



BIRMINGHAM CITY  
University

# **CMP6200/DIG6200 INDIVIDUAL UNDERGRADUATE PROJECT 2023–2024**

**A3: Dissertation**

## **AUDICRYPT**



Course: Integrated Master (MSci) of Computer Science  
Student Name: Matthew Potts  
Student Number: 21124021  
Supervisor Name: Shadi Basurra

# Abstract

In modern-day enterprise, data control, and policy alike, are essential components to ensuring legislation and discipline is properly enforced. However, although the risk of malicious attacks more often than not stem from external sources, an off chance does simultaneously exist where it can also originate from within. AudiCrypt is designed to resolve this loophole, by ensuring all actions within a sensitive-minded Local Area Network are chronologically traced, and can cryptographically-verifiable to entirely ensure Non-Repudiation. It ensures that everyone is answerable to their own actions, and can be disciplined as necessary if their actions are found to be in breach of company data policy, national or international legislation. Simultaneously, it offers a means of defence against internal malicious plots, all whilst being completely scalable to any size of network – given a server capable of operating PostgreSQL is within the network. This paper describes my research into the exciting field of Cryptography, my Design, Implementation, and Evaluations of AudiCrypt. A low-overhead, fast program – capable of running on any modern operating system, results visualise its ease-of-use, tight security, and comprehensive nature of receipts – which provide several valuable insights into multiple user's particular actions – that can be essential in determining the outcome of an investigation, and protecting the integrity of a network and its enterprise.

# Acknowledgements

Firstly, I would like to thank Shadi Basurra – for his supervision throughout the entirety of this Project. He has provided vastly ranging insights into my work, that has helped shaped the purpose, and structure, of my Project, and this dissertation. He has assisted me in realising the methodology in which this software can be installed, and the various routes I could investigate to help me piece together the jigsaw that is a program in Python. I'd also like to thank my partner – Tiyah Singh – for keeping me in one piece in what has been one of the most difficult periods of my University tenure, both inside, and out, of campus.

# Table of Contents

Abstract .....	2
Acknowledgements .....	3
Glossary .....	5
List of Figures .....	6
2.1 Quote list for Amazon Web Services (AWS) Quantum Ledger Database (QLDB): .....	6
Introduction .....	7
1.1 Problem Definition .....	7
1.2 Scope .....	7
1.3 Rationale .....	8
1.4 Project Aim and Objectives (Update this after coding) .....	8
2 Literature Review .....	9
3 Method and Implementation .....	12
3.1 Methodology: .....	12
3.2 Design: .....	12
3.3 Implementation: .....	12
3.3.1 Overview of Libraries To Be Used: .....	13
3.1.2 How will a “cryptographically-verifiable” receipt be structured? .....	14
3.1.3 Other Services I plan on integrating within “AudiCrypt” .....	15
3.4 Overview of Discussion and Beginning of Development .....	16
3.5 Programming implementation: .....	16
4 Evaluation .....	23
4.1 Evaluation Methodology .....	26
4.1.1 Evaluation Metrics .....	26
4.1.2 Baseline systems .....	<b>Error! Bookmark not defined.</b>
4.1.3 Dataset .....	<b>Error! Bookmark not defined.</b>
4.2 Results .....	26
4.3 Discussion .....	27
5 Conclusions .....	28
6 Recommendations for future work .....	29
7 fReferences .....	30
8 Bibliography .....	33
9 Appendices .....	36
9.1 Appendices One – Gantt Chart: .....	36

Word Count: 10,034

# Glossary

Product-as-a-Service	A centrally-hosted service, or product, that is distributed to a user; and is regulated – with updates by a third party. Can also be connoted to as an “on-demand” service.
AArch64	The 64-bit variant of the ARM Architecture. A new form of CPU Architecture being released rivalling “x86-64”. ARM is based on RISC set.
x86-64 (amd64)	64-bit variant of x86 instruction set used most often by Intel, AMD in their CPU’s. x86-64 is based on CISC set.

# List of Figures.

## 2.1 Quote list for Amazon Web Services (AWS) Quantum Ledger Database (QLDB):

Amazon Web Services, Inc. (n.d.)

	Price
Write I/O's	\$0.70 per 1 million requests
Read I/O's	\$0.136 per 1 million requests
Journal Storage Rate	\$0.03 per GB-month
Indexed Storage Rate	\$0.25 per GB-month
<i>Prices correct as of April 2024</i>	

## Introduction

In a simplified manner, my project aims to construct an open-source tool capable of producing both cryptographically-verifiable receipts of file transactions; and similarly, including a functionality to pipe this into a chronological, immutable, and critically – encrypted – database; that can be enforced to regulate data policies, and detect foul play, in a local environment. I intend on producing this software with enterprise use chiefly in mind – though in practice, it can be used in any Local Area Network with a server running popular Linux, or Windows, server distributions.

Enterprise policies can be more stringently enforced by offering a definitive method of proving who altered a particular file and how much they altered it. This is because disciplinary action can be readily reinforced with tangible proof of wrongdoing. This software will also serve as a deterrent; ideally, people will be discouraged from pursuing their malicious intentions if they are aware that this software is quietly operating in the background.

This report will be distinguished into nine independent segments. These include a “Literature Review”, a linguistic review of all theoretical research I’ve undertaken in preparation for my project – in order to understand various mechanisms, theories, and practices I can coordinate with one another to implement in my final Artefact; a discussion of my Methodology, my various Designs, a discussion of my Programming Implementation; an Evaluation, and a collection of tests I assembled; a Conclusion, and a discussion of how I could hypothetically advance this project further.

### 1.1 Problem Definition

In modern-day enterprise, practically all form of information is stored, and manipulated, digitally. As such, political institutions have legally enforced frameworks to uphold. These include the likes of GDPR (GDPR, 2013), and the Data Protection Act (DPA) 2018 (GOV, 2018). Enterprises often implement their own form of digital policies, including computer misuse ones. As such, it’s a pretty essential prerequisite that some form of transaction log be maintained.

However, there is a fundamental problem with this – receipts of data transactions stored on an enterprise’s database, without any form of security measures protecting them in place, are at risk of malicious manipulation without any form of record this even occurred. Certain pieces of metadata, such as names, and timestamps, can be modified to disguise genuine culprits.

As such, certain individuals can hypothetically deny any accountability towards a form of foul play, claiming a defence of third-party manipulation to cover their tracks, protect them against any form of disciplinary action, and yet be provided with a platform to further their malicious intents in the enterprise – all whilst under the guise of being a perfectly legitimate member.

Furthermore, institutions found to have breached articles of either GDPR or the DPA – due to negligent issues as discussed – can receive serious penalties as a result. Enterprises can face grave reputational damage to their brand, legal action against them, regulatory sanctions, further operational disruption due to investigations, and financial penalties. GDPR “enabled Data Protection Authorities (DPA’s) to fine companies up to 4 percent of their annual revenue in the event they were found in violation of the regulation’s requirements for data collection, processing, and use” (Wolff, J. and Atallah, N., 2021).

### 1.2 Scope

This was a tough section to go about in truth. Given how this is the first, major, project I have undertaken on this level of scale – it was difficult to gauge how far I could go with this software – whilst taking into consideration the various time constraints I had placed upon myself. Given this, I knew right from the get go – the likelihood of a Graphical User Interface being implemented were very slim – though in light of this, this left me time to invest in a colourful, welcoming terminal user interface.

I had also planned on integrating a form of blockchain technology into this – after research into “BigChainDB” (which is mentioned later) – and its various alternatives, it quickly dawned on me the difficulty of a theory like this in practice. Although an essential aspect of this Project was the

chronology of this logging practice, I felt that in order to ensure other requirements of my Project were implemented to an adequate standard – the blockchain theory had to be dropped – in order to make way for a system of timestamping within my PostgreSQL Database.

Moreover, although this Project – in theory – ideals to take into account different forms of data policy – this is still a feature that can be implemented in the future. My software has all the groundwork for a practice like this, but to formulate it around an entire policy would require an entire refactoring of my code structure – an activity I simply don't have the time to undertake.

### 1.3 Rationale

I feel that this is a problem predominantly tackled only by proprietary vendors, these being the likes of Amazon Web Services (AWS), and Google Cloud. Being a Cloud-based service, or more specifically – a Product as a Service (PaaS); both businesses set fixed quotes as to what extent an institution can use their service. For instance, AWS's variant of an enterprise-orientated cryptographically-verifiable log, called "Quantum Ledger Database" (QLDB) – provides similar features to my implementation – including a cryptographic log being "immutable", and "append-only"; whilst including a form of "centralisation", a complete "system-of-record application" that can access, and dissect, all stored records (Amazon Web Services, Inc, n.d.). A predominant issue with this though is pricing. Small-scale enterprises, who may have just started up, might not have adequate funding to invest in QLDB; a PaaS that might not even cover all their specific requirements.

Mindful of this, I wanted to create a tool that doesn't implement any form of pricing restrictions, can be expanded in scale to cope with enterprise development, straightforward enough so that it can integrate into any network (preferably of a LAN topology), and cryptographically robust enough so it protect the database, and data within it, against modern-day threats.

### 1.4 Project Aim and Objectives (Update this after coding)

Fundamentally, I propose an open-source compiled Python-based freeware capable of running on all modern operating system distributions, with a primary intention of maintaining a cryptographically-verifiable log of securely-hashed receipts of data transactions. These receipts provide a cryptographically-secure chronology of all manipulations made to all files within a specified directory on a network's server. The lack of commercialisation makes this software exemplary for any type of environment, establishes true data repudiation, implements data legislation, policy, and will act as a vital asset in disciplinary investigations.

A comprehensive list of objectives I envisaged is detailed below, ordered in most significant, to those less so:

- Create cryptographically-verifiable receipts for every transaction made to a particular file on a server's directory.
- Ensure every data transaction is made answerable to a singular user.
- Ensure an enterprise's data policy is regulated throughout.
- Ensure every receipt contains comprehensive-enough detail so it can be enforced to its greatest extent in a disciplinary investigation.
  
- Receipts must be stored in a logical, cryptographically-secure format. All logs must be encrypted, to which only a designated "Admin" or a client of authority has the permission to access them; also must be robust from external attack.
  
- Frequently require review of certain receipts, in order to suspect new trends emerging.
  
- Utilise anti-malicious definitions from third-party libraries to examine receipts for any discrepancies.
- In further updates, a Graphical User Interface (GUI) will also be implemented, showcasing all transactions, their users, changes they've made, and a showcase of the log entirely.
- This GUI will also visualise a process of exporting these receipts as evidence that can be used in an investigation.



## 2 Literature Review

Initially, I had quite a wide scope of thoughts, and theories, that I had mentally culminated as a benchmark going into this Literature Review. My initial search into this topic stumbled me across certain keywords, like “Cryptographic Integrity” in databases, and “Cryptographic Integrity in Data Ledging Systems” – which had progressed me into the theory of “Blockchains”, and their parent – a “Distributed Ledger Technology”. Blockchains use the concept of a “Hash” in extensive fashion, and I found this to be an aspect of them to be admirable – especially in protecting user data, and establishing secure links with a remote server and a client. Due to this, I researched further into Cryptographic methods – more noticeably, asymmetric cryptography. Logins, and RSA signatures, could be completely formulated within my program if each user stored a Private Key of their own, generated in real-time in a secure location. I found this to be a far more desirable practice than storing Private Keys either on a server, or a Client’s machine – as users can use other forms of Secure Storage mechanisms to securely house their Private Keys – outside of any risk of leakage within my Project configuration.

I also set about my research on a number of different platforms, these being the likes of “BCU Library Search”, and “IEEE Xplore”. The vast array of content they publish provides me a platform to leverage authenticity, and relevance, of a source towards my Project in far greater detail than most other sites. I made sure to research similar real-life case studies too, including those like “QLDB” at AWS, and “Trillian”, with Google – both are elaborated further on in the document. I also researched malware within Networks, concepts such as a “51% attack”, and how integral “redundancy” is as a concept to databases.

Similarly, another theme is “Blockchain”, which I’ve taken significant inspiration from. Although a little broader than those just mentioned, this is due to me looking into real-world case studies regarding Blockchain – including malicious events, and how specific configurations, in complement to the framework of a Blockchain itself – has promoted successful software developments in past years; an example of this being “Blockchain in Card Payment Systems” (Godfrey-Welch, Darlene; Lagrois, Remy; Law, Jared; Anderwald, Russell Scott; and Engels, Daniel W, “Blockchain in Payment Card Systems, 2018) – whereby blockchain aid in providing a “single tamper-resistant digital ledger”, coupled with “hash pointers used to record encrypted transactions in a structured manner”. My research into Blockchain framework’s also rooted further particular research into similar framework’s such as Hyperledger Fabric, as described in the last section ([wiki.hyperledger.org](http://wiki.hyperledger.org), n.d).

Likewise, two similar topics I’ve researched into include both Malware, and Cryptography. Regarding the latter, more specifically implementing “Access Control into a Distributed File System” (Harrington, A. and Jensen, C, 2003, June); where concepts including “client-side” encryption, combined with both asymmetric and symmetric cryptography; and synchronised server, and client-side authentication – where data is only transmitted once provided encryption methods are completed on both machines. Similarly, Malware intertwines well into this; providing concrete evidence that no Blockchain, regardless of precautionary measures – is entirely secure. Instances of “51% attacks” – where “a group of miners control more than 50% of the network’s mining hashrate, or computing power” (Ye, Congcong et al. 2018) – have enormous repercussions if exploited. Due to my project’s inherently sensitive nature, it’s of the highest priority to ensure vulnerabilities, as such, are comprehensively ironed out before release.

Finally, Artificial Intelligence constitutes a final theme of research; more specifically regarding how forms of AI could be integrated within my project to develop greater comprehensive data analysis – dictating whether specific user activity can be regarded as malicious; as touched on before. For context, I investigated Villaim Langsch’s “Speechless: A Virtual Personal Assistant” – where I used concepts like their “Speech Recognition Engine”, in its object-orientated framework, as an ideal cornerstone for AI usage in my project (Langsch, V. “Speechless a Virtual Personal Assistant.”, 2018).

Firstly, I investigated similar case studies, in relation to ledging – with integrated cryptographic technology involved in some fashion – either in SaaS distribution (i.e. Cloud), or as standalone applications; furthering into both AWS’s “QLDB” or Google’s “Trillian”. Progressing in regards to “QLDB, or “Quantum Ledger Database”; otherwise described as a “fully managed ledger database”,

providing “transparent, immutable, and cryptographically verifiable transaction logs” (Amazon Web Services, Inc, n.d.) - being a Software as a Service (SaaS), it’s capable of maintaining a “sequenced history of every application data change using an immutable and transparent journal” – with “ACID”-enforced integrity. “QLDB” serves significantly in providing indispensable inspiration for my project, including a benchmark for what features should be available, how it could develop, and how mine can distinguish itself as a practical alternative for a wide demographic looking for greater data security options.

In terms of how it operates, it shares similar taste in hashing, and cryptography methods; employing SHA-256 hashing with QLDB’s bespoke API to prove integrity of any data change. To maintain zero data redundancy, given that this is in database format – “QLDB” enforces Full “ACID” checks; ensuring “transactions have full-serializability”, “atomicity..., isolation, and durability properties” (Amazon Web Services, Inc, n.d.). Albeit, it claims QLDB can “easily scale up and execute 2-3x as many transactions as common blockchain frameworks”; I have reason to dispute – as numerous sources claim that despite evidently slower read/write speeds of a blockchain framework; “blockchains are faster than databases when it comes to verifying transactions” (sanvignesh, 2023), although this is discussed in greater depth in the next section.

In regards to Cryptography, I specialised my research predominantly into how it can affect User Access Control in Local Area Networks -more so because it’s particular environments, like that, I intend on distributing this software into. Likewise, I also branched a similar path into researching how Cryptography can be implemented in mainstream enterprise environments – starting with Del Rio explaining that “DLT arrangements...” (as mentioned earlier) “...can use cryptography for several purposes, such as identity verification and data encryption” (Del Río, C. 2017). Similarly, this also branched into how both symmetric and asymmetric encryption can be used to authenticate data transmission with “digital signatures”, or “private” keys.

This follows into Harrington and Jenson’s discussion, as briefly mentioned, into how Cryptography can be enforced as User Access Control (UAC) in Local Area Networks, in tandem with both symmetric, and asymmetric, encryption methods. Investigating into how “keys” can be utilised to restrict file access – they theorised how this could be coupled with a ledger-based file system to record false entries; which could be spurred into necessary disciplinary investigations if flag trends are perceived in software. An issue discovered though, with significant relevance – in particular – to myself, is that “asymmetric cryptography is too slow...several orders of magnitude too slow” in relation to file transfer, and authentication with the method (Harrington, A. and Jensen, C, 2003, June).

Instead, both advise that in particularly high-demand filing environments – a mechanism employing both symmetric and asymmetric hand-in-hand is more ideal; where a Log system whereby data is transferred symmetrically, and “asymmetric cryptography is reserved for generation and verification of digital signatures” (Harrington, A. and Jensen, C, 2003, June).

Malware, a momentous aspect of this project, predominantly concerns the risk of malicious activity to the fundamental anatomy of a Blockchain-like framework being implemented in this project. Although risks associated with cryptographic techniques do exist; I find those directly affecting Blockchain stability itself more significant. A “51% attack”, for instance, refers to a situation where “a group of miners control more than 50% of the network’s mining hash rate, or computing power” (Ye, Congcong et al. 2018). To put this into perspective, this correlates to either a single user, or a minority of those in a Blockchain controlling over 50% of a Blockchain’s computational power – a pertinent issue where “honest miners” are at greater risk of being exploited. Likewise, “the attackers would be unable to prevent new transactions from gaining confirmation” (Ye, Congcong et al. 2018) – in which case if a situation like this were to arise on a Local Area Network, file-editing transactions could be modified, and used to scapegoat others.

Another issue, though not technically a malicious form of file – but rather an activity – is “double spending”. Although not entirely commonplace in Blockchain frameworks, more so for decentralised networks; this can cause data redundancy, and files being mistakenly modified due to naming, or metadata issues. Given that I intend to store my ledger on a Blockchain, “one party cannot take control of all transactions in the network” (Aggarwal, S. and Kumar, N. 2021). In retrospect, this infers

a pertinent dilemma into how moderation will be organised, as it must be ensured that no particular node group has overwhelming control to framework transactions.

## 3 Method and Implementation

### 3.1 Methodology:

Following on from A2, describing a *modus operandi* I set about on implementing throughout this project; I intend on enforcing a Spiral-like approach. A few functional requirements I'd drafted aren't entirely necessary to implement early-on, ensuring other requirements; like implementing Local Area Network asymmetric cryptography can be constructed, and tailored, early on in the process. A mechanism, like creating a Graphical User Interface, in a generalized perspective is nowhere near as significant as implementing a robust encryption mechanism.

Building on to a Spiral methodology, I first intended on establishing embryonic prototypes in Figma, a free designing tool – these designs are discussed in more detail in the “Design” section below. With a basic prototype created, I plan on moving on to providing a functional skeletal template, that will serve as the foundation to the greater software I'm creating. This is a cycle, and will continue repeating throughout this project; until eventually, I will be able to showcase both several high-fidelity prototypes, and a well-defined, vigorous final Artefact to be presented.

Furthermore, to elaborate in greater depth the intricacies of my methodology; my software requires no expensive, unrealistic apparatus. All devices in a Local Area Network must be capable of Wi-Fi networking, and modern computers are easily capable of handling both advanced encryption, and hashing, algorithms (e.g. AES256, SHA256). This software is designed to be deployed to an enterprise demographic, to which a majority of these will, most likely, comply with my requirements. I intend on developing with Python, though there is a slight chance I might venture into miniature snippets of foreign code (e.g. Java, C++) if mechanisms operate at greater capacity compared to their Pythonic alternatives.

### 3.2 Design:

Having had, practically, zero experience at cryptographic practice within Python; I knew very little about libraries providing such functionality within Python. I wanted to make a head start with a decent, brawny Figma design; complete with both a comprehensive, and intricate, delve into different features included within the app; how they combine simultaneously with one another – together to create a harmonious, smooth, and most importantly – secure – experience for users in question.

At first, I was a bit lost as to how I should approach Figma. I'm aware such an initial prototype doesn't require intricate fidelity; but on its flip side, since I want to hit the implementation side of my project with the ground running, so to speak, I conceptualized a medium-to-high fidelity prototype. Functionality within this prototype wasn't of a high concern at this stage – though I do intend on fulfilling further accompanying documentation.

To contextualize certain processes within my software, which are typically encapsulated to the eyes of a typical user, I plan on employing flowcharts to describe specific features. Class diagrams, for instance, I could use to elaborate how both my App will be displayed, and how certain functionalities are built – and likewise, how they cooperate with one another to build a final product.

Furthermore, An initial Figma (low/medium-fidelity) prototype I designed can be found below in Appedices Two.

As I mentioned earlier, having had little experience cryptographically with Python – I invested a good time period into comprehensively researching differing libraries that offer modernized encryption techniques (i.e. SHA-256 hashing, AES-256 encryption, both asynchronously and synchronously), whilst having compatibility for networking – which itself is another field I have had a lack of experience in.

### 3.3 Implementation:

First and foremost, before I made a start in terms of programming; I wanted to ensure that I had a clear plan of action for the project ahead. This would be utilised, in conjunction with past Figma and

“draw.io” designs I had created beforehand as a roadmap I could use to chronologically order my development. An early example of my plan of action can be seen below in Figure 1 and 2:

```
Plan of action for Project testing:
- Test encryption between basic text
- Test "fswatch"; receiving cryptographically-verifiable hash; saving it to a variable in an encrypted manner.
- Test "ClamAV" with fake malicious file online, identify how this can be configured with receipts produced by program

Rules of the program:
- All functions are stored as methods within classes, ensures no data corruption - and results can be shared between methods without need for explicit parameters
- Program, on completion, must be compiled. Aspects of it can be sped up with libraries such as "Nuitka" and "Numba" to make use of parallel, pipelining, functionalities. GPU Processing can also be incorporated with other 3rd-party libraries online.

Structure of program:
- Database created with PostgreSQL, initially hosted locally for testing; then use another computer in Local Area Network for further testing.
- Create "clams" service that runs in background on a specified directory that a group of users are working on. This will be a local directory at first, "pycryptodomex"; maybe "hashlib", with "pyftplib" will be used to securely show remote Admin directory
- Implement Receipt functionality by taking variables (i.e. userID, file name, mods with "fswatch", import hash into receipt, and sign with private key with "pycryptodomex")
- Create a login service that takes a username, password - and 2FA of some sort, with hashing (i.e. "hashlib", or "pycryptodomex")
- Initialise form of FTP LAN network with "twisted"; can do with some form of a web app or a local GUI client; could also be done with "sockets" library; all fetched with "pip".
- PostgreSQL EAR (Encryption-at-rest w/ pycrypto)(?)
- Create a cryptographically-verifiable chain of data transactions posted to PostgreSQL database;
  - Make a table within PostgreSQL designated to storing data transactions on specified data directory in chronological order; immutable; hashed records (expanded in "dev_logbook")

- First and foremost, it is also important to get a login mechanism established; details of this to be shown below:

Server program structure:
-

Libraries being used in project:
- pycryptodomex
- nuitka
- numba
- twisted
- pyftplib
- asyncio
- clamAV
- fswatch
- biachaindb
-----x-----
- kivy (i.e. if I get near GUI stage, unlikely)
- kinter
- PyQt5
-----x-----
```

Figure 1 and 2: A basic Plan of Action document I drafted before my implementation.

### 3.3.1 Overview of Libraries To Be Used:

To elaborate further, one can distinguish the above into five sections; one being an initial list of tests I expect to implement upon completion of my software. Although these tests are in a bit of an embryonic state at this given moment, like with a majority of everything else in this document – I plan on developing it further as time progresses. Secondly, I've included a primitive list of “Rules” I've enlisted. This ensures consistency, third-party readability, maximised efficiency and performance, and defensive – in a sense that data won't get corrupted, and can't get modified during runtime by those with malicious intent. It also helps make the development process somewhat more coherent for me, as I will be jumping my mind to and from this project – to other assessments I have during this period.

Furthermore, a pretty comprehensive third segment entails my program's structure. Likewise, I plan on expanding this with time, as my own personal understanding of my program's realistic requirements become apparent. Firstly, I explicitly state that a “PostgreSQL” database, of relational format, will be implemented. During development, and further testing, I envisage on using a Virtual Machine (VM), on Parallels, a “hardware visualisation client” for macOS ([www.parallels.com](http://www.parallels.com), 2021). This VM will utilise Ubuntu Server 24.04 LTS for the ARM64 (AArch64) architecture. I did this for simplified hardware virtualisation, and greater performance, because I use an M2 ARM-based Mac as my main computer. Following on from this, to materialise my earlier-stated anti-malware requirement (see “1.4 Project Aim and Objectives”) – I enlisted “ClamAV”; an “open-source (GPLv2) anti-virus toolkit”, with additional functionalities including a “flexible and scalable multi-threaded daemon, a command line scanner and advanced tool for automatic database updates” (docs.clamav.net, n.d.). I found this quite a decent option to use, because of how it can be implemented on a file-to-file basis, is regularly kept up-to-date, and supports a wide range of file formats. Not only that, “the core of the package is an anti-virus engine available in the form of a shared library” (docs.clamav.net, n.d.). Due to this, I researched into some form of Python driver, via “PyPi” that I could install, via “pip”. Fortunately, “PyPi” hosts “clamd” – a “portable Python module to use the ClamAV anti-virus engine on Windows, Linux, MacOSX, and other platforms” (Grainger, T, n.d.). Grainger also mentions that a “running instance of the clamd daemon” is a requirement. This seemed like an exemplary choice for my software – and outsourced my malware requirement, saving me precious development time.

Thirdly, as I go on to mention thereafter; I intend on using either, or maybe even both depending on how implementation pans out, “hashlib” and/or “pycryptodomex” – with “pyftplib” – to securely display the contents of a remote server file directory to a user. Security is a major virtue in this, as “AudiCrypt” is designed to be entirely secure from unauthorised third-party access, and tampering – as such, any kind of contact between a server and a client will be symmetrically-encrypted. In terms of libraries being used, “hashlib” is a module built-in to Python – from 2.5 (Python documentation, n.d.) – “that implements a common interface to many different secure hash and message digest algorithms”, including those like “SHA-256”, and “SHA-512” (docs.python.org, n.d.). However, a key difference with “hashlib” and “pycryptodomex” is that the latter not only includes vast support for various hash types, such as those “hashlib” implement, but also other, more unfamiliar hash types, like “SHA-224”, “SHA-384”, and “SHA512/256” – a combination of both (pycryptodome.readthedocs.io, n.d.). Alternatively, I personally identify “pycryptodomex” as a far more meaningful library that I intend on employing abundantly throughout – due to its well-renowned encryption (symmetric, asymmetric), digital signature, key derivation, and secret sharing (Shamir) capabilities (www.pycryptodome.org, n.d.).

Furthermore, another library I’ve spoke about is “pyftplib” – described as a “high-level portable interface to easily write very efficient, scalable and asynchronous FTP servers with Python” (Rodola, G, n.d.). I intend on using this to display the contents of a directory on a server within a LAN configured by the Admin of a group. This library also provides users within this Network a means of opening, saving, and configuring files on their local server. All communications will be securely transmitted symmetrically, with “pycryptodomex”.

Moreover, the key premise of this Artefact is a software capable of producing cryptographically-verifiable receipts of data transactions – as such a method of examining a given directory for any changes made to particular files must be implemented. Fortunately, an open-source library titled “fswatch” exists that serves this precise purpose. Crucially, “fswatch” is multi-platform. According to their ‘readme.md’ document on the GitHub repository, it makes use of similar API’s across multiple operating systems to achieve similar goals. For instance, it utilises macOS’s “File System Events” API, “kqueue” on FreeBSD systems, the “File Events Notification” API of Solaris, “inotify” within the Linux kernel (and thus, all its distributions), and arguably most importantly – the “ReadDirectoryChangesW” API on Microsoft Windows (Crisostomo, E.M, 2024). Although Crisostomo does continue to mention a caveat in Microsoft Windows’ “ReadDirectoryChangesW” API being that it can only monitor directories, rather than individual files – this shouldn’t be too much of an issue, rather “fswatch” results can be filtered to only include the name of a file in question.

### 3.1.2 How will a “cryptographically-verifiable” receipt be structured?

However, the question does remain – how do I go about constructing a cryptographically-verifiable receipt? I drafted an embryonic concept in my mind. In contrast to Figure 1 and 2 above, I made a change of plan not long into the implementation process. I knew I had two predominant principles I had to abide by, these being my receipts can’t be tampered with by any user – and must be accessible to an Admin (or any trusted authority) at any time; and all receipts of data transactions made to a particular file must be chronologically-ordered on a database.

In light of this, my early receipt methodology went as follows:

- I’ve decided to use SHA3-256 as a standard. Although SHA-256 is widely used throughout the industry, being a “FIPS 180-4” National Institute of Standards and Technology (NIST) “*Secure Hash Standard*” hashing algorithm (NIST, 2017). However, contrary to wider knowledge, NIST simultaneously instigated “FIPS 202” – both of which were made “effective on August 5, 2015” (unblock.federalregister.gov, n.d.). Furthermore, NIST state that use of SHA-2 hashing algorithms is only as a “minimum” prerequisite. SHA-3, and its Extended Output variants can also be utilised as a standard – although it does state that “currently there is no need to transition applications from SHA-2 to SHA-3” (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, 2015).



So why SHA3-256? Firstly, this futureproofs AudiCrypt's secure hashing capabilities. Secondly, "-256" algorithms utilise "32-byte words"; ensuring backwards compatibility with 32-bit machines (Sectigo® Official, n.d). Thirdly, SHA-3 algorithms are also "designed to resist other attacks, such as length-extension attacks" (Dworkin, Morris J, 2015) – an issue that is well-documented about on "pycryptodomex"'s documentation (pycryptodome.readthedocs.io. n.d.).

- Moreover, SHA3-256 will securely hash the contents of a file when discovered by "fswatch". This hash will also be affected by other metadata, like a timestamp.
- During the creation stage of a user's account, a private and a public key will be generated. However, only the "public" key will be stored remotely. The private key will be generated during runtime, using a mixture of details regarding their user credentials and their machine; this private key will thus be utilised in creating a digital signature.

However, the issue now is the choice of a digital signature algorithm. Ordinarily, NITS produced another standard, "FIPS 186-5" – published in February 2023, it formulates a consensus as to certain digital signature algorithms to be enforced; these being both "RSA" and "ECDSA" (National Institute of Standards, and Technology, 2023). One could find "ECDSA" a more desirable algorithm to use in this case, due to its efficacy with lower bits; "a common RSA 2048-bit public key provides a security level of 112 bits. However, ECDSA requires only 224-bit sized public keys to provide the same 112-bit security level" (Anon, n.d.). However, "RSA" seems like an ideal algorithm for me personally – "RSA" is an industry standard, albeit "NITS" standardised. "ECDSA" is still a relatively new algorithm, which I may very well use in later iterations of this software.

- With a file hashed, a public and private key created for individual users, and a means of generating the latter on-the-fly and storing public keys remotely in an encrypted space on a local server. Conclusive details, such as a user's signature, timestamp, a hash of the file in question, an order ID, and a combination of both a hash of the receipt and a signature by a Client ensures all processes are cryptographically-verifiable from start to finish. Once receipts pass through the Admin, they're automatically stored in an encrypted area of storage on a local server, via similar hashing, signing, and encryption methods discussed previously in this section.
- In the event that an Admin must disclose the contents of a receipt for either a routine check, or a suspected case of foul play, AudiCrypt will also provide a secure means of transferring a user's public key to an Admin's machine – in order to cryptographically-verify signatures, and hashes.

This can be achieved via a secure HTTPS connection between a client and a server; paving a foundation for a SSL/TLS handshake to be accomplished. This can be facilitated with a 'twisted' instance running on a local server; with both a user's public key securely stored in a '.pem' file, and a self-signed authorising SSL certificate in a '.crt' file. This self-signed certificate will be entirely unique to a local network. To ensure this works, this software does require "OpenSSL" as a dependency before runtime, though OpenSSL can easily be installed via a user's terminal on several operating systems.

### 3.1.3 Other Services I plan on integrating within "AudiCrypt"

Furthermore, AudiCrypt will also provide a secure login service for clients returning to their machine. Public keys are stored in separate databases, though foreign keys from an encrypted user credential database will provide a link. All user credentials will undergo secure SHA3-256 hashing, that can be checked against during login, via 'twisted'. To maximise security, I plan on storing at least two security questions for each user – though these will be stored in a separate database, with a foreign key link. Both the questions, and answers, will be hashed with SHA3-256.

A major aspect of this software, though – is how a user interacts with files on a connected server, via FTP. I found that although “Twisted” has a robust selection of HTTP and SSL/TLS-related capabilities; it struggled in basic tasks such as providing a list of files in a specified directory. I found that “pyftplib” was a more intuitive alternative in this regard. Due to this, I’ve decided to use both in my final implementation. Although both servers can use a single IP Address, they’ll have to utilise two separate port values; for example one could initialise a HTTP server, via ‘twisted’ on Port 80 (unless blocked by a third-party or policy) – and have their FTP server operate on Port 20 (or 21), a standard for FTP across most machines ([www.winzip.com](http://www.winzip.com), n.d.). However, users are permitted to use several dynamic ports of their own, so long as they’re not required, or reserved, by their operating system. Ports, often between 1024, and 49,151 – are externally managed by a third party called the “Internet Assigned Numbers Authority” (IANA) (Arubanetworks.com, 20a23).

### 3.4 Overview of Discussion and Beginning of Development

Finally, now that we have a comprehensive idea of how a number of different concepts, and their dependencies operate behind the scenes – I’m going to discuss how I plan on integrating this into a single “cryptographically-verifiable” log, with “PostgreSQL” (PGSQL), on one’s local server installation. During my research into PGSQL’s documentation; I stumbled across “Write-Ahead Logging” (WAL): described ironically as a “transaction log” – it plays a very similar purpose to the core concept of this software (Annadanum, N, n.d.). Being a “Write-Ahead Log”, this also infers that data is written to a log ahead of such data being integrated into a database. It serves predominantly as a means of recovery – “to ensure that when there is a crash in the operating system or PostgreSQL or the hardware, the database can be recovered”. This somewhat serves as a “two bird, one stone” form of solution, ensuring my database is protected against tampering, power loss, and serves as a means of recording past changes to specific files, and directories. These WAL logs can also be filtered via “psycpg2”, or “paramiko” – a “pure Python...implementation of the SSHv2 protocol” ([www.paramiko.org](http://www.paramiko.org), n.d.) – with SSH commands corresponding to SQL queries interpreted via PGSQL. Given it’s also based on SSL, it includes encryption as default (Cloudflare, 2023). Each record will also be cryptographically hashed, with SHSA3-256, to include timestamps; these can be decrypted, and filtered, with SQL statements – via “paramiko” – to produce a list of transactions to a specified file or directory in chronological order.

The table containing a list of all data transactions will be secured with User Access Control (UAC). Only clients in a specified group, i.e. “Admin”, will be permitted to view records, and access WAL logs. I intend on also encrypting the WAL directory. In similar fashion to public, and private keys, a keypair for the encrypted WAL log directory will be stored in an encrypted table accessible only to those in the “Admin” group. Only Admins will be able to access and retrieve data from these WAL logs.

Now we can move on to a timeline of my programming implementation.

### 3.5 Programming implementation:

First, and foremost, I set about creating a login tool. This is an essential aspect of my program, and allows for all of my users to establish a secure connection with the database, only once they have passed thorough checks. I’m going to go section by section, explaining individual aspects one-by-one.

```
import ipaddress
import os, sys, subprocess, re, time, signal
from tqdm import tqdm
from colours import Colour
from reg import RegexFilters as rf
from secQ import SecurityQuestion as sq
12 usages  by Matthew Potts
class Constants():
    SLEEPTIME = 1
    AUDICRYPT_TABLES = ['uKeys', 'userCreds', 'cryReceipt']
    HIDE = True
```



To begin 'ui.py'; my initial file, to which all others are formulated off, I imported a select few native Pythonic libraries; these being those like "os"; which can handle "operating system dependent functionality" (Python, 2019); a majority of these being signals, like "SIGINT", and "SIGTERM". These are measured in case a user throws these in attempts to quit my app, to which a "Handler" method in my UI class is called to handle these, and clear all global variables from memory (as a security measure); this will be detailed in due course. Furthermore, "subprocess" is also called upon for checking Python third-party dependencies, with "pip"; and for "fswatch" functionality, which is also described further along. 'tqdm' is a third-party library, which I intend on specifying as a prerequisite for my software; though this can be included if I decide on publishing a compiled variant of my software. 'tqdm' serves as a "smart progress meter" (PyPI, n.d.). 'tqdm' has been inserted on a 'for' loop within a method of this 'ui.py' file checking third-party dependencies, to visualize this process – that otherwise would just be a lengthy collection of 'pip' jargon.

A collection of classes, and methods, within these files can be found below.

```
class RegexFilters():
    regFilters = {
        'username': r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$',
        'password': r'^(?=.*[a-z])(?=.*[A-Z]{2,})(?=.*[@$!%*?&]{2,}).{12,20}$',
        'security': r'^(?=.*[a-z])(?=.*[A-Z]).{5,}$',
        'fullName': r'^[A-Z][a-z]\s[A-Z][a-z]*$',
        'ipADDRESS': r'^((?:[0-5]{0-5})|2[0-4][0-9]|01?0-9|0-9)?\.\.){3}((?:25[0-5]|2[0-4][0-9]|01?0-9|0-9)?)$',
        'portSSH': r'^(22|[1-9][0-9]{2,3}|[1-5][0-9]{3,4}|6[0-5][0-9]{3}|65[0-4][0-9]{2}|6553[0-5])$',
    }

if __name__ == '__main__':
    print(Colour.RED+"This .py file can only be executed as a module, access denied."+Colour.RESET)
```

17

```

1 usage: Matthew Potts
class SecurityQuestion:
    9 usages: Matthew Potts
    def securityList():
        return {
            '1': 'What was the name of your first pet?',
            '2': 'What was the name of the street you grew up on?',
            '3': 'What was the name of your first school?',
            '4': 'What is your mother's maiden name?',
            '5': 'What was the make and model of your first car?',
            '6': 'What was the name of your childhood best friend?',
            '7': 'What city were you born in?',
            '8': 'What is your favorite movie?',
            '9': 'What is your favorite book?',
            '10': 'What is your favorite food?'
        }

if __name__ == '__main__':
    print(Colour.RED+"This .py file can only be executed as a module, access denied."+Colour.RESET)

```

Figure 3.3: An image of “secQ.py” – a .py file containing a list of 10 security questions that can be used by both an Admin and a Client – in both registering, and logging, into their account.

```

class Colour():
    MAGENTA = "\033[35m" # Magenta
    RESET = "\033[0m" # Reset
    RED = "\033[31m" # Red
    GREEN = "\033[32m" # Green

if __name__ == '__main__':
    print(Colour.RED+"This .py file can only be executed as a module, access denied."+Colour.RESET)

```

Figure 3.4: An image of “colours.py” – a .py file containing a class with several ASCII colour combinations, these will be used vastly throughout to categorise certain ‘print’ outputs into errors, guidance, or any other form of message.

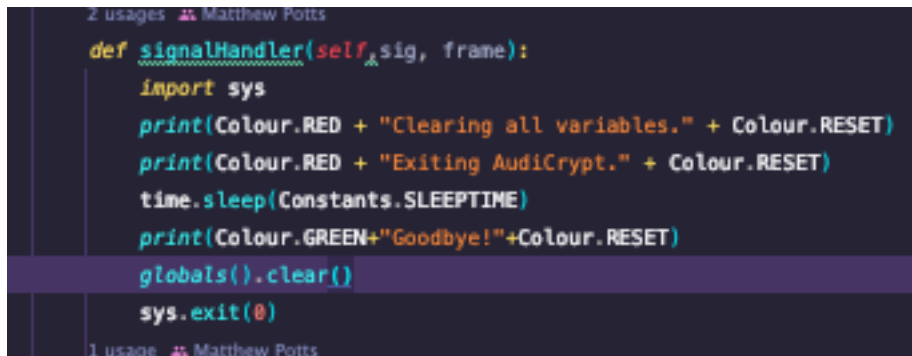
All Figure 3 classes are constricted in a practically-identical method, unescorted by any form of “\_\_init\_\_()” method that typically instantiate both ‘self’ – a “current instance of the class” ([www.w3schools.com](http://www.w3schools.com), n.d.); and any kind of data (i.e. that can be defined by data type if preferred). Given that neither of these classes are also instantiated by a variable outside of their class, and a basic ‘if’ statement at the bottom – checking whether a user has run the given .py file directly, or whether it’s being used as an import; variables within them can easily be called, just with the name of their class being a prefix – e.g. “GREEN” in “Colour()” (‘colours.py’) would become ‘Colours.GREEN’ in any .py file with ‘colours’ imported.

Furthermore, to add context to both ‘reg.py’ (Figure 3.2), and ‘colours.py’ (Figure 3.4) – ‘reg.py’ contains a dictionary for a majority of inputs ‘ui.py’, and related .py files, accept – throughout their course of execution. To name a few examples, keys “username”, and “password” are both designed to filter a String a user inputs to evaluate whether it’s at least 8 characters long, and contains a combination of lowercase, uppercase, and special characters. I felt it would make sense to implement this in a Regex sense, given that Regex is standard across all Python distributions, can easily be modified to a range of security requirements, and isn’t very computationally-taxing on a majority of computers.

Returning to ‘ui.py’, in Figure 3.1; this segment of code also details a new section called “Constants”. Although this class has been instantiated with a new instance, that will be discussed in a little more

detail towards the footer of this section, it can be used by just calling the class name – a practice that I have undertaken of an interchanging basis at various stages of ‘ui.py’. For example, to slow down “print()” commands displayed in quick succession in a terminal, I’ve employed ‘time.sleep()’ – ‘time’ being a native Python library; this function, though, is passed with “SLEEPTIME”; a constant value (i.e. specified in capital characters for third-party readability), that isn’t designed to be modified. This saves having to use an arbitrary value each time ‘.sleep()’ is called, and if I feel the flow of ‘ui.py’ is too fast in particular segments, a ‘+’ operator (i.e. with an arbitrary value, e.g. ‘3’) can be passed to slow it down – for better user comprehension.

I mentioned before of the use of a “signal handler”, designed to catch signal ‘processes’, such as “SIGINT”; I implemented this in a method called “signalHandler”, as visualised below.



```

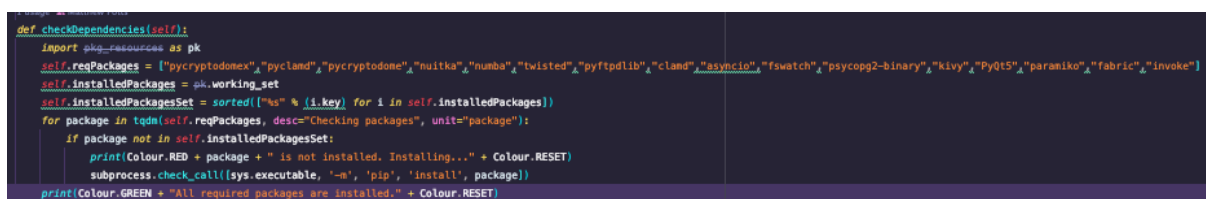
2 usages Matthew Potts
def signalHandler(self, sig, frame):
    import sys
    print(Colour.RED + "Clearing all variables." + Colour.RESET)
    print(Colour.RED + "Exiting AudiCrypt." + Colour.RESET)
    time.sleep(Constants.SLEEPTIME)
    print(Colour.GREEN+"Goodbye!" + Colour.RESET)
    globals().clear()
    sys.exit(0)
1 usage Matthew Potts

```

Figure 3.5 – A method titled “signalHandler()”; designed to check all ‘global’ variables with ‘globals()’, clear them from memory to prevent a memory extraction attack, and close the ‘AudiCrypt’ process with “sys.exit()”.

As discussed in Figure 3.5 – this removes all global variables stored in memory. One may also notice that this also requires a few arguments – ‘sig’, and ‘frame’; these specifying the nature of signal. This can only be made possible with “signal”, a native Python library responsible for acting on signals with “custom handlers” (docs.python.org, (n.d.); a default signal being “SIGINT” for instance, which is translated into a “KeyboardInterrupt”, so it can be piped into “signalHandler()” as “sig”.

In addition, my class – “UserUI()”, also contains a method called “checkDependencies()”, called upon in “checkLog()”, a method that will be discussed later. This method can be visualised below in Figure 3.6.



```

def checkDependencies(self):
    import pkg_resources as pk
    self.reqPackages = ["pycryptodomex", "pyclamd", "pycryptodome", "nuitka", "numba", "twisted", "pyftplib", "clamd", "asyncio", "fswatch", "psycog2-binary", "kivy", "PyQt5", "paramiko", "fabric", "invoke"]
    self.installedPackages = pk.working_set
    self.installedPackagesSet = sorted([i.key for i in self.installedPackages])
    for package in tqdm(self.reqPackages, desc="Checking packages", unit="package"):
        if package not in self.installedPackagesSet:
            print(Colour.RED + package + " is not installed. Installing..." + Colour.RESET)
            subprocess.check_call([sys.executable, '-m', 'pip', 'install', package])
    print(Colour.GREEN + "All required packages are installed." + Colour.RESET)

```

Figure 3.6: “checkDapedencies”, a method called upon in “checkLog()” responsible for taking a list of packages, and services, I require for a user to successfully utilise all features of AudiCrypt.

In Figure 3.6 above, a list of all packages required, including “pycryptodomex”, “asyncio”, and “fswatch” are listed above in a list called “self.reqPackages”, this is then iterated on, with “tqdm” visualising as a bar – a process with “subprocess” executing “pip -m install {library name}”; to which a result is printed to signify the end of this list, and for all package installations. This is a method that is called every time ‘ui.py’ is executed. It ensures no user is caught in a “missing packages” error during interpretation (or compilation error, with “nuitka”) error, during runtime.

Similarly, ‘def checkAV()’ takes this a step further. It checks as to whether a user has the “clamd” daemon running as a PID on their local machine. Although this is a requirement for “pyclamd”, and for AudiCrypt’s runtime; it must be installed via third-party instructions, stored online at the “ClamAV”

website mentioned earlier in this document. Again, this is a method of “UserUI()” that is called upon at every instance of runtime.

```
1 usage  Matthew Potts
def checkAV(self):
    try:
        self.oPut = subprocess.check_output(["clamscan", "--version"])
        if "ClamAV" in self.oPut.decode():
            print(Colour.GREEN + "ClamAV detected." + Colour.RESET)
            return
        else:
            print(Colour.RED + "ClamAV not detected." + Colour.RESET)
            return
    except subprocess.CalledProcessError:
        print(Colour.RED + "ClamAV not detected." + Colour.RESET)
        return
```

Figure 3.7: “CheckAV()” – an unexacting method, passing a subprocess to a user’s terminal checking to see whether a ‘—version’ argument exists in ‘clamscan’ – a feature of ClamAV. If an error exists in this process, a user is alerted.

A truly integral aspect of AudiCrypt is a local client’s ability to store, and upload (with SSH, i.e. “fabric”, and “paramiko”, as discussed later) securely “SHA3\_256” hashed data within memory, and on a remote PostgreSQL table. This is facilitated with “def hashString()”. A somewhat primitive method, bar further encapsulation by “pycryptodomex”; this method is responsible for returning a ‘.hexdigest()’, the “printable digest” of a message (pycryptodome.readthedocs.io, n.d.) – a cryptographically-secure hash, the components of which are discussed earlier, that hide plaintext. These hashes will be used within comparative operations with user-inputted credentials as a means of validation – without compromising user privacy in case a server is maliciously infiltrated.

```
18 usages  Matthew Potts
def hashString(self, string):
    from Crypto.Hash import SHA3_256
    self.hashObject = SHA3_256.new(data=string.encode())
    return self.hashObject.hexdigest()

Matthew Potts
def hashStringExDigest(self, string):
    from Crypto.Hash import SHA3_256
    self.hashObject = SHA3_256.new(data=string.encode())
    return self.hashObject
```

Figure 3.8: “def hashString()” – a somewhat basic method, it retrieves a String passed by a user, takes it into a new “SHA3\_256” object, courtesy of “Crypto.Hash” (a method within “pycryptodome”), and returns its “.hexdigest()” – so it can be used for storage, and comparisons, with variables on a remote server, with SSH and SQL queries.

AudiCrypt’s ‘ui.py’ contains two paths in terms of account management; these being the acknowledgement of whether a user is a new user, or a recognised (‘current’) user; either way – both must follow identical processes of data input, manipulation, and validation. “def curUser()” in Figure 3.9 below superlatively illustrates this in function. The method follows a logical order of a username, a password, two sets of security questions, and a user’s full name. Each instance of inputs are regulated by independent “While” loops, protecting ‘ui.py’ against “Exception” errors by catching false inputs.



```

def verifyLogin(self, vType):
    self.sshVal = {}
    while True:
        while True:
            self.host = input(Colour.MAGENTA + "Enter the IP (IPv4) address of your server: " + Colour.RESET)
            self.reg = rf.regFilters['ipADDRESS']
            if re.match(self.reg, self.host):
                print(Colour.GREEN + "IP address is valid." + Colour.RESET)
                self.sshVal['host'] = self.host
                break
            if not re.match(self.reg, self.host):
                print(Colour.RED + "Invalid IP address. Please try again." + Colour.RESET)
                continue
            else:
                print(Colour.RED + "Invalid value, please try again." + Colour.RESET)

        while True:
            self.port = input(Colour.MAGENTA + "Enter the SSH port of your server: " + Colour.RESET)
            self.reg = rf.regFilters['portSSH']
            if re.match(self.reg, self.port):
                print(Colour.GREEN + "Port is valid." + Colour.RESET)
                self.sshVal['port'] = self.port
                break
            if not re.match(self.reg, self.port):
                print(Colour.RED + "Invalid port. Please try again." + Colour.RESET)
                continue
            else:
                print(Colour.RED + "Invalid value, please try again." + Colour.RESET)
                continue

        while True:
            self.user = input(Colour.MAGENTA + "Enter the username of your server: " + Colour.RESET)
            if not self.user:
                print(Colour.RED + "Username cannot be empty." + Colour.RESET)
                continue
            print(Colour.GREEN + "Username is valid." + Colour.RESET)
            self.sshVal['user'] = self.user
            break

```

Figure 3.11: 'def verifyLogin()' is a method instantiated in the instance that a user is a returning ('current') user, whether they are either an 'Admin', or a 'Client', returning to their AudiCrypt configuration.

In Figure 3.11 above, one will notice that "self.sshVal", a variable, is initiated as a directory, to which – similar to 'def curUser()' in Figure 3.9 – will contain specific credentials – in this regard, however – to how a Client intends on using 'Fabric' – a far better, more coherent, alternative I found to "Paramiko" – a far more low-level, primitive, SSH API in comparison – to login to a remote server to execute specific 'psql' (PostGreSQL) SQL commands. A point to mention, here, is that although 'self.port' takes an input for a port to use for SSH connections on the remote server – I found after local testing, this is best left to Port 22 (as explained earlier in my 'Implementation' segment) – to accompany for as many use-cases as possible though, I left this as an open choice.





Figure 3.14: “def audiTableCheck()” takes these steps a little further, and utilises the “\dt” command to list all tables within a database, returning a Boolean as to whether one can be identified or not.

After both tests, to verify the integrity of a remote server, and whether past tables, or databases, can be identified; a login can now be verified. At this stage, a user’s login would have already been uploaded to a server – this process is covered, in further detail, further below in the document. By taking all parameters to check for, storing them in 3 individual lists, and iterating over them, in SSH, with SQL commands retrieving each segment of data for a user in a table called ‘uCredits’ (more information on that later); we can validate all pieces of login information safely. You may also point out that each piece of data is also hashed during its collection stage, its also stored remotely in this format. To any third party user, who could be investigating this maliciously, this would appear as a random scrabble of characters. In the ‘for’ loop, you’ll notice “self.vUserExec” ‘stdout’ output is heavily modified, by transforming it into a list. This retrieves only significant data, which is stored in index ‘5’ in all circumstances. If this fails, it returns the user to the main menu.

As a final layer of authentication, the software asks for a user’s Private Key. This is stored on a table called ‘uKeys’. By uploading a hash of an input a user makes (i.e. their Private Key), an SSH command can be executed to compare this against the value stored in a record corresponding, to themselves, in uKeys. Only if this is authorised does a user, regardless of whether they are an Admin or a Client, have full access to their AudiCrypt configuration. This Private Key will also be used for digitally signing receipts, with RSA.

Depending on whether they are a ‘Client’, or an ‘Admin’, this program breaks off. If one is an ‘Admin’, for instance, it’ll call the ‘AdminUI’ class in ‘adminUI.py’. This is demonstrated in Figure 3.15.

```
if self.comp == True:
    print(Colour.GREEN + "Private Key verified." + Colour.RESET)
    print(Colour.MAGENTA + f"AudiCrypt login complete. Welcome {(self.fullName.split())[0]}!" + Colour.RESET)
    self.ssh.close()
    import adminUI as aUI
    while True:
        aUI.adminUI()
        self.answer = input(Colour.MAGENTA + "Enter an option: " + Colour.RESET)
        if self.answer == "1":
            self.workDir = aUI.adminUI().getDic(self.sshVal)
        if self.answer == "2":
            aUI.adminUI().viewLog(self.sshVal)
        if self.answer == "3":
            aUI.adminUI().delConfig(self.sshVal)
        if self.answer == "4":
            aUI.adminUI().quitAudiCrypt()
        if self.answer == "5":
            aUI.adminUI().establishLink(self.sshVal)
        else:
            print(Colour.RED + "Invalid option. Please try again." + Colour.RESET)
            continue
```

Figure 3.15: If a user’s Private Key is correct, it will present a UI stored in ‘AdminUI.py’. Clients also have an alternative, in “ClientUI.py”.

Admin’s have 4 options, at the moment, that they can use – either to configure their configuration, or to view specific logs of receipts, live, within a graphical interface.



```

#set adminTableDetails=admin

#All values MUST be hashed with SHA3-256 before being inserted into the database.
#Insert all Admin details into the new database for the first time.

#To avoid confusion - the public key is NOT HASHED, the private key is HASHED, and the RSA signature is also HASHED.
#This is to ensure that the data is secure, and cannot be tampered with.
#This relies on a user having a secure password, and secure security questions. This is regulated by regex in "reg.py".
#The public key is stored in plaintext, as it is required for encryption/decryption.
#This also relies on the user storing their private key, and RSA signature in a secure location.
#When a user logs in, their public key is checked against a hash in the database,
#If the hash matches, the user is authenticated. If not, the user is denied access.

#Since the user has their private key, this will be kept securely stored in memory.
#Since only the admin has read access to the Crypted table, no one can read a user's signature either. Users can only write to this table.
#The admin can read the table, and verify the signature against the hash in the database.

self.uCredInsert = self.ssh.run command: f"sudo -w postgres psql -d audicrypt -- \INSERT INTO uCreds (name, phone, pword, sQIC, sQIA, sQICs, sQIA) VALUES ('{self.userDetails['FN']}','{self.userDetails['UN']}', '{self.userDetails['PW']}', '{self.userDetails['S']}"

if self.uCredInsert.ok:

    time.sleep(Constants.SLEEPTIME)

    print(Colour.GREEN + "Admin details inserted into uCreds." + Colour.RESET)

    self.puKeyStripped = self.puKey.replace(_del_ "-----BEGIN PUBLIC KEY-----", _new_ "")

    self.prKeyStripped = self.prKeyStripped.replace(_del_ "-----END PUBLIC KEY-----", _new_ "")

    self.puKeyStripped = str(self.puKeyStripped.strip())

    self.prKeyStripped = str(self.prKey)

    self.prKeyStripped = (self.prKeyStripped.split("-----BEGIN RSA PRIVATE KEY-----"))[1].split(
        '-----END RSA PRIVATE KEY-----')

    self.prKeyStripped = str(self.prKeyStripped[0])

    print(
        Colour.RED + "\n\nPLEASE REMEMBER/KEEP A NOTE OF YOUR PRIVATE KEY. YOU WILL REQUIRE THIS FOR LOGIN." + Colour.RESET)

    time.sleep(Constants.SLEEPTIME+3)

    print(Colour.MAGENTA + self.prKeyStripped + '\n' + Colour.RESET)

    print(Colour.MAGENTA + self.hashedKeygen(self.prKeyStripped))

```

Figure 3.16: 'def AdInitTableUpdate()' is a method called upon during an initial configuration of AudiCrypt by a new Admin.

In Figure 3.16, you'll notice 'self.uCredInsert' takes all variables within the 'self.userDetails' dictionary, all values of which were hashed, and inserts them into the 'audicrypt' database. It's also at this stage that all user-related keys are also uploaded. When either a Public or a Private key is established within 'pycryptodomex' – it contains headers, and footers, sectioning a Key into an area defining the beginning, and end, of a Key. To avoid storing this on our server, as there is a likelihood that a user won't copy this over – we use the '.replace()' method, with our String to get rid of it.

Another critical section of the Admin setup is creating all required tables, and establishing a clear “User Access Control” protocol within. This is formulated with ‘def adTableManip()’.

```

1 def adTableManip(self):
2     try:
3         # Manipulating a few parameters of the AudiCrypt tables to ensure no unauthorised access.
4         # This is to ensure that only the admin can read the cryReceipt table, and only the admin can write to the uKeys table.
5         # This is to ensure that the integrity of the AudiCrypt database is not compromised.
6         print(Colour.MAGENTA + "Altering table permissions, making a new Admin role." + Colour.RESET)
7         self.roles = ['audiadmin', 'audiuser']
8         self.roResult = self.ssh.run(command="sudo -u postgres psql -c \"SELECT rolname FROM pg_catalog.pg_roles;\"", hide=True, pty=True, watchers=[self.sPass])
9         self.roResult = self.roResult.stdout.split("\n")[2:-3]
10        self.extNames = [role.strip() for role in self.roResult if role.strip() in self.roles]
11        print(self.extNames)
12        for i in self.extNames:
13            self.ssh.run(command=f"sudo -u postgres psql -c \"DROP ROLE {i};\"", pty=True, watchers=[self.sPass])
14        self.ssh.run(
15            command=f"sudo -u postgres psql -c \"CREATE ROLE {self.roles[0]} WITH SUPERUSER LOGIN PASSWORD '{self.userDetails['PW']}'\"",
16            pty=True, watchers=[self.sPass])
17        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"INSERT INTO UCreds (role) VALUES ('{self.roles[0]}')\"", pty=True, watchers=[self.sPass])
18
19        print(Colour.MAGENTA + f"Adding the 'postgres' user to the {self.roles[0]} role." + Colour.RESET)
20        self.ssh.run(command=f"sudo -u postgres psql -c \"GRANT {self.roles[0]} TO postgres;\"", pty=True,
21            watchers=[self.sPass])
22
23        print(Colour.MAGENTA + f"Granting universal permissions to the {self.roles[0]} role." + Colour.RESET)
24        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"GRANT ALL PRIVILEGES ON \"cryReceipt\" TO {self.roles[0]};\"", pty=True,
25            watchers=[self.sPass])
26        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"GRANT ALL PRIVILEGES ON \"uKeys\" TO {self.roles[0]};\"", pty=True,
27            watchers=[self.sPass])
28        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"GRANT ALL PRIVILEGES ON \"uCredits\" TO {self.roles[0]};\"", pty=True,
29            watchers=[self.sPass])
30
31        print(Colour.MAGENTA + f"Creating a new {self.roles[1]} role." + Colour.RESET)
32        self.ssh.run(command=f"sudo -u postgres psql -c \"CREATE ROLE {self.roles[1]};\"", pty=True, watchers=[self.sPass])
33
34        print(Colour.MAGENTA + f"Revoking delete privileges from the {self.roles[1]} role." + Colour.RESET)
35        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"REVOKE DELETE ON \"cryReceipt\" FROM {self.roles[1]};\"", pty=True, watchers=[self.sPass])
36        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"REVOKE DELETE ON \"uKeys\" FROM {self.roles[1]};\"", pty=True, watchers=[self.sPass])
37        self.ssh.run(command=f"sudo -u postgres psql -d audicrypt -c \"REVOKE DELETE ON \"uCredits\" FROM {self.roles[1]};\"", pty=True, watchers=[self.sPass])

```

Figure 3.17: “adTableManip()”

From Figure 3.17 – one can tell that a lot of SQL commands are being repeated, though the role of this method is to establish a set of new roles, these being “audiadmin”, which is granted “superuser” status, and ‘audiuser’ – a role with a limited amount of privileges. A user with the ‘audiuser’ role is

entitled to write anything to 'cryReceipt' – a table which stores all receipts, and anything to "uKeys", and "uCreds"; albeit such 'read' privileges are more often than not – out of bounds.

## 4 Evaluation

### 4.1 Evaluation Methodology

#### 4.1.1 Evaluation Metrics

In order to evaluate the success of my program – I've segregated multiple different tests, on different regions of my program into a single test table. The 'actual' results will be compared to 'theorised' ones, to evaluate the efficacy, and success, of my Implementation.

### 4.2 Results

ID	Test Name	Theoretical Result	Actual Result	Is this what I expected?
1	Login in with Admin Credentials	Should login after Private Key verification	Does log in after verification	Yes
2	Logging in with User Credentials	Should login after Private Key verification	Does log in after verification	Yes
3	Making a new Admin Account	Takes all data, creates new 'audicrypt' database, and inserts all data into specific tables.	Does just that, all data can be received with SSH, SQL commands	Yes
4	Making a new Client Account	Takes all data, inserts into new record on uKeys, uCreds.	Does just that, returns to login screen after.	Yes
5	Trying to make a new account when one already exists	Should print an error message, return to menu	Prints "An Admin role for an AudiCrypt configuration already exists, please log in to this account to modify AudiCrypt."	Yes
6	Getting work directory as Admin	If directory is entered, it should be saved and returned to user.	Prints out entire directory relative path, 'while' loop until user responds 'y'/n'	Yes
7	Getting working directory that doesn't exist	Should ask user to "insert a valid directory path"	Does that after inserting non-existent directory path	Yes
8	Establishing connection on port 8888	Should respond "listening on {localhost}"	Responds with "Listening on 127.0.0.1".	Yes
9	Ask for working directory from Admin	Returns working directory as a String	Worked perfectly well, issue with 'SIGINT' error	Yes(?)

			handle catching though	
10	Attempt to download a file from server	File downloaded, moved to .audiTemp folder,	Worked well, but only tested with macOS machines	Yes(?)
11	Attempt to make a new Client account if a database doesn't exist.	Should print a message claiming to have an issue reaching the database.	Does just that, "There is an error with the AudiCrypt database. Ask your Admin to restart AudiCrypt configuration."	Yes
12	Wrong SSH values inputted	Should provide a record of the Exception, and return back to value input	Returns a 'Paramiko' "SSH Exception" code hasn't been handled correctly.	No
13	Input false option in either AdminUI() or ClientUI() choice	Return a statement telling the user it's a wrong option	Does exactly that	Yes
14	Get a live log of receipts being passed	Does repeatedly update at set intervals, and show new records	Does update with new inputs	Yes

### 4.3 Discussion

All in all, I feel that these tests reflect the efficacy of my software in a well-rounded manner; to which a robust design has helped ensure data is passed through safely, and securely. An issue I had during this stage was catching unexpected system interrupts, and ensuring all "asyncio" processes were also killed in a safe manner – features like "SIGINT" can't be caught as coroutines. Although my receipts were also being passed through to a live log, I did encounter a formatting issue with my Tkinter GUI during the latter stages of its development – an issue that I have since yet to rectify entirely. Fundamentally, though – a lot of practices I have implemented here have been as a result of theories, and concepts devolved from my Literature Review; particularly those regarding asymmetric transactions with other parties.

## 5 Conclusions

Finally, I believe that although this Project has had a few difficulties in terms of time constraints – I feel that AudiCrypt is a very promising, straightforward, yet secure, software that provides the building blocks for a framework that could be further developed with hypothetical maintenance. It stood up very well to several tests, and is constructed robust enough to efficiently react to multiple scenarios that could seriously affect the integrity of a server's configuration, a filesystem, and the Local Area Network it's connected to.

Although features, such as a direct implementation of data policy aren't available as of yet – this program is more than able to support it – I feel AudiCrypt also does a very good job at communicating with a user via a terminal user interface in a method that feels coherent to any user, regardless of whether they've had a limited experience within Cryptography, or computers, in general.

## 6 Recommendations for future work

This project has been an immensely entertaining experience for myself, and I feel that, although there a few paths I took that I feel might be a mistake in hindsight – it has helped me become a far better programmer than I was a few months ago. I feel that due to certain arrangements, such as employment, I haven't had the luxury of as much time as I felt this project deserved.

Nonetheless, the software I have built serves, in my opinion, as an ideal foundation for further maintenance in the future. With more sets of fresh eyes, thoughts, and implementations – this software, I believe, has a few features that could make the hallmarks of an exceptional software – especially for enterprise-based environments.

There have been a few moments throughout the duration of this Project where I have contemplated whether a programming language, like Python, a high-level language locked out of multithreading, efficient multi-core support, and intricate memory management – was a right choice for a Project of this nature. Quite frankly though, a majority of the libraries I found were more than capable of executing the sort of functionality I had envisaged – and it would take a lot more of the strenuous sort to really trouble this program into the idea of multi-core support.

In light of this, I do feel that this Project has motivated me further into researching new programming languages, and how they can integrate into Python – and whether they serve a better purpose for a given task than Python – I feel it'd make me a far greater programmer, and computer scientist, as a result.

## 7 References

www.parallels.com. (2021). Parallels: Mac & Windows Virtualization, Remote Application Server, Mac Management Solutions. [online] Available at: <https://www.parallels.com>.

docs.clamav.net. (n.d.). Introduction - ClamAV Documentation. [online] Available at: <https://docs.clamav.net> [Accessed 25 Apr. 2024].

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

Python documentation. (n.d.). What's New in Python 2.5. [online] Available at: <https://docs.python.org/3/whatsnew/2.5.html> [Accessed 25 Apr. 2024].

docs.python.org. (n.d.). hashlib — Secure hashes and message digests — Python 3.8.4rc1 documentation. [online] Available at: <https://docs.python.org/3/library/hashlib.html>.

pycryptodome.readthedocs.io. (n.d.). Crypto.Hash package — PyCryptodome 3.9.9 documentation. [online] Available at: <https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html>.

www.pycryptodome.org. (n.d.). Features — PyCryptodome 3.18.0 documentation. [online] Available at: <https://www.pycryptodome.org/src/features>.

Rodola', G. (n.d.). pyftplib: Very fast asynchronous FTP server library. [online] PyPI. Available at: <https://pypi.org/project/pyftplib/>.

Crisostomo, E.M. (2024). emcrisostomo/fswatch. [online] GitHub. Available at: <https://github.com/emcrisostomo/fswatch/> [Accessed 26 Apr. 2024].

NIST (2017). Hash Functions | CSRC. [online] Nist.gov. Available at: <https://csrc.nist.gov/projects/hash-functions>.

unblock.federalregister.gov. (n.d.). Federal Register :: Request Access. [online] Available at: <https://www.federalregister.gov/documents/2015/08/05/2015-19181/announcing-approval-of-federal-information-processing-standard-fips-202-sha-3-standard#:~:text=FIPS%20202%20and%20FIPS%20180> [Accessed 26 Apr. 2024].

Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce (2015). NIST Policy on Hash Functions - Hash Functions | CSRC. [online] Nist.gov. Available at: <https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions>.

Sectigo® Official. (n.d.). What is SHA Encryption? SHA-1 vs SHA-2. [online] Available at: <https://www.sectigo.com/resource-library/what-is-sha-encryption#:~:text=The%20primary%20difference%20between%20SHA>.

Dworkin, Morris J. (2015). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions".

pycryptodome.readthedocs.io. (n.d.). SHA-256 — PyCryptodome 3.17.0 documentation. [online] Available at: <https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html>.

National Institute of Standards, and Technology (2023) Digital Signature Standard. (Accessed: 26 Apr 2024).

Anon, (n.d.). Comparing ECDSA vs RSA - SSL.com. [online] Available at: <https://www.ssl.com/article/comparing-ecdsa-vs-rsa/>.

www.winzip.com. (n.d.). WinZip | Download Your Free Trial. [online] Available at: <https://www.winzip.com/en/learn/tips/ftp/port/?alid=885575657.1714307492#what-port> [Accessed 28 Apr. 2024].

Arubanetworks.com. (2023). TCP/UDP port number ranges. [online] Available at: <https://www.arubanetworks.com/techdocs/AOS-S/16.10/ATMG/YC/content/common%20files/tcp-port-num-ran.htm>.

Annadanam, N (n.d.). Working With Postgres WAL Made Easy 101 | Hevo. [online] Available at: <https://hevodata.com/learn/working-with-postgres-wal/> [Accessed 29 Apr. 2024].

www.paramiko.org. (n.d.). Welcome to Paramiko! — Paramiko documentation. [online] Available at: <https://www.paramiko.org>.

Cloudflare (2023). What is SSL (Secure Sockets Layer)? | Cloudflare UK. Cloudflare. [online] Available at: <https://www.cloudflare.com/en-gb/learning/ssl/what-is-ssl/>.

GOV (2018). Data Protection Act. [online] Gov.UK. Available at: <https://www.gov.uk/data-protection>.

GDPR (2013). General Data Protection Regulation (GDPR) . [online] General Data Protection Regulation (GDPR). Available at: <https://gdpr-info.eu>.

Wolff, J. and Atallah, N., 2021. Early GDPR penalties: Analysis of implementation and fines through May 2020. Journal of Information Policy, 11, pp.63-103.

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 Pricing. [online] Available at: <https://aws.amazon.com/qldb/pricing/>.

Python (2019). os — Miscellaneous operating system interfaces — Python 3.8.0 documentation. [online] Python.org. Available at: <https://docs.python.org/3/library/os.html>.

PyPI. (n.d.). tqdm: Fast, Extensible Progress Meter. [online] Available at: <https://pypi.org/project/tqdm/>.

docs.python.org. (n.d.). signal — Set handlers for asynchronous events — Python 3.8.2 documentation. [online] Available at: <https://docs.python.org/3/library/signal.html>.

www.w3schools.com. (n.d.). Python Self. [online] Available at:  
[https://www.w3schools.com/python/gloss\\_python\\_self.asp](https://www.w3schools.com/python/gloss_python_self.asp).

pycryptodome.readthedocs.io. (n.d.). SHA3-256 — PyCryptodome 3.210b0 documentation. [online] Available at: [https://pycryptodome.readthedocs.io/en/latest/src/hash/sha3\\_256.html](https://pycryptodome.readthedocs.io/en/latest/src/hash/sha3_256.html) [Accessed 13 May 2024].

www.pyinvoke.org. (n.d.). Welcome to Invoke! — Invoke documentation. [online] Available at: <https://www.pyinvoke.org> [Accessed 13 May 2024].



## 8 Bibliography

- Ubuntu. (n.d.). Ubuntu for ARM | Download. [online] Available at: <https://ubuntu.com/download/server/arm>.
- Amazon Web Services, Inc. (n.d.). Blockchain on AWS - Amazon Web Services. [online] Available at: <https://aws.amazon.com/sblockchain/>.
- Amazon Web Services, Inc. (n.d.). Amazon QLDB Features. [online] Available at: <https://aws.amazon.com/qldsdb/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.). Trillian 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](https://chat.openai.com). Available at: <https://chat.openai.com>.
- 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/pyftplib. [online] GitHub. Available at: <https://github.com/giampaolo/pyftplib> [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>.

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].

Gilbert, H., Handschuh, H. (2004). Security Analysis of SHA-256 and Sisters. In: Matsui, M., Zuccherato, R.J. (eds) Selected Areas in Cryptography. SAC 2003. Lecture Notes in Computer Science, vol 3006. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-24654-1\\_13](https://doi.org/10.1007/978-3-540-24654-1_13)

docs.twistedmatrix.com. (n.d.). twisted. [online] Available at: <https://docs.twistedmatrix.com/en/stable/api/index.html> [Accessed 28 Apr. 2024].

GeeksforGeeks. (2017). Python program to find IP Address. [online] Available at: <https://www.geeksforgeeks.org/python-program-find-ip-address/>.

