AudiCrypt: A Cryptographically-Verifiable Receipt Generator

Matthew Potts

Integrated Master of Computer Science (MSci)

Birmingham City University

An Overview of the Project.

AudiCrypt, a cryptographically-verifiable receipt generator, integrates maximized security, and low overhead into single discrete application. Based entirely off a colour-coordinated terminal user interface, for improved visibility, and built entirely with Python – AudiCrypt is designed ideally for enterprise-based networks – where accountability, and non-repudiation is essential.

In modern-day enterprise, data protection is a highly significant aspect of individual organisations – it's answerable to both domestic, and international legislation, including the Data Protection Act 2018, and GDPR. As such, user activity within local Enterprise networks must be tracked, in an immutable manner, to ensure all data modification is recorded – this can be used as crucial evidence in case a disciplinary investigation is launched.

AudiCrypt ensures a network, and an enterprise, is protected against malicious users – by maintaining discipline, enterprise compliance can be achieved. AudiCrypt acts, fundamentally, as a deterrent in this regard.

Methodology.

Right from the off, I decided on implementing a Spiral-like approach into my project; stemming off a basic framework – with a terminal interface – which would then progress into a complete Graphical User Interface if time constraints permitted. I wanted to ensure that mechanisms of both asymmetric, and symmetric, cryptography – were fully in place, and operating before anything, like a GUI, were in the mix.

The majority of my project, including my design, and theory, have all been documented on; I constructed my software with PyCharm, by Jetbrains. AudiCrypt is split into 6 '.py' files, all of which operate with one another. Although this does work, due to a hard-coded limit in terminals across all major operating systems – an input asking for a user's Private Key can't be fulfilled (this issue will be discussed during the Demonstration).

All modern computers can handle predominant hashing, and encryption algorithms, including SHA-256, and AES256; and although built in Python, particular functions (e.g. those in 'pycryptodomex') are produced in C; therefore execution time is not too degraded at all.

I completed GUI, Class, and Database Designs on both "draw.io", and "Figma". Copies of each can all be found in the Appendices of my Report.

Summary of Findings & Other Results.

The *lionshare* of my software works as expected, and throughout development – I ensured to develop my software as defensively as possible. In this context, that means using vast 'while', and 'try, except' statements to ensure user inputs are of a desired format before they're passed to any further logic. I made use of Python's 'signal' library to clear all variables from memory in case of a random exit by the user – for security's sake. This also ensured that any networking instances instantiated by the 'asyncio' library could be killed in a safe, organised manner.

Throughout my development, I've been thoroughly impressed by both 'pycryptodomex', and 'Fabric' – the latter a further development on top of Python's infamous SSH library – 'paramiko' – which I had a lot of trouble utilizing – as the majority of my commands require 'sudo' elevation privileges. The 'pycryptodomex' library encapsulates numerous different cryptographic algorithms – I was surprised as to how straight forward it is to manipulate data with this library – and how little trouble it causes computationally during runtime.

Although there is a formatting error with the live receipt log feature on the Admin's dashboard, I believe this can be easily rectified – apart from that, it updates perfectly with the database – and shows all receipts in a concise manner.

Recommendations for Future Work.

I like to believe that my software will act as a framework for future maintenance, and updates. AudiCrypt doesn't have a limit to how much it can be expanded on – theoretically it could support a wide-ranging network given SSH access is maintained. PostGreSQL is hallmarked as a DBMS with massive scalability, and AudiCrypt can utilize this, in tandem, to massively promote the opportunities associated with this software.

There have been a few stages during this software's development, where I have considered whether a high-level programming language, like Python, was the right choice. However, a majority of all third-party libraries work perfectly well, and Python can be compiled, and distributed, with third-party tools - like Nuitka.

AudiCrypt, hypothetically, will also receive a Graphical User Interface at some stage. I wanted to ensure that this software has a Console output at least. I feel a GUI frontend could be better implemented with other programming languages thought – e.g. C++, or Java. The existing Python code would be used as a backend.

Depending on how advanced AudiCrypt could hypothetically get, libraries (e.g. Numba) – do exist within Python, that provide multithreading, multi-core support – I doubt whether this stage would ever be reached though.

Demonstration.

text