project/cryptidy/>.

build-system.fman.io. (n.d.). *PyQt5 tutorial 2020: Create a GUI with Python and Qt*. [online] Available at: https://build-system.fman.io/pyqt5-tutorial.

‌

Cryptography.io. (2014). Fernet (symmetric encryption) — Cryptography 2.9.dev1 documentation. [online] Available at: https://cryptography.io/en/latest/fernet/.

Stack Overflow. (n.d.). Is the Fernet cryptography module safe, and can I do AES encryption with that module? [online] Available at: https://stackoverflow.com/questions/36117046/is-the-fernet-cryptography-module-safe-and-can-i-do-aes-encryption-with-that-mo [Accessed 20 Mar. 2024].

Eijs, H. (2022). Legrandin/pycryptodome. [online] GitHub. Available at: https://github.com/Legrandin/pycryptodome.

Eijs, H. (n.d.). pycryptodome: Cryptographic library for Python. [online] PyPI. Available at: <https://pypi.org/project/pycryptodome/>.

LAN Topologies Figure 2-1 Unicast Network Server Client Client Client. (n.d.). Available at: https://cdn.ttgtmedia.com/searchNetworking/Downloads/NetConsultantsch2.pdf [Accessed 20 Mar. 2024].

StudySmarter UK. (n.d.). Local Area Network: Meaning, Topology, Protocols & Security. [online] Available at: https://www.studysmarter.co.uk/explanations/computer-science/computer-network/local-area-network/.

Rodola, G. (2024). giampaolo/pyftpdlib. [online] GitHub. Available at: https://github.com/giampaolo/pyftpdlib [Accessed 24 Mar. 2024].

Hassan, G. M., Hussien, N. M., & Mohialden, Y. M. (2023). Python TCP/IP libraries: A Review. International Journal Papier Advance and Scientific Review, 4(2), 10-15. <https://doi.org/10.47667/ijpasr.v4i2.202>

Limited, R.C. (n.d.). *PyQt5: Python bindings for the Qt cross platform application toolkit*. [online] PyPI. Available at: <https://pypi.org/project/PyQt5/#:~:text=PyQt5%205.15.10&text=PyQt5%20is%20a%20comprehensive%20set>.

‌GeeksforGeeks. (2020). *What is Kivy?* [online] Available at: <https://www.geeksforgeeks.org/what-is-kivy/>.

‌Radman, E. (2024). eradman/entr. [online] GitHub. Available at: https://github.com/eradman/entr [Accessed 30 Mar. 2024]

Crisostomo, E.M. (2024). emcrisostomolfswatch. [online] GitHub.

Available at: https://github.com/emcrisostomo/fswatch [Accessed 30 Mar.

2024].

GeeksforGeeks. (2023). Hybrid Blockchain. [online] Available at: <https://www.geeksforgeeks.org/hybrid-blockchain/>.

BigchainDB. (n.d.). Features & Use Cases • • BigchainDB. [online] Available at: <https://www.bigchaindb.com/features/>.

docs.bigchaindb.com. (n.d.). BigchainDB Python Driver — BigchainDB Python Driver 0.6.2 documentation. [online] Available at: https://docs.bigchaindb.com/projects/py-driver/en/latest/ [Accessed 2 Apr. 2024].

kivy.org. (n.d.). Animation — Kivy 1.11.1 documentation. [online] Available at: <https://kivy.org/doc/stable/api-kivy.animation.html>.

Grainger, T. (n.d.). clamd: Clamd is a python interface to Clamd (Clamav daemon). [online] PyPI. Available at: https://pypi.org/project/clamd/ [Accessed 3 Apr. 2024].

docs.clamav.net. (n.d.). Introduction - ClamAV Documentation. [online] Available at: <https://docs.clamav.net/>.

cryptography.io. (n.d.). Installation — Cryptography 43.0.0.dev1 documentation. [online] Available at: https://cryptography.io/en/latest/installation/#installation [Accessed 11 Apr. 2024].

‌

GitHub. (n.d.). Unable to install bigchaindb-driver package from pip · Issue #511 · bigchaindb/bigchaindb-driver. [online] Available at: https://github.com/bigchaindb/bigchaindb-driver/issues/511 [Accessed 13 Apr. 2024].

‌

‌

‌

‌

‌

‌

‌

‌