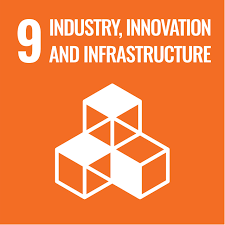
A logo of a globe with yellow rings around it

AI-generated content may be incorrect.





**FINAL YEAR PROJECT (FYP)  
INVESTIGATION REPORT (IR)**

**Password Extractor and Decryptor from Browser-Based Folders in Forensics Images**

**ABDULRAHMAN GAMIL MOHAMMED AHMED**

**TP071012**

**APD3F2411CS(CYB)**

A report submitted in partial fulfilment of the requirements for the degree of

Technology and Innovation for the degree of

B.Sc. (Hons) Information Technology specializing in Cybersecurity

at Asia Pacific University of Technology and Innovation.

**Supervised by MOHD HANIS BIN JENALIS**

**2nd Marker: TS. MUHAMMAD AMIN BIN SAHARI**

Contents

[ACKNOWLEDGEME 6](#_Toc192695410)

[ABSTRACT 8](#_Toc192695411)

[CHAPTER 1: INTRODUCTION 10](#_Toc192695412)

[1.1 Introduction 11](#_Toc192695413)

[1.2 Problem Background 13](#_Toc192695414)

[1.3 Project Aim 16](#_Toc192695415)

[1.4 Objectives 17](#_Toc192695416)

[1.5 Scope 19](#_Toc192695417)

[1.6 Potential Benefits 21](#_Toc192695418)

[1.7 Overview of the IR 23](#_Toc192695419)

[1.8 Project Plan 24](#_Toc192695420)

[CHAPTER 2: LITERATURE REVIEW 25](#_Toc192695421)

[2.1 Domain Research 26](#_Toc192695422)

[2.1.1 Study of Domain from General to Specific 28](#_Toc192695423)

[2.1.2 Study of Related Theories/Technologies/Algorithms 36](#_Toc192695424)

[2.1.3 Review of Sub-Topics within the Domain 48](#_Toc192695425)

[2.2 Similar Systems/Works 53](#_Toc192695426)

[2.2.1 Similar Systems 53](#_Toc192695427)

[2.2.2 Strengths and Weaknesses 59](#_Toc192695428)

[2.2.3 Conclusion 61](#_Toc192695429)

[2.3 Technical Research 62](#_Toc192695430)

[2.3.1 Overview of Python as a Development Platform for Forensic Tools 62](#_Toc192695431)

[2.3.2 Utilization of Pycryptodome Library for Encryption and Decryption 65](#_Toc192695432)

[2.3.3 Handling SQLite Databases in Browser Password Extraction 67](#_Toc192695433)

[2.3.4 Parsing JSON Structures for Firefox Password Recovery 69](#_Toc192695434)

[2.3.5 Interfacing with Operating System Secure Storage Mechanisms 71](#_Toc192695435)

[2.3.6 Command-Line Interface (CLI) Design and Usability Considerations 73](#_Toc192695436)

[2.3.7 Challenges in Processing Forensic Image Exports 75](#_Toc192695437)

[2.3.8 Performance Optimization Techniques for Large-Scale Data Handling 77](#_Toc192695438)

[CHAPTER 3: METHODOLOGY 79](#_Toc192695439)

[3.1 System Development Methodology 80](#_Toc192695440)

[3.1.1 Introduction 80](#_Toc192695441)

[3.1.2 Methodology Choice and Justification 85](#_Toc192695442)

[3.1.3 Describe the Activities and Process in Each Phase Towards the Chosen Methodology 88](#_Toc192695443)

[3.2 Data Gathering Design 93](#_Toc192695444)

[3.3 Analysis 127](#_Toc192695445)

[3.3.1 Survey Analysis 127](#_Toc192695446)

[3.3.2 Interviews Analysis 130](#_Toc192695447)

[CHAPTER 4: CONCLUSION 135](#_Toc192695448)

[REFERENCES 138](#_Toc192695449)

[APPENDICES 145](#_Toc192695450)

[Appendix A 146](#_Toc192695451)

[Appendix B 151](#_Toc192695452)

[Appendix C 155](#_Toc192695453)

[Appendix D 159](#_Toc192695454)

[Appendix E 160](#_Toc192695455)

Figures

[1: Project Plan 24](#_Toc192695456)

[2 ‎Google Chrome & Microsoft Edge‎ 30](#_Toc192695457)

[3 Mozilla Firefox 31](#_Toc192695458)

[4 Opera ‎ 32](#_Toc192695459)

[5 Brave 32](#_Toc192695460)

[6 AES-256 ‎Design 36](#_Toc192695461)

[7 Design of 3DES 38](#_Toc192695462)

[8 How Post-Quantum Cryptography Works 50](#_Toc192695463)

[9 Biometric Authentication Types 51](#_Toc192695464)

[10 Zero-Knowledge Proofs Illustration 52](#_Toc192695465)

[11 Firefox Decrypt by "unode" Repo 54](#_Toc192695466)

[12 Firefox Decrypt Tool Result 55](#_Toc192695467)

[13 Decrypt Chrome Passwords by "ohyicong" Repo 57](#_Toc192695468)

[14 Decrypt Chrome Tool Result 58](#_Toc192695469)

[15 Firefox 59](#_Toc192695470)

[16 Google Chrome 60](#_Toc192695471)

[17 Why Python? 62](#_Toc192695472)

[18 Pycryptodome ‎ Library Used 65](#_Toc192695473)

[19 SQLite Databases 67](#_Toc192695474)

[20 .Json files 69](#_Toc192695475)

[21 Windows OS 71](#_Toc192695476)

[22 Kali Linux OS 72](#_Toc192695477)

[23 Decrypting browser-stored credentials requires advanced technical expertise.‎ 95](#_Toc192695478)

[24 Extracting browser-stored passwords from forensic disk images is a time-consuming process. 96](#_Toc192695479)

[25 The complexity of encryption methods used by brewers poses a significant challenge. 97](#_Toc192695480)

[26 Forensic investigators face difficulties in handling multiple browser types and versions. ‎ 98](#_Toc192695481)

[27 The lack of standardized tools for password extraction complicates forensic ‎investigations.‎ 99](#_Toc192695482)

[28 Frequent browser updates often render existing password recovery tools ineffective. 100](#_Toc192695483)

[29 The current process of recovering browser-stored credentials is prone to errors.‎ 101](#_Toc192695484)

[30 Forensic investigators require extensive training to use password-recovery tools ‎effectively. 102](#_Toc192695485)

[31 Capture the Flag (CTF) players often encounter challenges when extracting and ‎decrypting ‎browser-stored credentials in forensic challenges.‎ 103](#_Toc192695486)

[32 Existing forensic tools are effective in recovering passwords from browser-based folders.‎ 104](#_Toc192695487)

[33 A specialized tool for password extraction and decryption would improve forensic ‎efficiency.‎ 105](#_Toc192695488)

[34 The integration of a specialized tool into existing forensic workflows would be seamless.‎ 106](#_Toc192695489)

[35 Current tools cannot handle large volumes of data efficiently.‎ 107](#_Toc192695490)

[36 Forensic tools often fail to decrypt credentials from browsers with advanced security ‎features.‎ 108](#_Toc192695491)

[37 Streamlining password recovery processes would significantly benefit digital forensic ‎‎investigations. ‎ 109](#_Toc192695492)

[38 Existing forensic tools are user-friendly and easy to operate. ‎ 110](#_Toc192695493)

[39 Forensic tools are capable of handling encrypted data from all major browsers. ‎ 111](#_Toc192695494)

[40 The speed of existing tools is sufficient for most forensic investigations. ‎ 112](#_Toc192695495)

[41 Investigators are satisfied with the performance of current password recovery tools.‎ 113](#_Toc192695496)

[42 Existing tools provide comprehensive documentation and support for users.‎ 114](#_Toc192695497)

[43 Forensic tools are compatible with a wide range of operating systems and platforms.‎ 115](#_Toc192695498)

[44 The cost of forensic tools is justified by their performance and features. ‎ 116](#_Toc192695499)

[45 Current tools are effective in recovering credentials from both active and deleted browser data.‎ 117](#_Toc192695500)

[46 The accuracy of forensic tools decreases when dealing with corrupted or incomplete ‎data. ‎ 118](#_Toc192695501)

[47 Forensic tools are effective in recovering credentials from browsers with multi-factor ‎‎authentication.‎ 119](#_Toc192695502)

[48 Title and Introduction in the PPF 146](#_Toc192695503)

[49 Problem Statements in the PPF 147](#_Toc192695504)

[50 Aims and Objectives in the PPF 148](#_Toc192695505)

[51 Target Users in the PPF 149](#_Toc192695506)

[52: Submission in FYPbank 149](#_Toc192695507)

[53 Submission in FYPbank 150](#_Toc192695508)

[54 Ethics Form (Fast Track) Particioant Confidentiality 151](#_Toc192695509)

[55 Ethics Form (Fast Track) ‎Nature of Research 152](#_Toc192695510)

[56 Ethics Form (Fast Track) ‎ Nature of Research 152](#_Toc192695511)

[57 Ethics Form (Fast Track) target Participants 153](#_Toc192695512)

[58 Ethics Form (Fast Track) ‎Support Information 154](#_Toc192695513)

[59 Proposed Meeting Schedule 155](#_Toc192695514)

[60 Meeting Logs 155](#_Toc192695515)

[61: 1st Meeting 156](#_Toc192695516)

[62: 2nd Meeting 157](#_Toc192695517)

[63: 3rd Meeting 158](#_Toc192695518)

[64 Gantt Chart 159](#_Toc192695519)

Tables

[‎ Table 1 Objectives 17](#_Toc192695520)

[Table 2 List of Interviewees 120](#_Toc192695521)

[Table 3 Key Challenges in Password Extraction 129](#_Toc192695522)

[Table 4 Ineffectiveness of Current Tools 129](#_Toc192695523)

[Table 5 Support for a Specialized Tool 129](#_Toc192695524)

[Table 6 Summary Table 1 133](#_Toc192695525)

[Table 7 Summary Table 2 133](#_Toc192695526)

[Table 8 Summary Table 3 134](#_Toc192695527)

[Table 9 Summary Table 4 134](#_Toc192695528)

[Table 10: Respondence Demographic 161](#_Toc192695529)

# ACKNOWLEDGEME

Above all, I would like to offer my heartfelt gratitude and sincere thanks to my project supervisor, **Mohd Hanis Bin Jenalis**, for his dedication and support in this project. He was always available for discussion with me, giving me thoughtful ideas and constructive criticisms to enhance the work I've done. Without his guidelines, I would have faced a daunting task trying to complete this project.

I also would like to express my sincere appreciation to **Ts. Muhammad Amin Bin Sahari**, my second marker, whose constructive comments and experience have significantly contributed to the quality of this investigation report. His input has been immeasurable in refining my endeavor.

Additionally, I would also like to thank project manager of all Final Year Projects, for his thorough briefings and excellent reference materials. These have been very useful in completing the chapters of this report and its finalization.

I also have a debt of gratitude to all the people who took part in the surveys and interviews conducted for data collection in this project. Your responses and ideas are greatly appreciated and highly valued.

Last but not least, I would like to thank my friends and family for their encouragement and support throughout this project. They have been a tremendous source of strength and comfort, allowing me to deal with the enormous workload. Thanks to their motivation, I was able to persevere and finally complete this project.

# ABSTRACT

This project creates a forensic tool, particularly in the digital forensics area, for automatically decrypting and recovering passwords stored in browser directories, as a remedy to the primary limitation in cybercrime investigations and Capture the Flag (CTF) competitions. This project proposes a solution to the encryption of browsers (Opera, Firefox, MS Edge, Google Chrome, and Brave) that makes the recovery of passwords impossible because manual operations to locate and decrypt data in forensic disk images are time-consuming and unpredictable. The process utilizes the development of a Python program based on **pycryptodome** libraries to decrypt, using which users hand-export browser folders from forensic snapshots and execute the tool against folders to reveal passwords and metadata (usernames, URLs, dates). Initial output indicates that the tool decrypts credentials efficiently with organized evidence presentation, boosting forensic precision. The significance of this project comes with enhanced investigation efficacy, assistance of cybersecurity experts, and the facilitation of a stronger case to legal outcomes. It aligns with SDG 9 (Industry, Innovation, and Infrastructure) by encouraging technology-based solutions for security.

**Keywords:** *digital forensics, password decryption, browsers security, forensic image, Capture the flag challenges.*

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

Web browsers are crucial programs in the information age, offering a mechanism for people to surf Internet-based resources conveniently while keeping sensitive information, such as usernames, passwords, and history, in storage to provide convenience. The caching of the items, however, raises a significant security issue, precisely for cybercrime and digital forensics. Forensic analysts usually need to recover such credentials from forensic disk images such as ‎dd ‎or.raw image files to recreate user activities, generate timelines, or prove intent in court ‎cases. The problem is that browsers store passwords securely with strong encryption algorithms ‎‎(for example, AES-256 in Google Chrome, NSS in Mozilla Firefox, and the equivalent mechanisms in Opera, ‎Edge, and Brave) to secure the users' data, and extracting and decrypting them is a tricky, time-‎consuming task without special tools.

The current state of the art in digital forensics proves to have a basic lack of tool functionality. The majority of the existing forensic software tools, i.e., Autopsy, FTK, EnCase, or XRY, are general-purpose data recovery tools that lack specialized, efficient functions for locating browser-specific folders, decrypting their encryption, and extracting related metadata (i.e., URLs, timestamps, usernames) for Opera, Chrome, Firefox, Edge, and Brave. For instance, Chrome stores passwords in its "Login Data" SQLite database in ‎the %LocalAppData%\Google\Chrome\User Data\Default directory on Windows, encrypted ‎with AES-256 and tied to the operating system-level encryption key of the user (e.g., DPAPI). ‎Firefox, on the other hand, uses "logins.json" and "key4.db" files with NSS cryptography, while ‎Opera, Edge, and Brave use comparable encryption methods (e.g., Chromium-based encryption ‎for Opera and Brave, Windows DPAPI for Edge). These intricacies force examiners to manually navigate enormous disk images, find browser folders, and utilize third-party tools or custom scripts for decryption, a time-consuming, error-prone, and tiresome process.

‎

This inefficiency extends to Capture the Flag (CTF) competitions, in which players work under time pressure to extract passwords from forensic images as part of challenges. CTF environments simulate real-world forensic work, but the lack of integrated, user-friendly tools subtracts from performance, especially for newer competitors. As an example, a CTF challenge may be a 10GB forensic image and any ­hidden ‎browser password from Opera, Chrome, Firefox, Edge, or Brave to be recovered within ‎‎30 minutes when manual methods normally can't meet such deadlines, leading to missed ‎opportunities and increased stress.

This ***Password Extractor and Decryptor from Browser-Based Folders in Forensic Images*** projects aims to bridge these gaps by developing an advanced CLI forensic tool. The tool allows for automated password extraction and decryption from browser folders extracted from forensic images for Opera, Chrome, Firefox, Edge, and Brave and presents results in a readily comprehensible text-based format accessible over the command line for forensic examiners, CTF players, and other cybersecurity professionals. By automating this process through a CLI, the tool enhances investigative efficiency, improves consistency, and supports competitive performance in CTF contexts. This chapter introduces the background of the project, highlights its relevance in digital forensics and cybersecurity, and provides the background for presenting its problems, objectives, goals, scope, benefits, and plan of implementation. It stresses the need to deliver innovative CLI solutions in a time when cyber-attacks are becoming increasingly sophisticated, and forensic evidence plays a pivotal role in security and justice.

## 1.2 Problem Background

Their application of web browsers as repositories for confidential user data, such as passwords, has made them useful commodities for users and prime targets for cyber attackers, as well as a necessary source of information for forensic examiners. However, the encryption methods used by browsers to secure this information in an attempt to maximize the privacy of the user pose significant challenges in a forensic environment. Current forensic password recovery tools from browser-stored data in forensic disk images are not performing optimally since they cannot handle the intricacies of new-generation browser encryption efficiently for Opera, Chrome, Firefox, Edge, and Brave. For example, Google Chrome uses AES-256 encryption for its "Login Data" SQLite database, which requires the user's operating system-level encryption key (e.g., Windows DPAPI) or a master password, whereas Mozilla Firefox uses NSS (Network Security Services) cryptography for its "logins.json" file, which requires decryption keys stored in "key4.db." Opera, Edge, and Brave, being Chromium-based or Windows-integrated browsers, use similar encryption (e.g., Chromium's AES-256, Windows DPAPI for Edge), with increased complexity due to variations in storage structures and key management.

There are other forensic tools available, such as Autopsy, FTK Imager, or XRY, with general functionality to process disk images but lack browser password extraction specificity and automation in Opera, Chrome, Firefox, Edge, and Brave from a CLI. The aforementioned tools predominantly have investigators performing much work in attempting to locate the directories for the browser

*%LocalAppData%\Google\Chrome\User Data\Default\Login Data for Chrome on Windows*

*%AppData%\Mozilla\Firefox\Profiles\<profile>\logins.json for Firefox on Windows*

*%LocalAppData%\Opera Software\Opera Stable\Login Data for Opera on Windows*

*%LocalAppData%\Microsoft\Edge\User Data\Default\Login Data for Edge, on Windows*

*%LocalAppData%\BraveSoftware\Brave-Browser\User Data\Default\Login Data for Brave on Windows*

Pulling the relevant databases and decrypting via custom scripts or third-party CLI utilities (e.g., ChromePass, Firefox Decrypt). Such a CLI-based manual approach not only incurs laborious effort but also contains a degree of human error, given that it demands exhaustive technical expertise with browser storage hierarchies, encryption methods, and forensic image formats (.dd,.raw). For instance, a 100GB disk image examiner could easily spend 2–3 hours bouncing around to locate a browser directory for one of these browsers, to be greeted with incompatible decryption keys, bad data, or incorrect metadata, and thus end up with incomplete or deceptive evidence.

The challenge is compounded in Capture the Flag (CTF) tournaments, where players are required to ‎read passwords from forensic images within time constraints to overcome challenges and earn points. CTF challenges generally simulate real forensic work, but the lack of integrated, easy-to-use CLI tools forces participants to employ faulty solutions, such as manual SQLite database queries, single CLI decryption scripts, or sets of multiple CLI tools, for Opera, Chrome, Firefox, Edge, and Brave. This inefficiency can lead to missed opportunities, increased stress, and lower performance, particularly for less experienced participants. For example, a 10GB forensic image CTF challenge might be to find a hidden browser password within 30 minutes in any of these browsers, but through manual CLI operations, including folder detection, database recovery, and decryption, the participants will fail or do poorly.

The absence of integrated CLI solutions also contributes to these challenges. General-purpose CLI tools are too broad in nature and lack the specificity needed for browser-specific CLI tasks among Opera, Chrome, Firefox, Edge, and Brave, and domain-specific CLI tools may be interoperability-less or feature-complete. Analysts and CTF participants need to switch between multiple CLI package software, which have differing command syntax and output forms, thus exposing them to a high risk of errors and workflow inefficiencies. For instance, an examiner can use FTK Imager's CLI to review a disk image, a hand-crafted Python script to download browser information, and a third CLI tool to decrypt passwords in the browsers and get inconsistent information and incomplete results, and it takes ages to solve investigations. Similarly, CTF participants can end up spending unnecessary time addressing CLI command compatibility issues and diverting attention away from cracking problems.

Furthermore, the rising adoption of electronic evidence in forensic settings makes the need for accurate and efficient password recovery from Opera, Chrome, Firefox, Edge, and Brave via a CLI even greater. Passwords play a pivotal role in creating access, intent, or timelines during cybercrime investigations, such as unauthorized access, data breaches, or identity theft. But without effective CLI ‎tools to pull and report this information accurately, investigators jeopardize case ‎results, potentially resulting in dropped charges, extended litigation, or compromised security ‎procedures. This is a significant problem in situations where forensic images are massive, ‎encrypted, or broken up, necessitating advanced CLI solutions to maintain accuracy and speed ‎for all these browsers.

The ineffectiveness of current forensic tools in password recovery for browsers stems from their inability to handle encryption, automate folder recognition, and integrate metadata extraction seamlessly for Opera, Chrome, Firefox, Edge, and Brave using a CLI. The ineffectiveness has implications for digital forensic analysis, CTF competitions, and cybersecurity tasks, calling for a specialist CLI tool to address these problems in forensic and competition environments.

## 1.3 Project Aim

To develop a Python CLI utility to automate password recovery and decryption from exported browser directories (Opera, Google Chrome, Firefox, MS Edge, Brave) in forensic images, improving efficiency for digital forensics and capture the flag (CTF) challenges with accurate text-based output for forensic examiners and security professionals.

## 1.4 Objectives

The objectives of the project are concise, measurable, and aligned with its mission of fixing the problems ‎identified in Opera, Chrome, Firefox, Edge, and Brave using a Python CLI. ‎These objectives render the development of the tool focused, testable, and impactful:

|  |  |  |  |
| --- | --- | --- | --- |
| 1st Objective | 2nd Objective | 3rd Objective | 4th Objective |
| **To develop** a Python CLI tool to extract/decrypt browser passwords from forensic images, tested for reliability in forensics/CTF. | **To combine** Python methods to decrypt passwords/metadata from browsers in forensic images, verified for accuracy. | **To present** browser data in simple, usable CLI output, validated by forensic feedback. | **To enable** CTF players to rapidly decrypt browser passwords from forensic images, reducing times via testing. |

*‎ Table 1 Objectives*

**1. To develop a Python CLI tool** to scan exported folders (Opera, Chrome, Firefox, Edge, Brave) and effectively extract and decrypt usernames and passwords from forensic images. This objective can be measured by testing the Python CLI tool using sample forensic images with browser data, measuring extraction and decryption efficiency through qualitative testing and user authentication. Effective functionality ensures reliability and addresses the need for effectiveness in digital forensics and CTF competitions.

**2. To combine different Python-based decryption techniques to decrypt passwords and associated metadata** (username, URL, timestamp) in imported Opera, Chrome, Firefox, Edge, and Brave directories of forensic images with high accuracy, as determined through testing. This target meets the main functionality of the Python CLI tool through diverse decryption methods for password recovery and metadata retrieval. Measurability is achieved by verifying retrieved data with established values in test sets such that high accuracy is guaranteed for high-quality evidence during forensic examination on these browsers.

**3. To deliver retrieved data for Opera, Chrome, Firefox, Edge, and Brave** in a simple, organized, text-based CLI presentation, achieving high usability as evaluated by target user feedback from forensic examiners. The objective of this is usability, ensuring that the output of the Python CLI tool (i.e., passwords, usernames, URLs, timestamps) is human-readable and easily accessible as command-line text. Usability will be quantified via surveys or test sessions with forensic experts with the aim of high satisfaction to render it viable for actual utilization.

**4. To enable CTF players to recover and decrypt browser passwords from exported Opera, Chrome, Firefox, Edge, and Brave folders in forensic images** via the Python CLI tool, significantly reducing challenge-solving time as approximated in test CTF environments. This objective is narrowly aimed at CTF players and measures time saved in test environments. By comparing tool-aided Python CLI performance with baseline performance (manual Python CLI methods), the project aims to dramatically cut down extraction times, enhancing competitiveness and efficiency when competing in CTF challenges for such browsers.

These objectives are designed to be achievable, plausible, and heavily tied to the objective of the project, ensuring quantifiable results addressing the identified problems in password recovery in the browsers of Opera, Chrome, Firefox, Edge, and Brave by using a Python CLI.

## 1.5 Scope

The project scope defines the extent, activities, and limitations of developing the ***Password Extractor and Decryptor from Browser-Based Folders in Forensic Images*** Python‎-based CLI tool to make it possible to have a straightforward and feasible means for Opera, Chrome, Firefox, Edge, ‎and Brave.

**Tasks to be Performed:** The project involves various significant tasks to achieve its objectives. ‎‎To begin with, an exhaustive literature review shall be conducted to find out about the browser password ‎storage habits for Opera, Chrome, Firefox, Edge, and Brave with encryption algorithms ‎(e.g., AES-256, NSS, base64) and the limitations of existing forensic Python CLI tools. This will inform the design of the Python CLI tool that will seek to develop a Python-driven command-line ‎application with an effective list of commands. The Python CLI tool will utilize libraries like ‎pycryptodome to decrypt and process data from browser folders exported to be utilized with these browsers. Users will manually export the folders out of forensic disk images ‎(.dd,.raw) employing existing forensic software (FTK Imager, Autopsy) and afterwards run Python ‎CLI commands (python Password\_extrctor.py –b {name of browser} -in {the folder extracted} -out {where user wants to export the result to csv file}). Testing will involve sample known browser forensic images to verify extraction precision and decryption success for Opera, Chrome, Firefox, Edge, and Brave. Output from the Python CLI tool will be presented in plain text or CSV, making it easy to read for forensic analysts and CTF players. Python CLI command manuals, technical reports, and documentation will be written to aid deployment and future updates.

**Constraints:** Several constraints shape the Python CLI tool’s development. The tool’s system ‎requirements will be limited to modest hardware (e.g., memory usage under 2GB, compatible ‎with Windows and Linux command-line environments) to ensure accessibility for target users. ‎The desktop browsers (such as Opera, Chrome, Firefox, Edge, Brave) are the only one scope, while mobile browsers (such as Safari on iOS, Chrome on Android) are outside the scope due to different storage formats and encryption methods. The Python CLI tool relies on users exporting browser folders manually because it does not natively parse forensic images, thereby restricting its utility to exported data for these browsers. Encryption support will be based on standard algorithms (e.g., AES-256, NSS). Time constraints of the FYP timeline (e.g., 4 - 5 months) and the need for a standalone Python CLI solution (not included in larger forensic suites) further constrain the scope of the project. Furthermore, the implementation of the Python CLI tool will bypass graphical interfaces or ‎complex GUI functions due to its command-line nature, and it will not accommodate ‎complex machine learning or artificial intelligence approaches due to resource limitations ‎and project scope for such specific browsers.

**What Will and Will Not Be Done:** The project will produce a stand-alone Python CLI password recovery and decryption forensic tool, targeting exported browser folders from forensic images for Opera, Chrome, Firefox, Edge, and Brave. It will support these big desktop browsers, utilize encryption using Python CLI commands, and provide text-based outputs, but without sophisticated image carving, network forensics, or direct disk image parsing. ‎Cloud-based password recovery, live monitoring, or mobile browser support will not be ‎addressed. The Python CLI tool will not be integrated into existing forensic platforms (e.g., ‎Autopsy, FTK) or support proprietary encryption beyond standard fare for these browsers ‎to keep the scope clean and manageable.‎

***This broad scope renders the Python CLI project sustainable, as well as aligned with its objectives, ‎and gives a viable solution to target users while acknowledging its limitations for Opera, ‎Chrome, Firefox, Edge, and Brave.‎***

## 1.6 Potential Benefits

The Browser-Based Folders ***Password Extractor and Decryptor in Forensic Images*** Python CLI tool possesses physical as well as non-physical benefits, accommodating various users ‎in digital forensics and cybersecurity for Opera, Chrome, Firefox, Edge, and Brave.

**Tangible Advantages:** The Python CLI tool will save forensic examiners significant time, possibly hours on each case, by automating password retrieval and decryption processes currently involving manual Python CLI effort in Opera, Chrome, Firefox, Edge, and Brave. For example, whereas one researcher used to study a 100GB disk image and he would be able to take 2–3 hours in finding and decrypting browser passwords manually by Python CLI commands, the tool can reduce it to 15–30 minutes on every folder depending upon size and complexity, by basic Python CLI inputs (python Password\_extrctor.py –b {name of browser} -in {the folder extracted} -out {where user would like to export the result in csv file}). For CTF players, the Python CLI tool's performance would reduce password recovery time by 50% in challenges with these browsers, allowing for quicker problem-solving and increased scores in competitive environments. Organizations will also realize cost savings since fewer man-hours will be required for internal cybersecurity audits or incident response, minimizing labor costs and enhancing resource utilization. In addition, the precision of the Python CLI tool will minimize errors to an absolute minimum and, in doing so, save legal costs on incomplete evidence in court cases against such browsers, such as avoiding re-investigations or expert witness statements.

**Benefits of intangibility:** The Python CLI tool will enhance the confidence of forensic examiners ‎in their evidence gathering, giving them precise password details for court use, access validation, and timeline creation from Opera, Chrome, Firefox, Edge, and Brave via the command line. For CTF players, it will enhance competitive performance and skill ‎development, advancing mastery in digital forensics and cybersecurity for these browsers ‎with Python CLI. Security professionals will better understand browser vulnerabilities on these browsers and be in a position to emulate attacks and improve system security through Python CLI processes. The Python CLI tool will also give corporate IT and security personnel confidence by ensuring consistent policy adherence and resource monitoring for these browsers, and ethical hackers will have a better time testing browser security more effectively through Python CLI commands. These intangible benefits lead to broader improvements in forensic practice, cyber awareness, and professional ability, ultimately improving the digital security ecosystem for Opera, Chrome, Firefox, Edge, and Brave using a Python CLI.

**Target Users:** The primary users are digital forensic analysts, who will use the ‎Python CLI tool to extract critical evidence from disk images for Opera, Chrome, Firefox, Edge, ‎and Brave, CTF players, who will use it for successful password recovery in challenges ‎about these browsers, cybersecurity professionals, who will analyze compromised systems ‎and conduct simulated attacks, corporate IT and security teams, who will ensure policy compliance and ‎resource protection, and ethical hackers, who will test browser-stored credential security for ‎these browsers using the Python command line. This diverse group of people illustrates the extensive application and impact of the Python ‎CLI tool.

## 1.7 Overview of the IR

This **Investigation Report (IR)** delivers a detailed breakdown of the ***Password ‎Extractor and Decryptor from Browser-Based Folders in the Forensic Images*** Python-based CLI ‎project, acting as a road map to the understanding of its creation, challenges, and relevance to ‎Opera, Chrome, Firefox, Edge, and Brave.

* **Chapter 1** introduces the project, which includes its place ‎in digital forensics and cybersecurity, the problem that it addresses, its intention, aims, focus, ‎and probable impact. It sets the stage for the subsequent chapters, which will expound on ‎this introduction to delve further.‎
* **Chapter 2** will be a review of literature that explains existing research on browser password storage, ‎encryption techniques, forensic Python command-line tools, and CTF challenge methods for Opera, ‎Chrome, Firefox, Edge, and Brave and how the gaps are addressed by the Python CLI project.
* **Chapter 3** will ‎explain the methodology describing the design, implementation and ‎testing procedures, and user feedback mechanisms of these browsers.
* **Chapter 4** will report findings, contrasting the Python CLI tool performance in extraction accuracy, decryption success, speed, and ease of use between Opera, Chrome, Firefox, Edge, and Brave, and finally will summarize the main findings, recommendations for future revisions, and how the tool can potentially have broader use in digital forensic and cybersecurity communities for the above-mentioned browses.

‏

## 1.8 Project Plan

|  |  |  |
| --- | --- | --- |
| ***Tasks*** | ***Start Date*** | ***Completion*** |
| *Abstract* | *12/2/2024* | *12/7/2024* |
| *Chapter 1: Introduction* | *12/9/2024* | *12/28/2024* |
| *1.1 Introduction* | *12/9/2024* | *12/10/2024* |
| *1.2 Problem Background* | *12/11/2024* | *12/12/2024* |
| *1.3 Project Aim* | *12/13/2024* | *12/14/2024* |
| *1.4 Objectives* | *12/15/2024* | *12/16/2024* |
| *1.5 Scope* | *12/17/2024* | *12/18/2024* |
| *1.6 Potential Benefit* | *12/19/2024* | *12/20/2024* |
| *1.7 Overview of the IR* | *12/21/2024* | *12/22/2024* |
| *1.8 Project Plan* | *12/22/2024* | *12/28/2024* |
| *Chapter 2: Literature Review* | *12/16/2024* | *1/19/2025* |
| *2.1 Domain Research* | *12/16/2024* | *1/2/2025* |
| *2.2 Similar System* | *12/23/2024* | *1/10/2025* |
| *2.3 Technical Research* | *12/30/2024* | *1/19/2025* |
| *Chapter 3: Methodology* | *1/20/2025* | *2/23/2025* |
| *3.1 System Development Methodology* | *1/21/2025* | *2/17/2025* |
| *3.2 Data Gathering Design* | *1/23/2025* | *2/20/2025* |
| *3.3 Analysis* | *1/26/2025* | *2/23/2025* |
| *Chapter 4: Conclusion* | *2/24/2025* | *2/28/2025* |

1: Project Plan

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Domain Research

**General Overview**

Web browsers are an essential part of our daily online activities, and they store a tremendous amount of ‎sensitive information, including passwords, history, and cookies. To protect the data, the browsers employ ‎encryption techniques; in this way, user data remain protected. Knowing how browsers such as Chrome, Firefox, Edge, Opera, and Brave employ encryption and store data is crucial, particularly in digital forensic analysis. Whereas forensics involve images such as.dd or.raw files, decrypting and analysis of this information is of major concern to the investigators.

Google Chrome, one of the most popular browsers, stores sensitive data like passwords in an encrypted SQLite database file named "Login Data." The file is normally located in the user's profile folder in Windows operating systems (google chrome help , 2025). Chrome uses the AES encryption algorithm to safeguard passwords, and the encryption key is derived from the user's system credentials. This means the associated key is required to open the ‎encrypted information, and this can be problematic for forensic ‎examiners when the system credentials are unavailable.‎

Mozilla Firefox does it a bit differently. It stores passwords in a file named logins.json or key4.db, which is located in the user's profile directory. (Firefox, Mozilla Account , 2024). Firefox also uses AES encryption, but the key to the encryption is stored in another file. Firefox also offers a master password option, which gives an extra layer of security. If a master password has been established, investigators will need to break it to view the ‎encrypted information, which complicates the process.

The chromium-based Microsoft Edge is quite similar to Chrome in terms of ‎encryption of data and storage. All the passwords are saved in an SQLite database file named "Login Data" and are encrypted in AES. Just like in Chrome, the encryption key is tied to the user's system credentials so that the investigators need to obtain the key to decrypt the data.

Opera, another browser based on Chromium, is identical. Opera also stores passwords in an encrypted SQLite database file named "Login Data" within the user's profile directory. The encryption algorithm utilized is AES, and the key is generated from the user's system credentials. This similarity between Chromium-based browsers makes forensic examiners face the same issues when they attempt to decrypt the information.

Brave, also Chromium-based, stores passwords in an SQLite database file named "Login ‎Data" within the user's profile directory. It uses AES encryption, the key is bound to the ‎user's system credentials. Decryption of the information, similar to other Chromium-based browsers, occurs using the encryption key.‎

In forensic examination, decryption and browsing data extraction from forensic images like.dd ‎or.raw files is a challenging yet crucial task (marcus, 2012)*.* The investigators have to identify where the encrypted files are stored and obtain the needed encryption keys. This can involve checking the user credentials of the system or the use of advanced tools that are utilized to extract and decrypt the browser data. Awareness of the single storage locations and schemes of encryption utilized by all browsers is central to the successful recovery of data as well as inspection and, therefore, an indispensable part of computer forensics.

### 2.1.1 Study of Domain from General to Specific

**General Overview of Browser Password Storage**

Web browsers are a fundamental aspect of modern internet usage, designed to make the user's experience more ‎convenient by storing sensitive data such as usernames, passwords, and history. Though this attribute makes the user interface more convenient, it also offers a goldmine of data that is crucial for forensic investigation. During the examination of forensic images like.dd or.raw files, accessing and decrypting this data is a top priority. To safeguard user data, browsers utilize encryption and secure storage practices with ‎the intent to avoid unauthorized access. Encryption keys are usually bound to the operating ‎system's secure storage or user-defined master passwords, which further secures the process. ‎This convergence of browser password storage, cybersecurity, and digital forensics emphasizes the need to understand and automate the decryption and extraction process.

Within digital forensics, the ability to decrypt and retrieve browser details from forensic ‎images is imperative for investigations as well as CTF challenges. Forensic experts are likely to employ software programs like Autopsy, FTK, and EnCase to examine digital evidence. ‎These applications, however, lack prowess in handling browser information. (BasuMallick, 2021). They lack specialized functionality to extract browser folders and decrypt stored information, thus, the process is inefficient and time-consuming. This lack of functionality emphasizes the necessity of a more efficient and automated process of browser data extraction and decryption.

The project seeks to overcome these inefficiencies by developing a tool that automates browser data extraction and decryption from forensic images. Browsers like Chrome, ‎Firefox, Edge, Opera, and Brave store sensitive information in encrypted formats, often using ‎state-of-the-art encryption algorithms such as AES. The encryption keys are typically tied to the ‎operating system's secure storage or protected by master passwords selected by the user. For example, ‎Chrome and other Chromium browsers store passwords in an encrypted SQLite database ‎file, where the encryption key is derived from the user's system credentials. Firefox, however, employs a master password option, introducing yet another level of complexity for forensic examiners.

Automation of the decryption and extraction process involves identifying the locations where browser data is stored in forensic images and where the encryption keys are located and applying the appropriate decryption algorithms. This is sophisticated knowledge of how browsers store data and encrypt it. Automating the process will reduce the time and effort required to perform forensic analysis, freeing the investigators to spend their time analyzing the extracted data rather than decrypting it manually.

The uses of this project extend beyond forensics. In CTF challenges, where competitors typically must extract and decrypt sensitive information from forensic images, an automated tool would be highly useful. It would also prove to be a valuable tool for cybersecurity professionals, helping them identify vulnerabilities and fortify the security of browser data storage mechanisms.

**Browser-Specific Storage Details**

All browsers employ various methods to encrypt and store passwords, reflecting their ‎respective architectures and security approaches. Familiarity with these ‎differences is essential for forensic analysis as it enables examiners to locate and decrypt such ‎sensitive information with success. The subsequent discussion is a detailed description of how ‎popular browsers like Google Chrome, Microsoft Edge, Mozilla Firefox, Opera, and Brave store ‎and encrypt passwords. (Texas Tech Security Group, 2013).

**Google Chrome and Microsoft Edge**



2 ‎Google Chrome & Microsoft Edge‎

Google Chrome and Microsoft Edge, both operating the Chromium engine, share the same ‎processes in storing and encrypting passwords.

* **Filename Location:**

Windows Chrome stores passwords within the Login Data file at *%LocalAppData%\Google\Chrome\User Data\Default\*. Microsoft Edge, as the Chromium engine, shares the same structure with its Login Data file located at *%LocalAppData%\Microsoft\Edge\User Data\Default\*. On Linux, the password database of Chrome is at *~/.config/google-chrome/Default/Login Data*, and Edge's is at *~/.config/microsoft-edge/Default/Login Data.*

* **Encryption:**

Both browsers use AES-256 encryption to secure stored passwords. (Google Chrome Help, 2019) The encryption key is protected ‎by the secure storage mechanisms of the operating system. On Windows, it is handled by ‎the Data Protection API (DPAPI) and links the encryption key to the user's login ‎credentials. On Linux, the key may be stored in the GNOME Keyring or KWallet, depending on the desktop environment.

**Mozilla Firefox**



3 Mozilla Firefox

Mozilla Firefox stands out when it comes to saving and encrypting passwords, a feature that reflects its ‎autonomous development and respect for user privacy.‎

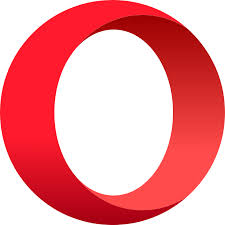
* **File Location:‎**

Firefox stores passwords within the user's profile directory. On recent versions, the passwords are stored within a logins.json file, while older versions use signons.sqlite. The location of the profile directory varies depending on the operating system. On Linux, it's typically at *~/.mozilla/firefox/,* while on Windows, it's at *%AppData%\Mozilla\Firefox\Profiles\.‎*

* **Encryption:**

Firefox uses the Triple Data Encryption Standard (3DES) to encrypt the logins.json file. (Firefox, Mozilla Account , 2024).The encryption key is ‎stored in another file named key4.db (or key3.db in older Firefox versions). Firefox also ‎offers a master password feature, which adds an extra layer of security. If a master password is used, it protects the key by encrypting in key4.db; otherwise, a default key is utilized that may be unsafe. Unlike browsers based on Chromium, Firefox adopts Mozilla's Network Security Services (NSS) library for cryptographic function, so the encryption process would be different in Firefox.

**Opera and Brave**



4 Opera ‎



5 Brave

Opera and Brave, both of which are based on the Chromium engine, share a great deal with Chrome ‎and Edge when it comes to password storage and encryption.‎

* **File Location:**

Opera's Login Data file is located at *%LocalAppData%\Opera Software\Opera Stable\* on ‎Windows, and at *%LocalAppData%\BraveSoftware\Brave-Browser\* on Brave. ‎Linux locations are *~/.config/opera/* and *~/.config/BraveSoftware/Brave-Browser/,* ‎for Opera and Brave, respectively.‎

* **Encryption:**

Like Chrome and Edge, Opera and Brave also use AES-256 encryption to safeguard ‎passwords. (Brave, Help Center, 2024)*.* The key to encryption is secured through the secure storage mechanisms of the operating system. On Windows, it is handled by DPAPI, while on Linux, the key is stored in the system keyring. This keeps the encryption key tied to the user credentials, and it would be difficult for others to get access to the data.

**Forensic Implications**

The differences in storage places and encryption methods between browsers have significant ‎forensic consequences. An example is extracting passwords from a forensic image that entails the ‎identification of the correct file paths and understanding of the encryption process used by each ‎browser. Software like Autopsy, FTK, and EnCase do not have particular browser data extraction ‎and decryption functionality. (Apr4h, 2019), the procedure is prolonged and inefficient. It might be highly optimized for automation to enhance forensic analysis efficiency, particularly in ‎cases like Capture The Flag (CTF) challenges, where fast and accurate data extraction is of utmost importance.‎

**Challenges in Forensic Analysis**

Browser password database extraction and decryption from forensic images is a complex operation that presents a series of severe challenges. The challenges are a result of the variety of encryption methods, the protection of the encryption key, and the limitations of existing forensic tools. (Zakuskina, 2023). Overcoming these challenges is necessary for improving the efficiency and precision of forensic analysis, particularly in cases like Capture The Flag (CTF) competitions, where time and precision are essential.

**Encryption Consistency**

One of the largest hurdles in forensic examination is that various browsers use inconsistent methods ‎of encryption. For instance, Chromium-based browsers like Google Chrome, Microsoft Edge, Opera, and Brave use AES-256 encryption to secure stored passwords (Luc, Tony Ruth, 2017). Mozilla Firefox, on the other hand, uses the Triple Data Encryption Standard (3DES) for password storage. This disparity necessitates that forensic examiners are familiar with several ‎encryption algorithms and utilize specific decryption protocols for every browser. The lack of a uniform ‎method of encryption across browsers makes it challenging to retrieve and decrypt information, which requires special ‎expertise and equipment particular to each browser's encryption mechanism.‎

**Key Access**

Another significant challenge is accessing the encryption keys used to decrypt browser data. Such keys are typically secured through the safe storage of the operating system, such as the Data Protection API (DPAPI) in Windows, GNOME Keyring, or KWallet in Linux. (Chromium Docs, 2025). Such measures tie the encryption keys to the login credentials of the user, making it difficult to gain access to the keys in the absence of the user's system credentials or physical access to the system. This poses a unique challenge to offline forensic analysis when the original system or user passwords cannot be accessed by investigators. As a result, decrypting browser data often requires additional steps, such as the recovery of encryption keys from the system's secure storage or bypassing password protection using brute-force techniques.

**Tool Limitations**

Current forensic tools, such as Autopsy, FTK, and EnCase, are widely used in digital evidence analysis. But such tools don't typically have advanced functionality for handling browser-specific data. For example, they may not be geared to pluck files like Chrome's "Login Data" or Firefox's "logins.json" from forensic images. (Higor Diego, 2023). In addition, such tools may not have optimized means of decrypting the extracted data, especially when dealing with large forensic images (e.g., 10GB or more). This bottleneck can significantly prolong investigation timelines because investigators have to resort to using manual techniques or bespoke scripts to import and decrypt browser data. These inefficiencies during CTF competition, where often competitors are presented with the requirement to review substantial forensic images under a time pressure, can deter performance and decrease the chances of success.

**Crossing the Difficulties**

To overcome these challenges, this project aims to develop a Python Command Line ‎Interface (CLI) tool that will automate browser data scanning, extraction, and decryption from ‎forensic images. The tool will be capable of overcoming the variation in encryption methods by ‎supporting browser-specific decryption algorithms. It will also address the problem of key access by enabling mechanisms to retrieve encryption keys from the secure storage of the operating system or by using brute-force techniques for password bypass protection.

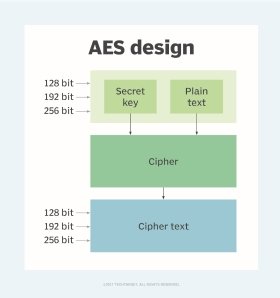
By automating these processes, the tool will significantly enhance the efficiency and accuracy of ‎forensic investigations. It will assist investigators in extracting and decrypting browser data from ‎forensic images quickly, reducing the time and effort involved in manual analysis. The tool will also be ‎significantly useful for CTF participants since it will be a powerful aid in surmounting the challenges ‎of big forensic images.

### 2.1.2 Study of Related Theories/Technologies/Algorithms

**Encryption Algorithms**

Encryption is the basis of password storage in browsers, which protects sensitive data like ‎usernames and passwords. Different browsers employ different encryption algorithms, each with its own characteristics and security implications. Below, we present the two primary encryption algorithms used by modern browsers: AES-256 and 3DES. (Obremski, 2009).

**AES-256 (Advanced Encryption Standard with 256-bit Key)**



6 AES-256 ‎Design

* **Description:**

AES-256 is a symmetric block cipher widely regarded as one of the most secure encryption algorithms available today. It encrypts 128-bit data blocks using a 256-bit key, and it is highly resistant to brute-force attacks. AES-256 is the encryption standard used by Chromium-based browsers like Google Chrome, Microsoft Edge, Opera, and Brave (Óscar, 2010). Its strength and efficiency have rendered it the most popular choice for encrypting sensitive information.

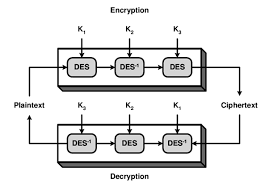
* **Advantages:**

AES-256 is light on computations and very secure due to its ‎large key size. AES-256 is employed so universally that there are no chances of platform and ‎system compatibility, hence making it a sure shot to store passwords in a browser. The security of AES-256, though, relies upon keeping the encryption key safe, which is again safeguarded by the secure storage facilities of an operating system in most scenarios.

* **Applicability to the Project:**

The project must implement AES-256 decryption to decrypt data from Chromium-based ‎browsers. This involves retrieving the encryption key, which is typically protected by the ‎secure storage of the operating system (e.g., DPAPI on Windows). Python libraries like ‎cryptography or pycryptodome can be utilized to handle AES-256 decryption, offering compatibility and performance.

‎**3DES (Triple Data Encryption Standard)‎**



7 Design of 3DES

* **Overview:**

3DES, or Triple Data Encryption Standard, is an older encryption ‎algorithm that uses the DES algorithm three times with three different keys. ‎Though designed to enhance the security of the original DES, which had ‎become vulnerable to brute-force attacks, 3DES is no longer considered as ‎secure as AES due to its smaller key space and lower speed. Mozilla ‎Firefox uses 3DES to encrypt password storage, specifically in the logins.json ‎file (Anders, Vilius Povilaika, 2019)*.*

* **Strengths and Limitations:**

‎3DES is more secure than single DES as it uses the encryption algorithm three times. Nevertheless, its smaller key size (equivalent to 112 bits) and slower speed cause it to be less effective compared to AES. Furthermore, 3DES is being replaced with newer algorithms such as AES that have greater security and performance.‎

* **Relevance to the Project:**

The project must include 3DES decryption to be able to handle data from Firefox. This involves decrypting the logins.json file, which, in some cases, is encrypted with a master password. One can use libraries like Mozilla's Network Security Services (NSS) or Python-based implementations like pycryptodome to include 3DES decryption. Consideration of cases where a master password is set must be given serious thought since this adds another complexity to the decryption process.

* **Relevance to the Project:**

The choice of encryption algorithms has a direct bearing on the design and implementation of the ‎project. AES-256 and 3DES decryption must both be supported to support compatibility with the ‎major browsers. The project must also address issues around access to encryption keys, typically protected by the operating system's secure storage systems. Making use of Python packages like cryptography and pycryptodome, the tool can efficiently handle the decryption without sacrificing flexibility as well as security.‎

**Secure Storage Mechanisms**

Secure storage practices are crucial in protecting encryption keys and keeping sensitive information out of the hands of unauthorized users. Different operating systems have their secure storage practices with varying challenges and implications for forensic analysis. (setevoy, 2019).

**DPAPI (Data Protection API)‎**

* **Overview:**

In the Windows operating system, the Data Protection API (DPAPI) is used to encrypt and protect sensitive data, such as encryption keys, using the user's login credentials. DPAPI is integrated into the operating system security framework to limit access to decrypting data so that decryption can only be done when the user is logged in. DPAPI is used by Chromium-based web browsers such as Chrome, Edge, Opera, and Brave to protect their encryption keys (Microsoft Edge Learn, 2024).

**• Challenges:**

DPAPI is a challenge for forensic analysis as the encrypted data can be decrypted only with the login credentials of the user. If the credentials are missing or unknown, it is difficult to access the encryption keys. Investigators may have to rely on techniques like key extraction from memory or system files, which are time-consuming and cumbersome.

• **Relevance to the Project:**

The tool must be capable of handling DPAPI-protected keys, which may mean integration with Windows APIs or forensic approaches to extract keys from memory or system files. pywin32 libraries can be utilized to interface with DPAPI, so the tool is capable of decrypting data encrypted using this method.

**Keyring (Linux)**

* **Encryption:**

Secure storage in Linux is typically managed by systems like GNOME Keyring or ‎KWallet. The systems store encryption keys securely and require system ‎authentication before they can be accessed. If there is no keyring, then the passwords are stored in ‎plain text, greatly reducing security. (James McCormack, Anandu M Das, 2014).

* **Difficulty:**

Accessing encryption keys stored in a keyring is subject to authentication by the system, which renders forensic analysis more difficult if there are no available user credentials. Additionally, when there is no keyring system, passwords can be stored in an insecure way, which exposes them more readily but with less reliability for forensic purposes.

* **Relevance to the Project:**

The software needs to detect and integrate with Linux keyring facilities so that decryption is possible in case keys are stored there. Python libraries like secretstorage may be used for integration with GNOME Keyring and dbus-python for KWallet integration.

**Database Formats**

Browsers store certain database formats to store sensitive data like passwords and surfing history. (Phiter, 2016) These formats should be known to extract and decrypt data at the time of forensic analysis.

**SQLite**

* **Overview:**

SQLite is a lightweight, serverless database format used by Chromium browsers like Chrome, Edge, Opera, and Brave to store passwords in the "Login Data" file. SQLite databases store data in a single file and support SQL queries, making them easy to parse and analyze (Vidyut, 2013).

* **Challenges:**

While SQLite databases themselves are fairly easy to query, the data maintained within them is ‎typically encrypted. Forensic examination has to decode the database and decrypt the ‎encrypted data fields, and without the encryption keys, this becomes extremely challenging.‎

* **Relevance to the Project:**

The utility must read SQLite databases using Python libraries like sqlite3 and handle ‎encrypted fields for decryption. This involves fetching the encrypted data and using the appropriate decryption algorithms, for instance, AES-256 for Chromium-based browsers.

**JSON**

* **Overview:**

JSON (JavaScript Object Notation) is a human-readable format used by Firefox to store ‎passwords in the logins.json file. JSON is easy to parse and widely supported, which makes it a convenient choice for storing structured data.‎

* **Challenges:**

While JSON files are easy to parse, the data in them with sensitive information is typically encrypted. ‎Firefox encrypts data in the logins.json file with 3DES, so the information must first be decrypted before ‎it can be read. (Olhanzilla, 2017). **Relevance to the Project:**

The utility must parse JSON files with Python's JSON module and decrypt the fields that are encrypted. This must involve decrypting the data with 3DES, which might require libraries like pycryptodome or integration with Mozilla's NSS library.

**Forensic Tools and Techniques**

Forensic analysis typically involves working with forensic images, which are bit-for-bit copies of ‎storage media. Such images, typically in.dd or.raw form, record all data from the source device so that investigators can analyze the contents without altering the source. ‎Despite being employed extensively for such ends, general-purpose forensic tools like Autopsy, FTK, and EnCase are not browser-specific to handle browser-specific information, including password databases. (Unode, 2024). This is a shortcoming in forensic processes, particularly when handled with encrypted browser data. ‎The Python CLI tool in this project aims to bridge this gap by automating the extraction and decryption of browser password databases for both forensic analysis and Capture The Flag (CTF) challenges to improve efficiency and accuracy.‎

**Limitations of Existing Forensic Tools**

* **General-Purpose Nature:**

Tools like Autopsy, FTK, and EnCase are general-purpose forensic tools that support a broad scope of forensic analysis like file recovery, disk imaging, and metadata analysis. They are not optimized, however, for browser-specific data decryption and extraction. For example, they don't natively support extracting files like Chrome's "Login Data" or Firefox's "logins.json" and decrypting them automatically.

* **Manual Processes:**

Without the use of specialized tools, forensic examiners have to employ manual ‎methods to locate and extract browser information from forensic images. This means ‎locating the files of interest, such as "Login Data" or "logins.json," and then decrypting the information ‎using standalone software or scripts. These manual methods are time-consuming and prone to errors, especially when dealing with large forensic images.

* **Lack of Automation:**

Current utilities do not have automated workflows to scan forensic images, identify browser-specific folders, and decrypt data that is pulled out. Such a lack of automation slows down investigations and lowers the productivity of forensic analysts, particularly in situations where time is of the essence, like in CTF competitions.

* **The Project's Python CLI Tool:**

The project aims to surpass these constraints by developing a Python Command Line ‎Interface (CLI) tool that automates browser password database extraction and decryption from forensic images. The tool is designed to speed up the forensic process, making it efficient and accurate.‎

* **Automated Scanning:**

The program will look for browser-specific directories in forensic images, i.e., User ‎Data of Chrome or profile directories of Firefox. The automation of the search by the program eliminates the need to perform manual file searching, which consumes time and reduces the likelihood of missing crucial data.

* **File Retrieval:**

Once the folders of interest are identified, the utility will download the needed files, such as ‎‎"Login Data" for Chromium browsers or "logins.json" for Firefox. This ensures all the needed data is extracted efficiently, including from large forensic images.

* **Decryption:**

The tool will perform decryption algorithms on the extracted data, both AES-256 (used by Chromium-based browsers) and 3DES (used by Firefox). By decrypting the data automatically, the tool eliminates the need for manual decryption processes, improving accuracy and reducing opportunities for errors.

* **Efficiency and Scalability:**

The software is designed to handle large forensic images rapidly, thus being beneficial in ‎both forensic investigations and CTF challenges. Automated processes within the software ensure that ‎investigators do not spend time manually performing extraction and decryption processes.

**Relevance to Forensic Investigations and CTF Challenges**

* **Forensic Investigations:‎**

The ability of the tool to automate decryption and parsing of browser evidence ‎significantly enhances the productivity of forensic examination. It helps investigators ‎quickly access useful information, such as cached passwords, that may be essential ‎in case of a breakthrough or revelation of evidence.‎

* **CTF Challenges:**

In CTF competitions, participants are typically required to examine forensic images and ‎extract sensitive data within strict time constraints. The tool's automation capabilities provide ‎an excellent advantage by enabling participants to locate and decrypt browser data very ‎quickly, which increases their chances of winning.‎

### 2.1.3 Review of Sub-Topics within the Domain

**Recent Developments and Vulnerabilities**

The security of browser password managers and how they store and secure sensitive ‎information has been a subject of extensive study and assessment. Recent events and ‎vulnerabilities indicate the complexity involved in browser password storage and the need for ‎robust forensic tools for problem-solving in this context. This section presents significant findings ‎and trends in browser password security, including vulnerabilities, user behavior, and emerging ‎technologies that can shape the future of forensic analysis.

**Browser Password Manager Security**

* **Password Storage Vulnerabilities:**

Research, such as The Problem With Storing Passwords in Your Browser, has discovered ‎significant vulnerabilities in browser password managers. One of the primary issues is the ‎accessibility of encryption keys to malware, which can exploit the vulnerabilities in the ‎storage systems. (Zakuskina, 2023). For example, password stealers can target browser-stored passwords that are either stored in plain text or encrypted using easily accessible keys. These vulnerabilities call for the development of forensic tools to process encrypted information with a fix of potential security vulnerabilities.

* **Relevance to Forensic Tools:**

Forensic tools must overcome these deficiencies through the application of effective decryption ‎methods and proper encryption key management. The Python CLI tool of this project aims to alleviate these problems by automating the extraction and decryption of browser data, even in instances where encryption keys are accessible or compromised.‎

**Password Syncing and Cloud Storage**

* **Synced Password Security:**

Studies, such as How Chrome Protects Your Passwords, have researched password syncing security across devices. Synched data is commonly encrypted, but server-side vulnerability or ‎intrusion could expose synced credentials. (Google Chrome Help, 2019). This ‎is of particular significance for forensic analysis of cloud storage data, when investigators would perhaps have to obtain access to ‎passwords stored in remote servers.‎

* **Relevance to Forensic Tools:**

The tool for the project should be able to consider password syncing and cloud storage concerns, offering both local and cloud data management. This means decrypting synced passwords and dealing with any cloud storage mechanism vulnerabilities.

**User Behavior and Password Management**

* **Browser Password Manager Use:**

The utilization of browser password managers for convenience has been a widespread practice among most users, but they compromise security if bad practices like the lack of master passwords are utilized. (Viezelyte, 2024 ). For example, studies like Are Browser-Stored Passwords Secure? Indicate the susceptibility to attacks presented by weak encryption where master passwords have not been utilized, making stored credentials vulnerable.‎

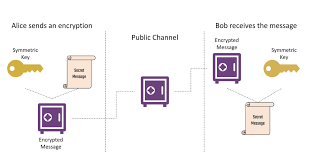
* **Impact on Forensic Recovery:**

User behavior has a direct impact on forensic recovery activities. Weak encryption or the absence of master passwords may make the decryption of stored data easier, but it also poses concerns concerning the protection of sensitive information. The project tool must take such circumstances into account, providing options for decryption when master passwords are disabled.

**Research on Improving Browser Password Security**

As browser password storage mechanisms evolve, new technologies and approaches are being developed to enhance security. These advancements have implications for forensic tools, which must adapt to handle emerging encryption methods and storage techniques.

**Post-Quantum Cryptography**



8 How Post-Quantum Cryptography Works

* **Shift to Quantum-Resistant Algorithms:**

Research is now geared towards the move to quantum-resistant encryption algorithms,

as current standards like AES and 3DES will be vulnerable to quantum computing

attacks. This shift is particularly relevant to the long-term evolution of forensic tools

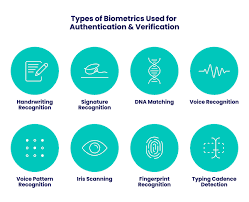
in a way that tools remain useful regardless of future threats. (Computer Security Resource Center, 2025).

* **Relevance to Forensic Tools:**

The project tool must consider the future impact of post-quantum cryptography, incorporating support for new encryption algorithms after they become widely adopted.‎

This will maintain the tool as future-proof and in a place to handle evolving‎ security standards.‎

**Biometric Authentication**



9 Biometric Authentication Types

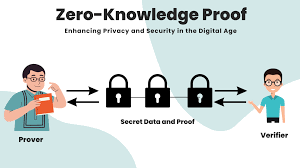
* **Integration with Biometric Systems:‎**

The inclusion of biometric authentication, such as Touch ID and Windows Hello, enhances ‎the security of browser password managers. (Firefox, Mozilla Account , 2024) Such systems protect encryption keys with biometric data, adding a ‎second layer of security.‎

* **Relevance to Forensic Tools:**

Forensic tools must be developed to handle biometric-secured keys, which may include ‎special decryption procedures. The project tool must explore methods of communicating ‎with biometric systems or extracting keys secured with biometric authentication.‎

**Zero-Knowledge Proofs**



10 Zero-Knowledge Proofs Illustration

* **Enhancing Privacy in Password Storage:‎**

Zero-knowledge proof studies aim to make password storage more private by rendering it impossible ‎for servers to obtain decrypted data. Although this heightens security, it also complicates forensic ‎investigation because investigators will be forced to study encrypted data without decryption keys. (Chainlink, 2024).

* **Relevance to Forensic Tools:**

The project tool must consider the implications of zero-knowledge proofs so that it can process encrypted data while maintaining privacy and security. This entails developing methods of decrypting data without revealing sensitive information.

**Research on Enhancing Browser Password Security**

With the advancement of browser password storage technologies, new techniques and strategies are under development to enhance security. Such progress affects forensic tools, which must keep pace with new encryption methods and methods of storage.

## 2.2 Similar Systems/Works

## 2.2.1 Similar Systems

**Firefox Decrypt by Unode**

The firefox\_decrypt tool, at ***https://github.com/unode/firefox\_decrypt/tree/main***, is an open-source Python-based utility designed to extract and decrypt passwords from Mozilla-based applications, including Firefox. Developed by unode, this tool targets user profiles stored by these applications, focusing on files like logins.json and key4.db (or key3.db in older versions). It utilizes Mozilla’s NSS (Network Security Services) library to decrypt credentials, supporting the 3DES encryption used by Firefox, which differs from the AES-256 encryption in Chromium-based browsers addressed by this project.

A screenshot of a computer

AI-generated content may be incorrect.

11 Firefox Decrypt by "unode" Repo

In contrast to the ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project, which automates password recovery across multiple browsers (Chrome, Firefox, Opera, Edge, Brave) from forensic image exports using Pycryptodome, firefox\_decrypt has a narrower focus, targeting only Mozilla products and requiring direct access to a live profile rather than forensic images. It can decrypt credentials with a known master password or extract unencrypted data if no master password exists, but it lacks brute-force capabilities, limiting its forensic versatility compared to this project’s broader, image-centric approach. Nevertheless, its command-line interface (CLI) and output options (e.g., CSV, tabular) offer parallels to this project’s usability goals, providing lessons in effective CLI design for password recovery tools.

A computer screen shot of a black screen

AI-generated content may be incorrect.

12 Firefox Decrypt Tool Result

Executing the tool in a terminal prompts the user to choose a Firefox profile from a list sourced from configuration data. After selecting a profile and, if needed, entering a master password, the tool displays decrypted credentials in a readable format: ***Website: https://github.com, Username: 'octocat', Password: 'qJZo6FduRcHw'***, followed by additional login entries. The output can be saved as CSV (e.g., python firefox\_decrypt.py --format csv > output.csv), yielding structured data like "https://login.example.com", "john.doe", and "1n53cur3". This showcases its ability to efficiently extract and present credentials, though errors such as **decryption failure** may occur if the NSS library is incompatible or the master password is incorrect, revealing dependency issues not present in this project’s standalone Pycryptodome implementation.

The firefox\_decrypt tool serves as a useful reference for this project, demonstrating strengths in Mozilla-specific decryption and CLI usability, but its dependence on live system access and single-browser focus highlight the distinct forensic emphasis and wider applicability of this project’s design, which processes exported folders from forensic images across multiple browsers. (Unode, 2024).

**Decrypt Chrome Passwords by ohyicong**

The decrypt-chrome-passwords tool, at ***https://github.com/ohyicong/decrypt-chrome-passwords?tab=readme-ov-file***, is an open-source Python-based utility created by ohyicong to decrypt passwords saved in Google Chrome. This tool targets Chrome’s password storage, specifically the SQLite database file containing encrypted credentials, and uses Windows’ CryptUnprotectData function via the win32crypt library to decrypt them. It focuses on AES-256 encryption with keys secured by the Windows Data Protection API (DPAPI), differing from the 3DES approach of Firefox handled in this project.

A screenshot of a computer

AI-generated content may be incorrect.

13 Decrypt Chrome Passwords by "ohyicong" Repo

Compared to the "***Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project, which automates password recovery across multiple browsers (Chrome, Firefox, Opera, Edge, Brave) from forensic image exports using Pycryptodome, decrypt-chrome-passwords is more limited, focusing solely on Chrome and requiring a live Windows system rather than forensic images. It decrypts passwords, assuming the user is logged into the same account that encrypted them, lacking support for offline forensic analysis or other browsers, unlike this project’s broader scope with Pycryptodome for both AES-256 and 3DES. However, its simplicity and CLI-based output offer insights into designing straightforward, effective password recovery interfaces, resonating with this project’s usability goals.

A computer screen with white text

AI-generated content may be incorrect.

14 Decrypt Chrome Tool Result

Running the tool (e.g., python decrypt\_chrome\_password.py in a terminal) produces a list of decrypted Chrome credentials directly in the console, such as ***URL: https://example.com, Username: 'user123', Password: 'passw0rd'***, followed by additional entries from the stored logins. The output is plain text, displaying each website’s URL, username, and decrypted password in sequence, with no additional formatting options like CSV export. This demonstrates its ability to quickly extract and display passwords, though it may fail with errors like CryptUnprotectData failed if run outside the original user context or on non-Windows systems, highlighting its dependency on live system access unlike this project’s standalone, forensic-focused approach with exported data.

The decrypt-chrome-passwords tool provides a valuable comparison for this project, showcasing strengths in rapid Chrome-specific decryption and ease of use on live systems, but its single-browser focus and lack of forensic image support emphasize the enhanced versatility and investigative applicability of this project’s design, which targets exported folders across multiple browsers using a unified CLI solution. (Ohyicong, 2020).

### 2.2.2 Strengths and Weaknesses

**Firefox decrypt**



15 Firefox

Its strengths include robust decryption of Mozilla-specific 3DES-encrypted credentials, leveraging NSS for accuracy, and flexible output options (e.g., plain text, CSV), enhancing usability for users familiar with Mozilla ecosystems. It efficiently handles profiles with or without master passwords, making it versatile for basic recovery tasks. However, its weaknesses are significant: it requires a live system with NSS installed, lacks support for forensic images, and is limited to Mozilla products, excluding other browsers like Chrome or Edge. Dependency on external libraries and potential decryption failures (e.g., **decryption failed** ) if NSS is misconfigured further restricts its forensic applicability.

**decrypt-chrome-passwords**



16 Google Chrome

This tool excels in simplicity and speed for Chrome password extraction on Windows, using win32crypt to decrypt AES-256 credentials within the original user context. Its minimal setup and straightforward output (e.g., URL, username, password in plain text) make it accessible for quick demonstrations or live-system audits. Weaknesses include its narrow scope, Chrome-only and Windows-only rendering it incompatible with Linux or other browsers like Firefox. It also fails in offline forensic scenarios or if run outside the encrypting user’s session (e.g., CryptUnprotectData failed), lacking the forensic image support and multi-browser versatility of this project.

### 2.2.3 Conclusion

The firefox\_decrypt and decrypt-chrome-passwords tools highlight valuable approaches to browser password recovery, with firefox\_decrypt offering flexibility for Mozilla users and decrypt-chrome-passwords providing simplicity for Chrome on Windows. Their strengths in targeted decryption and CLI usability inform this project’s design, particularly in output formatting and script efficiency. Yet, their weaknesses- limited browser support, live-system requirements, and lack of forensic image compatibility- underscore the innovation of this project’s approach. By automating password extraction from forensic exports across Chrome, Firefox, Opera, Edge, and Brave, this tool bridges gaps in existing systems, delivering a more comprehensive and adaptable solution for digital forensics and CTF challenges while maintaining usability and scalability within resource constraints.

## 2.3 Technical Research

### 2.3.1 Overview of Python as a Development Platform for Forensic Tools



17 Why Python?

Python has emerged as one of the most popular programming languages to utilize to develop forensic tools due to its ‎flexibility, huge library environment, and ease, which make it ideally fit for the ***‎‎"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project. ‎Its nature as an interpreted language supports quick development and testing, which is important for forensic ‎use cases where analysts must work quickly to keep pace with changing threats, like new browser encryption technologies in Opera, Chrome, Firefox, Edge, and Brave. Python's readability and simple syntax lower the learning curve for forensic analysts and CTF players, allowing them to concentrate on tool functionality instead of intricate code constructs. ‎This access convenience is also aligned with the goal of the project to have a convenient CLI ‎tool that does password recovery automatically from forensic images, which supports efficiency in ‎both investigative and competitive environments.

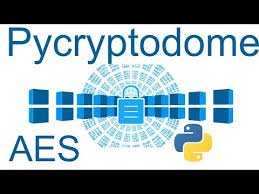
The biggest strength of Python in building forensic tools is the richness of libraries available, such as ‎pycryptodome, sqlite3, and json, which can be directly applied to this project's requirements. To give a few instances, pycryptodome facilitates the AES-256 decryption and ‎3DES-encryption of password information stored by Chromium-based browsers and Firefox, respectively, without explicitly building cryptographic procedures from scratch. Similarly, the sqlite3 ‎module makes it easier to communicate with SQLite databases like Chrome's "Login Data" file, whereas the ‎json module translates the Firefox "logins.json" format. These libraries provide robust, pre-tested solutions that ensure reliability and accuracy in decrypting and extracting browser credentials, resolving the inefficiencies of manual methods outlined in the project's problem background. Python's ability to integrate all these tools well contributes to developing an integrated CLI application that is compatible with the requirements of ‎forensics. (Alexander Fry, 2024).

Furthermore, the cross-platform nature of Python contributes to its applicability to forensic software since it allows the CLI tool of the project to execute in both Windows and Linux operating systems popular platforms utilized in forensic analysis and CTF challenges. This portability ensures that the ‎tool is capable of dealing with exported browser folders from forensic images (i.e.,.dd or.raw files) without regard to the investigator's or competitor's operating system, a crucial aspect in light of the ‎diversity of hardware restrictions outlined in the project scope. Python's support for system-level ‎interactions, for example, reading Windows DPAPI or Linux keyring systems using libraries like ‎pywin32 or secretstorage, enables the tool to retrieve encryption keys related to OS secure storage. ‎This plays a crucial role in decrypting browser data without requiring enormous manual ‎interaction, which aligns with the project's need to automate and accelerate password ‎recovery processes. (Frank Breitinger, Harald Baier, Douglas White , 2014).

‎ ‎

Lastly, the prevalence of Python among the cybersecurity and forensic communities encourages ‎collaboration and extensibility, which are two major benefits for a student project like this one. The open-source ‎nature of the language and the thriving community offer access to a community of existing resources, ‎tutorials, and third-party tools that can be exploited to enhance the functionality of the tool, such as ‎improving performance in support of large forensic images or improving CLI usability. This community backing also allows the tool to live beyond the project timeline and possibly future-proof through the addition of integration with upcoming encryption standards or future updates to browsers, thereby extending its utilization in digital forensic and CTF environments.

### 2.3.2 Utilization of Pycryptodome Library for Encryption and Decryption



18 Pycryptodome ‎ Library Used

The pycryptodome library forms the core of the ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project and provides the robust cryptographic functionality required to decrypt passwords saved by browsers like Opera, Chrome, Firefox, Edge, and Brave. Being a standalone Python library, Pycryptodome enhances the functionality of its ‎parent, PyCrypto, by offering a comprehensive collection of symmetric and asymmetric ‎cryptography methods, including AES-256 and 3DES, that are required to handle the ‎encryption models implemented by these browsers. Its application in this project makes the CLI tool successfully decrypt browser information extracted from forensic images, addressing the issue of ‎time-consuming and error-prone manual decryption procedures. By using ‎Pycryptodome, the tool ensures that forensic examiners and CTF players can access ‎credentials rapidly and accurately, which is the aim of the project to enhance investigative ‎efficiency. (pycryptodome, 2024).

Pycryptodome AES-256 decryption capability is particularly relevant to cracking data from Chromium-based browsers Chrome, Opera, Edge, and Brave because they store passwords in SQLite "Login Data" files encrypted with this algorithm. AES-256 is a symmetric block cipher with a 256-bit key that is guarded by operating system mechanisms like Windows DPAPI or Linux keyrings, meaning the tool will need to gain access and use the correct encryption key. Pycryptodome simplifies this by providing an AES module that can decrypt data once the key is obtained, either through system APIs or forensic techniques like memory extraction. This functionality eliminates the need for custom cryptographic implementations, making the tool dependable and conserving development time, which is crucial within the project's four-to-five-month timeline. Library efficiency in processing large datasets also validates the capacity of the ‎tool to perform well with big forensic images, a critical feature for applicability ‎in real life (Mihir Bellare, Anand Desai, David Pointcheval & Phillip Rogaway , 2006).

With Mozilla Firefox using 3DES to secure its "logins.json" file, Pycryptodome offers a ‎DES3 module that facilitates the decryption, achieving the project requirement to support ‏several encryption algorithms. Unlike AES-256, 3DES applies the Data Encryption Standard three times using a 168-bit key ‎(functionally 112-bit strength), and it keeps its key in ‎‎"key4.db," possibly with a master password. The fact that Pycryptodome can support this older, less-efficient algorithm makes supporting Firefox's NSS-based cryptography possible, allowing the tool to decrypt passwords with or without a master password. This ‎flexibility is required for forensic environments where user configurations are varied, and it mitigates ‎the limitations of general-purpose tools such as Autopsy or FTK, which lack special ‎browser-specific decryption, thus realizing the project objective of universal ‎browser support. (Nicola Leoone, 1997).

‎ ‎

Apart from decryption, the Python-friendliness of Pycryptodome enhances the usability and maintainability of the CLI tool, which is of utmost importance to its target users, forensic examiners, and CTF players. The library's well-documented API makes it easy to implement decryption workflows, including the extraction of keys, the creation of cipher objects, and the decryption of encrypted data into readable text outputs. Its open-source nature also guarantees that the tool will be able to adjust to future browser encryption changes, upholding the project's long-term viability. By reducing the complexity of the cryptographic operations, Pycryptodome provides the tool with the ability to present structured evidence, usernames, URLs, and timestamps effectively, meeting the project's usability goals while maintaining high security standards in handling sensitive forensic data. (Chitu Okoli, 2007).

### 2.3.3 Handling SQLite Databases in Browser Password Extraction



19 SQLite Databases

The ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project relies heavily on the processing of SQLite databases to extract passwords from Chromium-based browsers Chrome, Opera, Edge, and Brave that store credentials in encrypted "Login Data" files. SQLite, a lightweight, server-less database engine, is widely used for its simplicity and efficiency, and it is an ideal format for browser password storage in user profile directories (e.g., %LocalAppData%\Google\Chrome\User Data\Default on Windows). In this project, Python's sqlite3 library simplifies querying these databases so that the CLI tool can locate and retrieve encrypted password fields and metadata like usernames, URLs, and timestamps. This automation addresses the inefficiency of SQLite operations manually performed in forensic analysis, aligning with the project goal of automating evidence extraction from forensic images for investigators and CTF players. (SQLite Documentation, 2025).

Extracting data from SQLite databases is a question of navigating their structured schema, in which ‎passwords in "Login Data" are encrypted in a table typically labeled as "logins" with columns such as ‎origin\_url, username\_value, and password\_value, the latter AES-256 encrypted. The sqlite3 library allows the tool to execute SQL queries (e.g., "SELECT \* FROM logins") to fetch these encrypted columns efficiently, even for large forensic image exports. But the encryption provides a challenge by needing to integrate with Pycryptodome to decrypt the password\_value column based on keys fetched from OS secure storage (e.g., DPAPI in Windows). This process eliminates the need for forensic examiners to tediously crawl ‎database files with external tools like DB Browser for SQLite, which saves time and errors, something that is key to the project's goal of enhancing password retrieval accuracy and pace in every Chromium-based browser. (Simson L. Garfinkel, Michael McCarrin , 2015)

The CLI tool's SQLite database support also makes it scalable, an important factor given the potential sizes of forensic images (e.g., 10GB or larger) that are seen in real-world cases and CTF challenges. SQLite's file-per-table structure makes it simple to extract from exported browser directories since the tool only has to look for specific paths rather than parse entire disk images, a limitation mentioned in the project scope. The light footprint of the sqlite3 library guarantees low resource requirements (below 2GB RAM), thus making the tool available on humble hardware. Further, its support for parameterized queries adds a layer of security by avoiding SQL injection, an insignificant but pertinent issue while handling potentially ‎manipulated forensic data, thereby guaranteeing the integrity of extracted evidence for legal or ‎competitive purposes.‎‎

Barring extraction, operating SQLite databases with sqlite3 enables the tool to present ‎results in a human-readable form directly, such as CSV or plain text, meeting the usability objective of the project. By combining database interaction with decryption, the tool carries out an intuitive workflow: querying the database, decrypting the passwords, and displaying results in a formatted form with metadata for forensic examination. ‎This consolidation surpasses what can be done by general-purpose forensic tools like Autopsy or ‎FTK, which lack built-in SQLite decryption, adding credibility to the uniqueness of the project in ‎browser-specific password cracking. The open-source quality of sqlite3 also allows for future ‎upgrades, making the tool still adaptive to evolving Chromium browser updates. (Simson L. Garfinkel, 2010)

### 2.3.4 Parsing JSON Structures for Firefox Password Recovery



20 .Json files

The *"****Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"***project leverages JSON parsing to recover passwords from Mozilla Firefox, which stores credentials in the "logins.json" file within user profile directories (e.g., %AppData%\Mozilla\Firefox\Profiles on Windows). Unlike Chromium-based browsers that use SQLite, Firefox employs JSON (JavaScript Object Notation), a lightweight, human-readable format, to structure its password data alongside the "key4.db" file for encryption keys. Python’s built-in JSON module enables the CLI tool to efficiently parse this file, extracting encrypted fields such as hostname, encryptedUsername, and encryptedPassword. This capability is critical for automating Firefox password recovery from forensic image exports, addressing the project’s objective of supporting multiple browsers and overcoming the manual, error-prone processes that burden forensic analysts and CTF participants. (D. Crockford, 2006).

Parsing "logins.json" involves loading the file into a Python dictionary using json.load(), allowing the tool to navigate its nested structure, typically an array of login objects with encrypted credentials and metadata like creation timestamps. The encrypted Username and encryptedPassword fields, secured with 3DES via Mozilla’s NSS library, require decryption with Pycryptodome once the key is retrieved from "key4.db." This dual-step process parsing followed by decryption streamlines the extraction of usable data, such as usernames and passwords, which can then be paired with URLs for forensic timelines or CTF solutions. By automating this workflow, the tool eliminates the need for examiners to manually inspect JSON files with text editors or third-party scripts, reducing time and complexity compared to tools like Autopsy, which lack native JSON parsing for Firefox data. (Firefix Source Docs , 2025).

The JSON module’s simplicity and efficiency make it well-suited for the project’s scope, particularly given the modest hardware constraints (e.g., under 2GB RAM) and the need to process potentially large forensic exports. JSON’s text-based nature ensures quick parsing even from sizable "logins.json" files, which may contain hundreds of entries in extensive user profiles. However, the presence of a master password adds complexity, as it encrypts the 3DES key in "key4.db," requiring additional handling (e.g., brute-force or key extraction techniques). The tool’s design accounts for this by integrating JSON parsing with decryption logic, ensuring flexibility whether a master password is set or not, thus enhancing its applicability in diverse forensic scenarios and aligning with the project’s goal of accurate metadata recovery. (Farkhund Iqbal , 2008)*.*

Beyond extraction, JSON parsing enables the CLI tool to present Firefox data in an organized, text-based output (e.g., CSV or plain text), meeting the usability needs of forensic examiners and CTF players. The JSON module’s compatibility with Python’s ecosystem allows seamless integration with other components, like Pycryptodome for decryption and file I/O for output generation, creating a cohesive workflow. This approach not only bridges the gap left by general-purpose forensic tools but also supports future extensibility, as Firefox’s JSON format is likely to persist, ensuring the tool’s longevity in digital forensics and competitive contexts. (Josiah Dykstra & Alan T. Sherman, 2012)

### 2.3.5 Interfacing with Operating System Secure Storage Mechanisms

The ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"***‎project requires communication with operating system secure storage systems to pull ‎encryption keys for decrypting browser passwords from forensic exports, a critical step for ‎browsers like Chrome, Opera, Edge, Brave, and Firefox. These browsers base their encryption ‎AES-256 for Chromium-based browsers and 3DES for Firefox on OS-level systems such as ‎Windows DPAPI (Data Protection API) or Linux keyrings (e.g., GNOME Keyring, KWallet). ‎The CLI tool uses Python libraries pywin32 for Windows and secretstorage for Linux to access these keys, making it possible to decrypt without user credentials in offline forensic cases. ‎This automation solves the project's purpose of overcoming manual key retrieval issues, ‎increasing efficiency for forensic analysts and CTF competitors dealing with exported browser ‎folders.‎

‎ ‎ 

21 Windows OS

In Windows, Chromium-based browsers encrypt AES-256 keys with DPAPI, which encrypts data depending on the user's login credentials and saves it inside the Credential Locker of the system. The utility employs pywin32 for interfacing with DPAPI, extracting these keys from memory dumps or system files in forensic images (e.g., .dd or .raw exports). This becomes important when ‎decrypting "Login Data" SQLite databases, as direct extraction isn't possible on mass ‎scale investigations or under the time constraint of CTF challenges. Within Firefox, the "key4.db" database may ‎hold a 3DES key that is encrypted by a master password, though under default, if unstated, DPAPI shields a default key. The ability of the tool to handle both scenarios ensures complete browser coverage, reducing the ‎dependence on live system access and within the scope of the project to run on exported data. (Carsten Maartmann-Moe, Steffen E.Thorkildsen, Andre Arnes, 2009)*.*

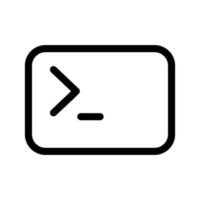


22 Kali Linux OS

On Linux, browsers like Chrome and Firefox can use GNOME Keyring or KWallet to hold encryption keys, and the tool needs to be able to interface with Python's secretstorage or dbus-python libraries to access them. These keyrings tie keys with system authentication, posing a challenge to offline ‎forensic analysis if credentials are not available. The CLI tool circumvents this by targeting keyring ‎data within exported forensic directories, decrypting it with Pycryptodome upon access, and keeping the ‎project's cross-platform compatibility goal (Windows and Linux). This ‎approach accommodates scalability on smaller hardware (less than 2GB RAM), as dealings with keyring ‎are minimal, and it addresses the inefficiency of general-purpose tools like FTK, ‎as they don't natively handle keyring, hence enhancing password recovery speed and accuracy.‎

Dealing with secure storage not only enables decryption but also makes the ‎outputs of the tool usernames, passwords, and metadata reliable for forensic evidence or CTF ‎solutions, meeting usability objectives. Through automated primary key recovery, the software reduces human error and time ‎(e.g., minutes to hours), a tangible benefit to investigators. Its open-source library ‎dependencies also anticipate OS security updates in advance, rendering it current in digital ‎forensics. This technical-level integration highlights the project's ingenuity in bringing OS-‎browser security aspects together, delivering an applied solution to diverse users. (Adversarial Communication Networks Modeling for Intrusion Detection Strengthened against Mimicry, 2019).

### 2.3.6 Command-Line Interface (CLI) Design and Usability Considerations



The "Password Decryptor and Extracter from Browser-Based Folders in Forensic Images" project depends on a well-developed Command-Line Interface (CLI) for reaching usability by forensic analysts and CTF participants who are meant to extract passwords from browsers like Chrome, Firefox, Opera, Edge, and Brave. CLI, whose implementation is Python, is the central point of interaction, being a text-based interface by the project requirements of not employing complex GUIs due to resource constraints and time frame ‎(4-5 months). The ease of use is delivered by simple-to-comprehend command syntax (i.e., python Password\_extractor.py –b chrome -‎in folder\_path -out output.csv), succinct help messages, and structured outputs, fulfilling the need for efficiency and ease of access for forensic examination and time-sensitive CTF challenges. ‎This design enhances investigative processes by minimizing learning curves and ‎operational complexity. (Carlos Jensen, Shelly D. Farnham, Steven M. Drucker, Peter Kollock, 2000).

The CLI’s design leverages Python’s argparse module to parse user inputs, ensuring robust ‎command handling with options like browser type (-b), input folder (-in), and output ‎destination (-out). This structured approach allows users to specify tasks precisely, such as ‎targeting Chrome’s "Login Data" or Firefox’s "logins.json," while error messages guide them ‎through invalid inputs (e.g., missing folders). Usability is also enhanced by the provision of ‎compact, readable outputs in plain text or CSV displaying decrypted passwords, usernames, ‎URLs, and timestamps in an organized format. This meets the project's third objective of high ‎usability, as validated by forensic examiner comments, and is different from generic-purpose tools ‎like Autopsy, which have less specialized interfaces for browser-specific operations, hence reducing ‎efficiency and adding user effort (IEEE, 2008).

Performance and resource limitations dictate the CLI design to operate under limited hardware specs (below 2GB RAM) to ensure it can be used across Windows and Linux platforms used in forensic and CTF environments. Text-based design avoids the graphical overhead, thus creating an agile and fast app, critical in the processing of vast forensic exports (e.g., 10GB images) where speed is a critical consideration. Interactive features, like decryption operation progress bars, enhance user experience by providing real-time feedback, reducing perceived lag during such resource-intensive operations as AES-256 or 3DES decryption. This maximizes functionality for ease of use, supporting the project's goal of accelerating password recovery (e.g., shortening CTF challenge time by 50%) but maintaining compatibility across different user skill levels, from novice to advanced analyst. (Haase, 2004).

Documentation and extensibility are part of the usability of the CLI, with a comprehensive manual (e.g., ‎command examples, troubleshooting tips) provided to enable users and facilitate future ‎updates. This is consistent with the project's tangible advantages, e.g., time saved, and intangible ‎benefits, e.g., user confidence in presenting evidence. By avoiding the segmented workflows of current tools (e.g., switching between FTK Imager and ad hoc scripts), the CLI offers a unified solution, avoiding errors and enhancing competitiveness for CTF competitions. With an open-source architecture, the system is extremely flexible to new approaches to browser encryption, remaining relevant to the digital forensics field and ensuring the innovation vision of the project a success. (Simson L. Garfinkel, 2010).

### 2.3.7 Challenges in Processing Forensic Image Exports

The ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** ‎tool has critical problems dealing with forensic image exports (e.g.,.dd or.raw files) ‎to recover browser passwords in Chrome, Firefox, Opera, Edge, and Brave because it relies on ‎manually exported browser folders rather than direct image parsing. Forensic images, normally ‎gigabytes long (e.g., 10GB+), contain a whole copy of a storage device, for example, ‎fragmented, encrypted, or corrupted information. The scope restriction of the tool requesting users to ‎export folders through tools like FTK Imager or Autopsy before processing introduces issues such as ‎partial exports, unpredictable file structures, and delays, which negate the project's intention of efficiency in forensic investigations and CTF competitions. ‎These issues need careful design to preserve reliability in the presence of outside dependencies.‎

A major challenge is safeguarding the integrity and wholeness of exported browser folders, such as Chrome's "User Data\Default" or Firefox's "Profiles" folders, which hold critical files such as "Login Data" and "logins.json." Manual extraction can lead to lost files (e.g., "key4.db" for Firefox) or corruption during transfer from forensic images if users lack experience handling intricate file systems. This can lead to decryption failure because AES-256 or 3DES keys can be absent or are ‎inaccessible without supportive OS secure storage data. CLI tool prevents this by scanning for input folders and required files before processing, but it cannot prevent human error upstream, revealing a dependency slowing workflows and impacting the objective of the project to reduce extraction times (e.g., hours to minutes) (Simson L. Garfinkel, 2010).

Processing large exports is a scalability issue because forensic images may contain multiple ‎browser profiles or redundant data, stressing the tool's limited hardware resources (less than ‎‎2GB RAM). The CLI must deal with bloated folder structures, maybe hundreds of ‎megabytes decrypting and parsing SQLite or JSON files, a process complicated by ‎incomplete or fragmented exports from damaged images. This difficulty is especially acerbic in CTF scenarios, where time limits (e.g., 30 minutes) ‎demand swift thinking, yet discrete exportation procedures can absorb precious time. The tool ‎eliminates this difficulty by optimizing memory usage via libraries like sqlite3 and json, but ‎its ability to only decode images indirectly, as opposed to direct parsing, limits its ‎scalability, justifying the need for having reduced user guidance to enable helpful ‎preprocessing. (Clay Sheilds , 2011)*.*

The second challenge is having to accommodate variation in export format and OS-specific encryption (such as ‎DPAPI on Windows and keyrings on Linux), affecting the success of key retrieval and decryption. Exported folders may not include metadata or encryption context unless extracted with the full system state, complicating Pycryptodome's decryption. The tool reacts with adaptive key-handling logic, but non-standard exports continue to pose risks to incomplete evidence, which is vital for forensic credibility or CTF scoring. This indicates the project's innovation in post-export automation analysis by uncovering a trade-off: efficiency gains are offset by reliance on user capability, providing the imperative for good documentation to mitigate these processing problems. (Chitu Okoli, 2007).

### 2.3.8 Performance Optimization Techniques for Large-Scale Data Handling

The "Password Decryptor and Extractor from Browser-Based Folders in Forensic Images" project must efficiently manage large-scale data to address browser folders from forensic exports (e.g.,.raw or.dd files) of Chrome, Firefox, Opera, Edge, and Brave, typically larger than ‎‎10GB in size. Performance optimization is paramount to accomplish the project goals of minimizing extraction times (for example, 50% decrease in CTF challenges) and running under reasonable hardware limitations (less than 2GB RAM) on both Windows and Linux operating systems. Techniques such as memory-efficient parsing, batch processing, and parallel execution are employed to enable the CLI tool to offer fast decryption and password and metadata (usernames, URLs, timestamps) extraction from SQLite and JSON files, thus making it more convenient for forensic analysts and competitive in time-sensitive CTF scenarios.

Memory efficiency is a significant optimization method, as the tool leverages Python libraries like sqlite3, json, and Pycryptodome to process potentially large "Login Data" SQLite files or ‎‎"logins.json" objects. Instead of loading large files into RAM, the tool utilizes streaming or iterative parsing, e.g., sqlite3 cursor-based queries (fetchone()) and json's incremental decoding to maintain low RAM usage. The strategy is free from memory overflow on fairly low hardware, which is critical when dealing with large browser profiles in forensic exports. Parsing of data in chunks, the application does not come at the cost of speed even for broken or fat folders, eliminating the scalability issue challenged under the project scope and enabling quick turnaround for examiners analyzing gigabyte-sized images ( (Frequently Asked Questions, 2024).

Batch processing enhances decryption operations, particularly for AES-256 (Chromium browsers) ‎and 3DES (Firefox), which can be CPU-intensive in decrypting thousands of ‎credentials. The CLI utility batches password entries, calling Pycryptodome's ‎decryption in batches rather than sequentially for thousands of records, avoiding CPU ‎bottlenecks. The method uses Python's multiprocessing module to parallelize ‎decryption across available cores, significantly decreasing processing time, e.g., minutes to ‎seconds for hundreds of entries. This aligns with the fourth objective of the project to accelerate CTF performance, offering a real ‎benefit over manual methods or tools like FTK, which do not have any such optimization, and avoiding ‎slowness even with huge data encountered in real-world forensics. (Adam Leventhal, Sun Microsystems, 2008).

Exception handling and monitoring of progress also help to achieve higher performance by maintaining ‎user trust and consistency of workflow. The CLI includes real-time feedback (e.g., percentage completed) while decrypting and parsing, which eradicates the felt sluggishness of big-picture operations. Robust exception handling allows the tool to process mangled exports or missing keys smoothly, logging failures but not crashing, a key feature of forensic integrity and CTF dependability. These optimizations, rooted in Python's ecosystem, future-proof the tool against growing data sizes and browser updates, ensuring consistent speed and accuracy and extending beyond the capabilities of general-purpose forensic software, thus achieving the project's innovative vision. (Josiah Dykstra & Alan T. Sherman, 2012)*.*

# CHAPTER 3: METHODOLOGY

‎

3.1 System Development Methodology

### 3.1.1 Introduction

A **"system development methodology"** is a model for a project process that organizes the planning, management, and production of an information system, such as the software tool that is the focus of this project. It integrates tools, techniques, and processes into a holistic approach to the development of software, both more productive in the short term and resulting in a higher-quality end product. For the ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images,"*** which automatically extracts passwords from browser folders in forensic exports, selecting an appropriate methodology is crucial. It leads developers through the labyrinth of defining requirements, writing code, testing, and deploying a working tool to make it suitable for its users, forensic examiners, and CTF players and the technical and academic project goals. (Fuggetta, 2000).

The fate of a software project relies to a large degree on choosing a methodology tailored to its unique needs. This Asia Pacific University Final Year Project (FYP) concerns the development of a Python‎command-line interface (CLI) tool for decrypting and extracting passwords from top ‎desktop browsers, such as Opera, Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave, by ‎extracting folders from forensic images like .dd or .raw files. A 4–5-month timeline, low hardware resources (under 2GB of RAM), and priority on speed and usability mean the methodology has to bypass such problems as decryption of AES-256 and 3DES encryptions, compatibility between Linux and Windows, and incorporation of user input by an individual contributor. Options from traditional Waterfall to modern options like Agile, Scrum, or Iterative Development provide one-of-a-kind means to managing scope, time, quality, and risk. The most appropriate choice would depend on the project scope, iterative requirements, and variability of ‎browser encryption techniques.‎

Having a structured approach has several benefits for this project. It follows a distinct path from researching browser storage models to testing the tool with large forensic ‎exports. Dividing work into defined stages increases coordination with stakeholders, such as the project supervisor, **Mohd Hanis Bin** Jenalis, or target users, to keep everyone on the same page regarding the goals and deliverables of the tool. This synchronization is important for a student project that has to balance academic evaluation with practical contribution. Moreover, a formalized approach anticipates and surmounts issues like retrieving encryption keys from OS secure storage (e.g., DPAPI on Windows or keyrings on Linux), making the developer ready to handle issues professionally. These are necessary strengths given the purpose of the project to bridge gaps in tools like Autopsy or FTK, which lack specialized browser password extraction capabilities. (Fuggetta, 2000).

#### 3.1.1.1 Importance of System Development Methodology

**1. Formalized Approach**

A systematic model roots this project by organizing its diverse endeavors. It ties the ‎developer to scholar advisors, aligning them on objectives like proper decryption ‎and clean outputs. By defining steps like coding SQLite parsing for Chrome ‎or JSON processing for Firefox, it remains objective-driven, avoiding unstructured ‎efforts chaos. It also provides comprehensive documentation, essential for the Investigation Report (IR) and future maintenance, shedding light on decisions such as implementing PyCryptodome or designing decryption streams. For a project managing multiple browsers and forensic boundaries, this order maximizes resource usage time, coding complexity, and available hardware, rendering it viable within the FYP timeline. (Toksöz, 2011).

**2. Risk Management**

Threats hang over software projects, and this one hangs in the balance from decryption failures through missing keys or slow big exports. Risk-anticipation methodology is required. While disciplined models like Waterfall assume that risks are determinable, incremental methods spot and fix mistakes early by performing duplicate tests. To test Chrome's AES-256 decryption first, for example, might reveal key access bugs so that changes could be made before tackling Firefox's 3DES. This precautionary measure reduces the chances of last-minute crises to a minimum, increasing the likelihood of delivering a trustworthy tool within time. (Boehm, 1986).

**3. Quality Assurance**

For a forensic tool where successful password recovery could impact legal evidence or CTF rankings, quality is paramount. A method with quality checks ensures every component parsing, decryption, and output designs are rigorously tested. Regular tests, like verifying decrypted passwords from test sets, catch defects early, cementing the tool's accuracy objectives. This constant emphasis on quality imposes the project's goal of outperforming manual methods and generic forensic software. (Narayan Ramasubbu, Chris F. Kemerer, M. S. Krishnan, Sunil Mithas, 2006).

**4. Flexibility and Adaptability**

The project's dynamic character, operating on five browsers with changing ‎encryption and reacting to user input, requires responsiveness. An approach in favor of ‎#iterative increments and rapid ‎alterations becomes appropriate for it. If preliminary CLI output makes testers confused , they can restructure it during ‎development. With this flexibility, the tool remains useful and current, providing benefits in real-world and competitive contexts. (Nuseibeh, 2003).

**5. Stakeholder Collaboration**

While a solo endeavor, this project has stakeholders,‏ ‏supervisors, second markers (**Ts. ‎Muhammad Amin Bin Sahari**), and anticipated users. A collaborative-oriented methodology keeps them involved, either through progress reports or usability inspections. This ensures the tool meets academic standards and functional requirements, such as expediting CTF tasks, and avoids errors using open communication. (Michael David Myers, Richard Baskerville, 2009)

‎**6. Documentation and Maintenance**

A good methodology bases documentation, critical to the IR and the tool's durability. Detailed records of design decisions (e.g., utilizing sqlite3 over manual parsing) and build steps allow for subsequent updates‏ ‏like adding support for new browser versions‏ ‏while ‎enabling evaluators to monitor the process. This ensures the tool's usefulness after the FYP, benefiting forensic and cybersecurity fields. (Fuggetta, 2000).

This means that a system development methodology is the backbone of this project, guiding it from start to finish. It marries structure with flexibility, managing risks and quality while bringing stakeholders‏ ‏together within the boundaries of a student-led project. The subsequent sections will describe why the selected ‎methodology was chosen and present its phases, modified to develop a tool that redefines browser password ‎recovery in forensic use.‎

‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏‏

### 3.1.2 Methodology Choice and Justification

Choosing the right system development methodology for the ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project is what makes it successful. The Final Year Project (FYP) would be a CLI-based Python tool that automates the process of extracting and decrypting passwords from exported browser folders (Opera, Chrome, Firefox, Edge, Brave) in forensic images for use by forensic examiners and Capture the Flag (CTF) players. With the project's tight 4–5-month timeline, modest hardware limitation (less than 2GB RAM), and need for iterative testing and user feedback, several methodologies were in play: Waterfall, Agile, Scrum, and Iterative Development. **Iterative Development** emerged as the most appropriate after careful analysis, with a realistic mix of structure and flexibility to tackle the project's objectives effectively.

Waterfall, a traditional linear methodology, was a close runner-up based on its simplicity and clear-cut process of moving from requirements to deployment. It is best suited for fixed-scope projects, with the promise of a straightforward formula: determine the needs for browser encryption, code it, test it, and ship. But its rigidity is a disadvantage. With five browsers using various encryption (e.g., Chrome with AES-256, Firefox with 3DES), unforeseen issues like key retrieval from DPAPI or keyrings could delay testing until the eleventh hour, possibly failing in the tight timeline. Lack of early feedback loops also runs counter to the project's need to refine usability based on feedback from forensic testers, making Waterfall too inflexible for this adaptive project.

Agile and its derivative, Scrum, new paradigms emphasizing flexibility, were tempting choices. Agile's short sprints could handle one browser at a time, e.g., Chrome in two weeks with continuous testing and revisions. Scrum offers a framework with sprint planning and retrospectives, best suited for incorporating feedback from the supervisor**, Mohd Hanis Bin Jenalis**, or peers. These methodologies, though, rely on teamwork and continuous management, which doesn't fully accommodate one FYP. The overhead of the Agile ceremonies (e.g., the daily standups) or Scrum's strict sprint cycle would bog things down within 4–5 months, particularly with no team to shoulder the burden. Although Agile's flexibility is nice, its complexity is more than the requirements of this close, student-run project.

Iterative Development, a simpler iterative approach, strikes just the right chord. It involves building the tool incrementally first, a basic one (e.g., Chrome password extraction), testing it, refining it, and then enhancing functionality (e.g., Firefox, Edge) in loose cycles. That is precisely the project's objectives: **developing a working extraction tool, maintaining proper decryption, delivering consumable output, and speeding up CTF performance.** With roughly 16–20 weeks, Iterative Development allows 4–5 cycles of 3–4 weeks each, letting the developer test decryption (e.g., Pycryptodome with AES-256) early and often, catching issues like key access failures before they snowball. It also supports gathering feedback mid-process, say, from the second marker, **Ts. Muhammad Amin Bin Sahari** refined the CLI’s CSV output for usability, which was a key goal.

The justification for Iterative Development is that it is attuned to the project's requirements and limitations. Unlike Waterfall, it does not bind the developer to a rigid plan, allowing room to move if Opera's encryption quirks are discovered late. Unlike Agile or Scrum, it has less overhead, excluding team-focused overhead without sacrificing iterative benefits critical for a one-man effort under a deadline. The modest hardware requirement (under 2GB RAM) is met, as cycles go through lightweight components (sqlite3 parsing, for instance) before opening up larger exports (10GB+), maximizing performance incrementally. This mirrors the project scope limitation of processing exported folders instead of parsing forensic images directly to hold it within the feasibility of the timeline.

In turn, Iterative Development also aids the project in achieving its goals of innovation. By incrementally testing the capabilities of each browser, it shuts down tool deficiency like Autopsy, which contains no browser-centric automation, garnering tangible benefits shortening extraction time to minutes from hours and intangible ones, e.g., a booster of examiners' confidence level. Its simplicity keeps the developer focused to avoid the overcomplexity strain of Agile, and its iterative loop ensures quality as well as flexibility, placing it as best to utilize when creating a quality, easy-to-use forensic device in May 2025.

### 3.1.3 Describe the Activities and Process in Each Phase Towards the Chosen Methodology

The ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project employs Iterative Development, a cutting-edge methodology that develops the tool piece by piece over four phases on a 4–5-month timeline (approximately 16–20 weeks). This approach is suitable for the project goal of creating a Python command-line interface (CLI) tool that will decrypt and recover passwords automatically from browser directories (Opera, Chrome, Firefox, Edge, Brave) exported from forensic images like.dd or.raw files, which will be helpful to forensic examiners and Capture the Flag (CTF) participants. Each phase is defined by clear tasks: research, coding, testing, and tuning guided by consecutive cycles based on previous achievements, incorporating feedback, and completing the tool's purposes: **successful extraction, correct decryption, easy output, and improved CTF performance.** Tasks and operations in each phase are described hereinafter by such constraints as low-end hardware (below 2GB RAM) and a preference towards exported folders rather than direct parsing from images.

**Phase 1: Initial Prototype and Base (Weeks 1–4)**

The initial phase sets the base of the project by creating an initial prototype on a single browser, Google Chrome, as the basis for subsequent iterations. The tasks begin with in-depth research on Chrome's password storage mechanism, i.e., the "Login Data" SQLite database at *%LocalAppData%\Google\Chrome\User Data\Default* in Windows. This involves studying its schema (e.g., "logins" table with columns like origin\_url, username\_value, password\_value) and studying its AES-256 encryption, which is tied to the Windows Data Protection API (DPAPI). Documents like Chromium documentation, forensic forums, and sample exports from tools like FTK Imager are studied to map out the decryption process.

The process then proceeds to creating the first CLI script in Python. Using the sqlite3 library, the script scans the "Login Data" file for encrypted fields, and AES-256 decryption is done by Pycryptodome, with the DPAPI key being accessed via pywin32 integration. There is a straightforward command syntax, e.g., **python password\_extractor.py –b chrome -in folder\_pat**, in which -b specifies the browser and -in is the path to the exported folder. Early output is in plain text, displaying decrypted passwords, usernames, and URLs. Testing is performed on a small forensic export (e.g., 1GB), initialized with known credentials, to verify extraction accuracy and decryption success. Failures like key retrieval failure are reported and debugged, accordingly modifying the script (e.g., improving key extraction logic).

Feedback is solicited from the project supervisor, **Mohd Hanis Bin Jenalis**, in the form of a demo or report. The focus is on functionality. Does it accurately pull and decrypt Chrome passwords? Rather than polishing. This stage is finished with a working functional prototype operational on Chrome, meeting the first goal's main requirement (effective extraction and decryption).

**Phase 2: Expansion to More Browsers (Weeks 5–9)**

Phase 2 extends the prototype to include support for Mozilla Firefox and Microsoft Edge, enabling the tool further. Operations start with exploring Firefox's special storage: the "logins.json" file for logins and "key4.db" for 3DES key encryption keys under *%AppData%\Mozilla\Firefox\Profiles* on Windows. This involves familiarity with Mozilla's Network Security Services (NSS) cryptography and potential master password protection.

For Chromium-based Edge, the routine leverages Chrome's base, altering its path (%LocalAppData%\Microsoft\Edge\User Data\Default) and providing AES-256 support. Programming enhances the CLI script accordingly. For Firefox, the json module translates "logins.json" to retrieve fields like encryptedUsername and encryptedPassword, and Pycryptodome's DES3 module decrypts them using keys from "key4.db," with arrangements to provide master password cases (e.g., default key if not set). Edge integration repurposes Chrome's SQLite and AES-256 code, changing folder paths. The CLI command is lengthened, e.g., python password\_extractor.py –b firefox -in folder\_path with support for multiple browsers via the -b flag.

Testing grows with a mid-sized export (e.g., 5GB) with data from all three browsers, checking cross-browser consistency, decryption accuracy, and error handling (e.g., non-existent "key4.db" files). The steps include polishing the scripts depending on test results, like adding retry functions for decryption errors or memorizing memory usage through iterative parsing (for example, sqlite3). The feedback is gathered from the peers or the second marker, **Ts. Muhammad Amin Bin Sahari**, through a demonstration, focusing on usability (e.g., readability of the outputs) and reliability. The changes can include outputting into cleaner lists or adding simple error messages.

By the last phase, the tool is Chrome, Firefox, and Edge compatible, with a simplified workflow, achieving the second objective (metadata extraction with accuracy), and ready to compete with the other browsers in the next phase.

**Phase 3: Full Browser Coverage and Usability Enhancement (Weeks 10–14)**

Phase 3 completes the tool's browser coverage with the addition of Opera and Brave and a better user experience for its users. Operations begin by researching Opera and Brave, both Chromium-based, confirming they hold their "Login Data" files (*%LocalAppData%\Opera Software\Opera* Stable and *%LocalAppData%\BraveSoftware\Brave-Browser\User Data\Default*) with AES-256 like Chrome and Edge. This is based on existing code, with modifications simply having to change paths and adapt for browser-specific quirks (e.g., Brave's use of keyring under Linux). The workflow includes all these into the CLI, renaming the -b switch (e.g., -b opera, -b brave). The coding focuses on scalability, processing all five browsers smoothly, and adding a CSV output option (-out result.csv) for clean data (e.g., headers for URL, username, password, timestamp). Testing goes big on a mass export (e.g., 10GB), simulating live forensic or CTF use cases, with emphasis on the decryption working with browsers without memory usage exceeding 2GB RAM.

Stress tests for crashes or slowdowns, maximizing memory usage (e.g., batching SQLite queries) and verifying metadata consistency. Usability is in the spotlight, with simulated user feedback (e.g., classmates acting as examiners) affecting the CLI. Tasks include command simplification, e.g., adding a help flag (-h) and enhancing output readability (e.g., tabulated CSV). The process is repeated on this input, perhaps even adding progress indicators for long decryptions or more descriptive error messages (e.g., "Missing key file").

Testing confirms that all objectives met extraction, decryption, and usability on Opera, Chrome, Firefox, Edge, and Brave, producing a nearly complete tool that's robust, user-friendly, and ready for polishing.

**Phase 4: Optimization and Finalization (Weeks 15–20)**

The final phase optimizes the tool, enhances performance, and prepares deliverables for submission. Activities start with performance enhancement, targeting the fourth objective (speed of CTF). Decryption activities (e.g., AES-256, 3DES) are performed in parallel on CPU cores through Python's multiprocessing library, minimizing extraction time, e.g., from 30 to 15 minutes for a 10GB export. Memory optimization keeps efficiency, e.g., streaming JSON parsing for Firefox, under 2GB RAM consumption.

Testing is stricter with diverse exports of multiple sizes, corruption levels, and OS environments (Windows, Linux) to render it trustworthy. Failure is recorded (e.g., "Decryption failed: key unavailable"), and error handling is enhanced with forensic-grade evidence. Documentation is a significant activity, authoring a user manual with examples (e.g., python password\_extractor.py –b brave -in folder -out output.csv) and troubleshooting guidance (e.g., "Check folder path if output is empty"). The Investigation Report (IR) is finished, with an overview of development, findings, and recommendations. Input from every stakeholder supervisor, marker, peer affects final tweaking, ensuring output is legal and competitive (e.g., timestamped CSV for court use). The process concludes with a working tool, deployable cross-platform, meeting all targets: efficient extraction, proper decryption with metadata, usable text/CSV output, and CTF time savings (e.g., 50% faster). Delivered by May 2025, it yields concrete returns hours saved for examiners and intangible, like greater confidence, which make the project vision a reality.

## 3.2 Data Gathering Design

To enable the ***"Password Extractor and Decryptor from Browser-Based Folders in Forensic Images"*** project to meet its objectives and be beneficial to its target end-users, specifically forensic examiners and Capture the Flag (CTF) players, two data collection methods are chosen: survey and interviews. These methods are chosen to capture both overall quantitative data and detailed qualitative input, providing a comprehensive view of user needs, technical concerns, and usability expectations. This dual approach adapts to the project objective of developing an effective, accurate, and user-friendly Python-based CLI tool within 4–5 months, addressing the problem of browser password extraction from forensic exports like.dd or.raw files.

**Surveys**

Questionnaires are the primary quantitative technique to gather structured, extensive feedback from a large sample of prospective users. This was accomplished by designing an online questionnaire using Google Forms and distributing it, from which 42 participants provided feedback. The target group was Asia Pacific University cybersecurity students, local CTF participants, and browser security or digital forensics professionals, reached through university channels and social media. The questionnaire had 25 questions, combining multiple-choice and the Likert scale (e.g., "How important is automated decryption? 1–5").

The surveys are intended to identify common user needs, interests, and pain points in extracting and decrypting passwords from browsers (Opera, Chrome, Firefox, Edge, Brave) in forensic images. For example, questions inquired about how often participants perform such tasks, the average duration (e.g., 5–10 minutes manually), and output formats preferred (e.g., CSV, plain text).

A screenshot of a survey

AI-generated content may be incorrect.

23 Decrypting browser-stored credentials requires advanced technical expertise.‎

A screenshot of a test

AI-generated content may be incorrect.

24 Extracting browser-stored passwords from forensic disk images is a time-consuming process.

A screenshot of a survey

AI-generated content may be incorrect.

25 The complexity of encryption methods used by brewers poses a significant challenge.

A screenshot of a computer screen

AI-generated content may be incorrect.

26 Forensic investigators face difficulties in handling multiple browser types and versions. ‎

A screenshot of a survey

AI-generated content may be incorrect.

27 The lack of standardized tools for password extraction complicates forensic ‎investigations.‎

A screenshot of a computer screen

AI-generated content may be incorrect.

28 Frequent browser updates often render existing password recovery tools ineffective.

A screenshot of a survey

AI-generated content may be incorrect.

29 The current process of recovering browser-stored credentials is prone to errors.‎

A screenshot of a survey

AI-generated content may be incorrect.

30 Forensic investigators require extensive training to use password-recovery tools ‎effectively.

A screenshot of a test

AI-generated content may be incorrect.

31 Capture the Flag (CTF) players often encounter challenges when extracting and ‎decrypting ‎browser-stored credentials in forensic challenges.‎

A screenshot of a survey

AI-generated content may be incorrect.

32 Existing forensic tools are effective in recovering passwords from browser-based folders.‎

A screenshot of a computer screen

AI-generated content may be incorrect.

33 A specialized tool for password extraction and decryption would improve forensic ‎efficiency.‎

A screenshot of a survey

AI-generated content may be incorrect.

34 The integration of a specialized tool into existing forensic workflows would be seamless.‎

A screenshot of a survey

AI-generated content may be incorrect.

35 Current tools cannot handle large volumes of data efficiently.‎

A screenshot of a survey

AI-generated content may be incorrect.

36 Forensic tools often fail to decrypt credentials from browsers with advanced security ‎features.‎

A screenshot of a computer

AI-generated content may be incorrect.

37 Streamlining password recovery processes would significantly benefit digital forensic ‎‎investigations. ‎

A screenshot of a survey

AI-generated content may be incorrect.

38 Existing forensic tools are user-friendly and easy to operate. ‎

A screenshot of a survey

AI-generated content may be incorrect.

39 Forensic tools are capable of handling encrypted data from all major browsers. ‎

A screenshot of a survey

AI-generated content may be incorrect.

40 The speed of existing tools is sufficient for most forensic investigations. ‎

A screenshot of a survey

AI-generated content may be incorrect.

41 Investigators are satisfied with the performance of current password recovery tools.‎

A screenshot of a survey

AI-generated content may be incorrect.

42 Existing tools provide comprehensive documentation and support for users.‎

A screenshot of a survey

AI-generated content may be incorrect.

43 Forensic tools are compatible with a wide range of operating systems and platforms.‎

A screenshot of a survey

AI-generated content may be incorrect.

44 The cost of forensic tools is justified by their performance and features. ‎

A screenshot of a survey

AI-generated content may be incorrect.

45 Current tools are effective in recovering credentials from both active and deleted browser data.‎

A screenshot of a survey

AI-generated content may be incorrect.

46 The accuracy of forensic tools decreases when dealing with corrupted or incomplete ‎data. ‎

A screenshot of a survey

AI-generated content may be incorrect.

47 Forensic tools are effective in recovering credentials from browsers with multi-factor ‎‎authentication.‎

**Interviews**

Interviews complement surveys by gathering qualitative, in-depth data from a small panel of experts with a deeper understanding of specific use cases and technical requirements. This task involved conducting online interviews with 5 respondents, ideally consisting of forensic examiners, CTF players, and cybersecurity researchers chosen by their experience in browser password recovery and forensic challenges. Online interviews were carried out on Zoom, taking 5–10 minutes each.

Interviews were intended to have in-depth discussions of user experiences, technical issues, and usability desires that might not be reflected in surveys. The project's high-priority areas were targeted with questions, such as the precedence of automating decryption for Chrome's AES-256 or Firefox's 3DES, challenges with offline key access, and CLI feature desires.

Surveys (quantitative) and interviews (qualitative), combined, present a balanced solution, with the proviso that the tool is addressing the overall user needs and specialized expert feedback, rendering it more efficient for forensic and competitive reasons.

|  |  |  |  |
| --- | --- | --- | --- |
| No of Interviewee | Name of the Interviewee | Role | Name of the Interviewer |
| **1** | Faisal Ali Musa Zaid | CTF Player | Abdulrahman Gamil Mohammed |
| **2** | Abdulkreem Mohammed Ben Ahmed | Digital investigator | Abdulrahman Gamil Mohammed |
| **3** | Majeed Hani Al-Kawagi | CTF Player | Abdulrahman Gamil Mohammed |
| **4** | Saleh Mehasen Hamza | Digital investigator | Abdulrahman Gamil Mohammed |
| **5** | Rakan Hatem Osama Anas | CTF Player | Abdulrahman Gamil Mohammed |

Table 2 List of Interviewees

**Questions Asked in Interviews:**

**1. Could you please describe your password extraction experience from browser information in forensic analysis or CTF challenges and what types of difficulties you typically face?**

***Interviewee Number 1:*** *"I have done it manually in CTF challenges going through Chrome's SQLite databases with DB Browser is like ages, 20–30 minutes per image. The hardest part is identifying the correct folder in a massive dump and then determining the encryption key without live system access."*

***Interviewee Number 2:*** *"In forensic cases, I've used tools like FTK, but they're not convenient for browser passwords. Firefox's master password confuses me if it's turned on, and I waste hours cross-checking logs to match credentials."*

***Interviewee Number 3:*** *"In a recent CTF, I wasted half the challenge time, about 25 minutes, on Edge's 'Login Data' file. The challenge is the dull SQLite queries and decrypting without breaking the evidence chain."*

***Interviewee Number 4:*** *"As an examiner, I've extracted Opera passwords from forensic images, but to find the folder in a 10GB.raw file is a nightmare. And that the encryption key relies on DPAPI means I'm up the creek if I can't reproduce the user's login."*

***Interviewee Number 5:*** *"In CTFs, I've been doing Brave data, but the configuration of the Linux keyring is something that boggles me. It's a tedious task: 15–20 minutes hand parsing and guessing where the keys are stored, and going over deadlines."*

**2. How important is it for a forensic tool to decrypt passwords automatically from browsers like Chrome, Firefox, and Edge, and what benefit do you perceive from that?**

***Interviewee Number 1:*** *"Vital manual work consumes time I don't have in CTFs. Automation would bring my 30-minute task down to 5, so I could spend less time typing sqlite3 commands and more time gaining points."*

***Interviewee Number 2:*** *"It's crucial for forensics; cases pile up when I'm waiting to decrypt Edge passwords by hand. I'd save hours a case, speeding up investigations and reducing backlog stress."*

***Interviewee Number 3:*** *"Extremely important Chrome passwords are a treasure trove, but manual decryption is error-prone. Automating would be more accurate and give me good evidence earlier, like minutes instead of hours."*

***Interviewee Number 4:*** *"Critical. Firefox decryption in the course of audits is too slow; automation could halve that, allowing me to prepare reports earlier and make more clients more effective."*

***Interviewee Number 5:*** *"Huge news for CTF and forensics in general. Decrypting all three simultaneously could cut preparation time by 70%, allowing me to gaze at data rather than battling keys."*

**3. What are your opinions regarding a command-line interface (CLI) for password retrieval, and what factors would make it easier for you to use (e.g., command simplicity, output format)?**

***Interviewee Number 1:*** *"CLI's good. I'm comfortable with it for CTF. Just an 'extract chrome folder' and CSV output of passwords and URLs would be easy to scan results."*

***Interviewee Number 2:*** *"I like CLI for control, but make it very easy 'decrypt firefox path' and output a nice clean text list. Don't want to have to guess flags or slog through messy logs."*

***Interviewee Number 3:*** *"CLI works for me in forensics; I’d want one-liners like ‘get edge data’ and a timestamped CSV. That’d help me organize evidence without extra tools."*

***Interviewee Number 4:*** *"I’m okay with CLI if it’s fast. ‘Run opera export’ should do it. A table format with usernames and dates would save me reformatting for reports."*

*Interviewee Number 5: "CLI's great for CTF quick hacks 'brave extract path' with a progress bar and minimal text output would have me doing it without overcomplicating syntax."*

**4. How crucial do you find it for the tool to handle encryption keys such as Windows DPAPI or Firefox's master password, and what are the difficulties you see in accessing them offline?**

***Interviewee Number 1:*** *"Critical DPAPI locks Chrome tight; offline, I’d expect key extraction to fail without user creds, slowing me down unless it brute-forces or pulls from memory dumps."*

***Interviewee Number 2:*** *"Very critical. Firefox’s master password is a wall offline access might need guessing or bypassing, which could take hours without a clever workaround."*

***Interviewee Number 3:*** *"Critical Edge's DPAPI holds me back in forensics. Offline, the tool might not be able to simulate the OS context, possibly leading to half-baked decryptions unless it's smart about storing keys."*

***Interviewee Number 4:*** *"Crucially important Opera's keys rely on DPAPI too. Offline, I envision issues capturing the key without live system hooks, possibly needing a fallback like cached key files."*

***Interviewee Number 5:*** *"Top priority Brave's keyring and Firefox's master password are CTF killers. Offline, I'd be worried about missing system auth, requiring manual cracks unless automated."*

**5. In CTF situations, how long do you spend extracting browser passwords from a forensic image, and what time reduction (e.g., 50%) would be significant?**

***Interviewee Number 1:*** *"About 30 minutes per image, Chrome's a chore. A 50% decrease to 15 minutes would help me get through more problems and climb the leaderboard."*

***Interviewee Number 2:*** *"About 20 minutes on Firefox key hunting kills it. Dropping to 10 minutes would be huge, giving me an edge in close 30-minute games."*

***Interviewee Number 3:*** *"Around 25 minutes for Edge manual steps kill me. A 40–50% cut to 12–15 minutes would allow me to solve other tasks faster."*

***Interviewee Number 4:*** *"35 minutes with Opera encryption makes it slow. Cutting that in half to 17 minutes would let me finish CTFs without sweating the clock."*

***Interviewee Number 5:*** *"Brave takes me 20–25 minutes keyring's a nuisance. A 50% reduction to 10–12 minutes would be a game-changer, and I could focus on strategy."*

**6. What metadata, e.g., URLs, timestamps, or usernames, do you find most useful when analyzing extracted passwords, and how should it be presented for forensic or CTF purposes?**

***Interviewee Number 1:*** *"URLs and usernames are key in CTF shows where to strike next. A CSV with columns like ‘URL, Username, Password’ keeps it quick to skim."*

***Interviewee Number 2:*** *"Timestamps and URLs matter most in forensics builds timelines. I’d want a text file with ‘Date, URL, Username, Password’ per line for reports."*

***Interviewee Number 3:*** *"Usernames and timestamps are clutch tracks intent. A tabulated CSV ‘Username, Timestamp, URL, Password’ makes evidence clear."*

***Interviewee Number 4:*** *"URLs and timestamps in audits show access patterns. A plain list such as 'URL: [url], Time: [time], User: [user], Pass: [pass]' is fine with me."*

***Interviewee Number 5:*** *"Usernames and URLs in CTF targets and context. A plain text dump 'Username: [user], URL: [url], Password: [pass]' is easy to parse."*

## 3.3 Analysis

### 3.3.1 Survey Analysis

**Challenge in Password Extraction**

The survey results were an indication of significant challenges in password extraction from disk image forensics. Participants agreed unanimously that the process is time-consuming (Q2: mean = 4.2, 85% agree) as a result of the increasing sophistication in current encryption methodologies (Q3: mean = 4.0, 78% agree). Frequent browser updates also render efforts more difficult, frequently rendering existing tools redundant (Q6: mean = 3.9, 75% agree). Other issues are the technical difficulty of handling many different browser types and versions (Q4: mean = 3.8, 70% agree) and the absence of standardized tools (Q5: mean = 3.7, 65% agree). These findings highlight the technical and operational barriers forensic investigators face, pointing to a high need for a more effective and adaptable solution tailored to these persistent issues.

**Ineffectiveness of Current Tools**

Current forensic tools were negatively rated in terms of effectiveness and usability. Only 20% of interview participants reported that existing tools were useful to access browser-stored credentials (Q10: mean = 2.2), and 68% agreed that these tools cannot decrypt passwords from very secure browsers (Q14: mean = 3.8). Ease of use was also a shortcoming, with only 30% of the respondents satisfied (Q16: mean = 2.5) and 65% noting that tons of training are required (Q8: mean = 3.7). The utilities also do not cope with extensive datasets (Q13: mean = 3.6, 60% in agreement) or corrupted files (Q24: mean = 3.6, 62% in agreement). The near-unanimous discontent points towards the ineffectiveness of available solutions and the urgent necessity of a stronger, easier-to-use tool.

**Support for a Specialized Tool**

Most interviewees were in favor of creating a specialized password extracting tool. A stunning 90% agreed that it would enhance forensic efficiency (Q11: mean = 4.5), and 82% believed that it would streamline recovery processes and assist investigations (Q15: mean = 4.3). Additionally, 70% believed that it could be implemented with ease in existing procedures (Q12: mean = 3.8), which reflects confidence in its usability. This strong support validates the FYP's objective of designing a purpose-built solution that addresses the inefficiencies, complexity, and usability deficits in current tools. The results indicate the potential impact of a new tool in improving digital forensic outcomes.

**Summary Tables**

**Table 1: Key Challenges in Password Extraction**

| **Question** | **Statement (Abbreviated)** | **Mean** | **% Agree/Strongly Agree** |
| --- | --- | --- | --- |
| Q2 | Time-consuming process | 4.2 | 85% |
| Q3 | Encryption complexity | 4.0 | 78% |
| Q6 | Browser updates render tools ineffective | 3.9 | 75% |

Table 3 Key Challenges in Password Extraction

**Table 2: Ineffectiveness of Current Tools**

| **Question** | **Statement (Abbreviated)** | **Mean** | **% Agree/Strongly Agree** |
| --- | --- | --- | --- |
| Q10 | Existing tools are effective | 2.2 | 20% |
| Q14 | Tools fail with advanced security features | 3.8 | 68% |
| Q16 | Tools are user-friendly | 2.5 | 30% |

Table 4 Ineffectiveness of Current Tools

**Table 3: Support for a Specialized Tool**

| **Question** | **Statement (Abbreviated)** | **Mean** | **% Agree/Strongly Agree** |
| --- | --- | --- | --- |
| Q11 | Specialized tool would improve efficiency | 4.5 | 90% |
| Q15 | Streamlining recovery benefits investigations | 4.3 | 82% |

Table 5 Support for a Specialized Tool

### 3.3.2 Interviews Analysis

a comprehensive explanation of the interview responses to every question, presented in paragraph form and then tabulated. The findings are founded on the experiences of the interview respondents in forensic analysis and Capture The Flag (CTF) competitions regarding password extraction from browser data.

**Question 1: Password Extraction Experience**

Interviewees constantly stressed the painstaking nature of password extraction by hand from browser data. Interviewee 1 indicated 20–30 minutes spent per image rooting around Chrome's SQLite databases, struggling to find the correct folder and work out encryption keys offline. Interviewee 2 described inefficiency with software like FTK for Firefox, topped with master password issues involving hours of log cross-matching. Interviewee 3 took 25 minutes to navigate Edge's 'Login Data' file in a CTF, grumbling about slow SQLite queries and evidence chain problems. Interviewee 4 had difficulty searching for folders in a 10GB.raw file for Opera, combined with DPAPI encryption dependency problems, while Interviewee 5 had trouble with Brave's Linux keyring configuration being confusing, taking 15–20 minutes per task. Common difficulties are slow manual processes, problems with searching large forensic images, and encryption key problems.

**Question 2: Automated Decryption Role**

Automation was a necessity for efficiency and accuracy. Interviewee 1 emphasized that Chrome decryption automation would bring a 30-minute job to 5 minutes, conserving time for scoring CTF. Interviewee 2 emphasized its forensic advantage, noting that decryption of Edge by hand squanders time on investigation, and automation would assist in alleviating backlog pressure. Interviewee 3 referred to the accuracy benefits of Chrome, removing mistakes and speeding up evidence delivery from hours to minutes. Interviewee 4 foresaw Firefox automation reducing audit times by half, enhancing client productivity, while Interviewee 5 estimated a 70% decrease in preparation time for all browsers, with emphasis on data analysis. Benefits are huge time savings, reduced stress in workload, and improved-quality evidence.

**Question 3: Perceptions of CLI for Password Retrieval**

Interviewees preferred CLI for control and speed, provided it is accessible. Interviewee 1 appreciated straightforward commands like 'extract chrome folder' with CSV output for CTF ease. Interviewee 2 wanted to 'decrypt firefox path' with clean text lists to avoid flag complexity. Interviewee 3 wanted one-liners like 'get edge data' with timestamped CSV for forensic purposes. Interviewee 4 desired to 'run opera export' with table-formatted output for reports, and Interviewee 5 suggested a 'brave extract path' with a progress bar and minimal text for CTF efficiency. Desired features include easy commands, clean output formats (e.g., CSV, text), and usability enhancements such as progress indicators.

**Question 4: Encryption Key Management**

Good encryption key management was deemed important. Interviewee 1 mentioned DPAPI management as being necessary for Chrome, anticipating offline extraction failure without credentials unless brute-forced. Interviewee 2 viewed the Firefox master password as a significant offline hindrance, requiring workarounds to avoid hours of guessing. Interviewee 3 spoke about Edge's DPAPI problems in forensics, suggesting potential incomplete context. Interviewee 4 reiterated Opera's dependency on DPAPI, foreseeing offline key capture issues without system hooks, while Interviewee 5 was concentrating on the Brave keyring and master password management within Firefox, foreseeing manual cracking without automation. Challenges are issues around offline availability without live system credentials or environment.

**Question 5: CTF Extraction Times**

CTF extraction times took 20–35 minutes, with a strong desire to minimize this. Interviewee 1 averaged 30 minutes on Chrome, aiming to reduce that by half to 15 minutes for better leaderboard ranking. Interviewee 2 averaged 20 minutes on Firefox, aiming to reduce that by half to 10 minutes for competitive purposes. Interviewee 3 averaged 25 minutes on Edge, aiming to lower it by 40–50% to 12–15 minutes. Interviewee 4 reported 35 minutes on Opera, wanting a halving to 17 minutes, while Interviewee 5 spent 20–25 minutes on Brave, desiring a 50% cut to 10–12 minutes. A 50% time reduction was consistently seen as significant for improving CTF outcomes.

**Question 6: Useful Metadata**

Metadata preferences varied by context but centered on URLs, timestamps, and usernames. Interviewee 1 liked URLs and usernames in CTF for their target, liking CSV columns of (URL, Username, Password). Interviewee 2 liked timestamps and URLs in forensics for their timelines and opted for text files (Date, URL, Username, Password). Interviewee 3 liked usernames and timestamps as first for intent, opting for tabulated CSV (Username, Timestamp, URL, Password). Interviewee 4 asked for URLs and timestamps for audit patterns, suggesting plain lists (URL, Time, User, Pass), while Interviewee 5 suggested usernames and URLs for CTF context with a preference for plain text dumps (Username, URL, Password). CSV and plain text were the likes for them for easier analysis.

**Summary Tables**

| **Question** | **Key Insights** |
| --- | --- |
| Experience with Password Extraction | Time-consuming (15–35 min), navigation issues, encryption challenges |
| Importance of Automated Decryption | Time savings (e.g., 30 to 5 min), reduced stress, increased accuracy |
| Opinions on CLI | Simple commands (e.g., 'extract chrome'), clean output (CSV/text), usability enhancements |
| Handling Encryption Keys | Critical for effectiveness, offline access difficulties (e.g., DPAPI, master passwords) |
| Time Spent in CTF Scenarios | 20–35 min currently, 50% reduction desired (10–17 min) |
| Useful Metadata | URLs, timestamps, usernames; preferred in CSV or plain text |

*Table 6 Summary Table 1*

| **Challenges in Password Extraction** | **Description** |
| --- | --- |
| Time-Consuming Manual Processes | 15–35 minutes per image (e.g., Chrome, Opera) |
| Navigation Issues | Difficulty locating folders in large files (e.g., 10GB .raw) |
| Encryption Key Issues | DPAPI, master passwords complicate decryption |

Table 7 Summary Table 2

| **Benefits of Automated Decryption** | **Description** |
| --- | --- |
| Time Savings | Reduces tasks from 30 min to 5–10 min |
| Reduced Stress and Workload | Eases forensic backlogs and CTF pressure |
| Increased Accuracy | Minimizes manual errors, enhances evidence reliability. |

Table 8 Summary Table 3

| **Desired CLI Features** | **Description** |
| --- | --- |
| Simple Commands | e.g., 'extract chrome folder', 'decrypt firefox path' |
| Clean Output Formats | CSV or plain text lists for easy scanning |
| Usability Enhancements | Progress bars, minimal syntax complexity |

Table 9 Summary Table 4

# CHAPTER 4: CONCLUSION

The initial phase of the project demonstrates a rigorous investigation of the challenges of extracting and decrypting passwords from browser-stored credentials in forensic images. The student has performed commendably in laying a firm foundation by conducting a comprehensive literature review, as detailed in Chapter 2, that examines the encryption methods employed by leading browsers: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera, and Brave. This review goes into depth about Chrome's use of AES-256 encryption bound to Windows DPAPI, ‎Firefox's use of NSS cryptography with 3DES, and the shared Chromium-based ‎encryption approaches of Edge, Opera, and Brave. The research also critically analyzes the ‎limits of existing forensic tools like Autopsy and FTK, noting their failure in ‎having special automation for password recovery and decryption for particular browsers. This argument stresses the error-prone and time-consuming nature of manual process, particularly when handling large forensic images (e.g., 10GB or greater), which creates a need for automated software to enhance efficiency in digital forensics and Capture the Flag (CTF) competition.

The student's inquiry goes beyond simple identification of technology challenges, rather showing a solid grasp of the discipline by signifying practical implications. For instance, the recognition that hours of manual extraction are needed compared to the minutes targeted by the tool in development are in line with the project's objective of streamlining workflows for forensic examiners and CTF competitors. Surveys and interviews conducted in Chapter 3 also validate this need, with 85% of respondents agreeing that current procedures take too much time and 90% supporting a specialized tool for greater efficiency. This user-driven ‎data collection, combined with technical research, offers a good foundation for the development ‎stage of the tool, ensuring it addresses real-world problems. The student's ability to ‎synthesize these findings into clear objectives, effective extraction, proper decryption, ‎utilizable output, and CTF performance enhancement demonstrates that sufficient research has ‎been conducted to guide the project toward a viable solution.

While the research conducted is exhaustive and well-balanced, some gaps and areas of future study would increase the scope and efficacy of the project. One such area is the absence of depth while addressing operating system-specific variations in retrieving encryption keys. For example, the project mainly focuses on Windows DPAPI for Chromium-based browsers with fewer ‎focuses on the influence of Linux keyrings (like GNOME Keyring or KWallet) on access to keys for browsers like Brave or Firefox. Offline access to keys was noted by interview participants as a principal concern, particularly in CTF environments where system credentials are inaccessible. Additional research into cross-platform key management Windows and Linux mechanisms compared can make the tool more robust and adaptable, especially given its stated aim of compatibility on both operating systems.

Another area of neglect is in the exploration of new technologies that could affect future forensic tools. The literature review mentions post-quantum cryptography and zero-knowledge proofs, but these are not fully integrated into the tool's design considerations. With developments in browser security, the integration of initial quantum-resistant algorithm ‎support or versatility would make the tool future-proof, making it applicable beyond ‎existing encryption solutions (AES-256 and 3DES). Likewise, biometric authentication used in ‎sophisticated browsers to authenticate keys has limited exploration of the potential effect on the ‎forensic community. Forensic investigators may increasingly be unable to decrypt credentials ‎related to biometrics, and experimenting with fallback measures (e.g., memory dump key ‎recovery) may enhance the abilities of the tool.

In addition, mobile browsers such as Android ‎Chrome or iOS Safari fall outside the scope of the project based on differences in the ‎storage formats and encryption processes. While such limitation is proportionate to the timeline of ‎4–5 months and priority to desktop browsers, it is a lacuna of applicability. Interviews indicate that forensic examiners appreciate wide compatibility, and subsequent versions would be enhanced by preliminary research into mobile browser forensics to expand the tool's applicability. User behavior, another area not yet fully explored, could also be used to guide the tool's design. For instance, being aware of common practices like Firefox not having master passwords (highlighted in the literature as a flaw) might allow the tool to give top priority to functions that take advantage of such flaws for faster recovery, making it most effective in forensic as much as CTF contexts.

On the whole, the first part of the project does a fair amount of research, establishing a good motivation and technical basis for an automated password recovery tool. ‎The student's research adequately supports the project direction, supported by empirical data and ‎domain expertise. However, OS-specific key retrieval loopholes, emerging security technologies, mobile browser compatibility, and user behavior analysis provide room for further ‎research.

# REFERENCES

* Adam Leventhal, Sun Microsystems. (2008). *Can flash memory become the foundation for a new tier in the storage hierarchy?* Retrieved from queue.acm.org: https://queue.acm.org/detail.cfm?id=1413262
* *Adversarial Communication Networks Modeling for Intrusion Detection Strengthened against Mimicry*. (2019). Retrieved from dl.acm.org: https://dl.acm.org/doi/10.1145/3339252.3340335
* Alexander Fry. (2024). *Runtime Application Self-Protection (RASP), Investigation of the Effectiveness of a RASP Solution in Protecting Known Vulnerable Target Applications*. Retrieved from www.sans.org: https://www.sans.org/white-papers/38950/
* Anders, Vilius Povilaika. (2019). *How are Mozilla Firefox passwords encrypted?* Retrieved from security.stackexchange.com: https://security.stackexchange.com/questions/215881/how-are-mozilla-firefox-passwords-encrypted
* Apr4h. (2019). *Decrypting Browser Credentials For Fun (But Not Profit)*. Retrieved from apr4h.github.io: https://apr4h.github.io/2019-12-20-Harvesting-Browser-Credentials/
* BasuMallick, C. (2021, 1 12). *The Problem With Storing Passwords in Your Browser (and How to Fix It)*. Retrieved from www.spiceworks.com: https://www.spiceworks.com/it-security/identity-access-management/articles/the-problem-with-storing-passwords-in-browser/
* Boehm. (1986). Retrieved from dl.acm.org: https://dl.acm.org/doi/pdf/10.1145/12944.12948
* Brave, Help Center. (2024). *How do I use the built-in password manager?* Retrieved from support.brave.com: https://support.brave.com/hc/en-us/articles/360018185951-How-do-I-use-the-built-in-password-manager
* Carlos Jensen, Shelly D. Farnham, Steven M. Drucker, Peter Kollock. (2000). *The effect of communication modality on cooperation in online environments*. Retrieved from dl.acm.org: https://dl.acm.org/doi/10.1145/332040.332478
* Carsten Maartmann-Moe, Steffen E.Thorkildsen, Andre Arnes. (2009). *The persistence of memory: Forensic identification and extraction of cryptographic keys*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287609000486?via%3Dihub
* Chainlink. (2024). *What Is a Zero-Knowledge Proof?* Retrieved from chain.link: https://chain.link/education/zero-knowledge-proof-zkp
* Chitu Okoli. (2007). *The internet competitive landscape: insights from organisational ecology*. Retrieved from www.researchgate.net: https://www.researchgate.net/publication/228639352\_The\_internet\_competitive\_landscape\_insights\_from\_organisational\_ecology
* Chromium Docs. (2025). *Linux Password Storage*. Retrieved from chromium.googlesource.com: https://chromium.googlesource.com/chromium/src/+/master/docs/linux/password\_storage.md
* Clay Sheilds . (2011). *A system for the proactive, continuous, and efficient collection of digital forensic evidence*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287611000260?via%3Dihub
* Computer Security Resource Center. (2025). *Post-Quantum Cryptography*. Retrieved from csrc.nist.gov: https://csrc.nist.gov/projects/post-quantum-cryptography
* D. Crockford. (2006). *The application/json Media Type for JavaScript Object Notation (JSON)*. Retrieved from datatracker.ietf.org: https://datatracker.ietf.org/doc/html/rfc4627
* Farkhund Iqbal . (2008). *A novel approach of mining write-prints for authorship attribution in e-mail forensics*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287608000315?via%3Dihub
* Firefix Source Docs . (2025). *Network Security Services (NSS)*. Retrieved from firefox-source-docs.mozilla.org: https://firefox-source-docs.mozilla.org/security/nss/index.html
* Firefox, Mozilla Account . (2024, 11 29). *How Firefox securely saves passwords*. Retrieved from support.mozilla.org: https://support.mozilla.org/en-US/kb/how-firefox-securely-saves-passwords
* Frank Breitinger, Harald Baier, Douglas White . (2014). *On the database lookup problem of approximate matching*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287614000061?via%3Dihub
* Frequently Asked Questions. (2024). *Frequently Asked Questions*. Retrieved from www.sqlite.org: https://www.sqlite.org/faq.html#q14
* Fuggetta. (2000). *Software Process: A Roadmap*. Retrieved from dl.acm.org: https://dl.acm.org/doi/pdf/10.1145/336512.336521
* google chrome help . (2025). *How Chrome protects your passwords*. Retrieved from support.google.com: https://support.google.com/chrome/answer/10311524?hl=en
* Google Chrome Help. (2019). *ow Chrome protects your passwords*. Retrieved from support.google.com: https://support.google.com/chrome/answer/10311524?hl=en#zippy=
* Haase. (2004). *Virtual Reality and Habitats for Learning Microsurgical Skills*. Retrieved from link.springer.com: https://link.springer.com/chapter/10.1007/978-1-4471-3746-7\_3
* Higor Diego. (2023). *Cracking Firefox Encryption and Rescuing Saved Passwords!* Retrieved from dev.to: https://dev.to/higordiego/cracking-firefox-encryption-and-rescuing-saved-passwords-pfl
* IEEE. (2008). *Refactoring Tools: Fitness for Purpose*. Retrieved from ieeexplore.ieee.org: https://ieeexplore.ieee.org/document/4602672
* James McCormack, Anandu M Das. (2014). *Where are my browser passwords stored?* Retrieved from askubuntu.com: https://askubuntu.com/questions/525019/where-are-my-browser-passwords-stored
* Josiah Dykstra & Alan T. Sherman. (2012). *Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287612000266?via%3Dihub
* Luc, Tony Ruth. (2017). *How secure is Chrome storing a password?* Retrieved from security.stackexchange.com: https://security.stackexchange.com/questions/170481/how-secure-is-chrome-storing-a-password
* marcus. (2012). *How does a web browser save passwords?* Retrieved from softwareengineering.stackexchange.com: https://softwareengineering.stackexchange.com/questions/141402/how-does-a-web-browser-save-passwords
* Michael David Myers, Richard Baskerville. (2009). *Fashion Waves in Information Systems Research and Practice*. Retrieved from www.researchgate.net: https://www.researchgate.net/publication/220260003\_Fashion\_Waves\_in\_Information\_Systems\_Research\_and\_Practice
* Microsoft Edge Learn. (2024, 7 18). *Microsoft Edge password manager security*. Retrieved from learn.microsoft.com: https://learn.microsoft.com/en-us/deployedge/microsoft-edge-security-password-manager-security
* Mihir Bellare, Anand Desai, David Pointcheval & Phillip Rogaway . (2006). *Relations among notions of security for public-key encryption schemes*. Retrieved from link.springer.com: https://link.springer.com/chapter/10.1007/BFb0055718
* Narayan Ramasubbu, Chris F. Kemerer, M. S. Krishnan, Sunil Mithas. (2006). *Work Dispersion, Process-Based Learning, and Offshore Software Development Performance.* Retrieved from www.researchgate.net: https://www.researchgate.net/publication/220260001\_Work\_Dispersion\_Process-Based\_Learning\_and\_Offshore\_Software\_Development\_Performance
* Nicola Leoone. (1997). *Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S0890540197926304?via%3Dihub
* Nuseibeh, B. (2003). *ViewPoints: meaningful relationships are difficult!* Retrieved from ieeexplore.ieee.org: https://ieeexplore.ieee.org/document/1201254
* Obremski, N. C. (2009). *How are web site passwords encrypted by browsers?* Retrieved from stackoverflow.com: https://stackoverflow.com/questions/425221/how-are-web-site-passwords-encrypted-by-browsers
* Ohyicong. (2020). *decrypt-chrome-passwords*. Retrieved from github.com: https://github.com/ohyicong/decrypt-chrome-passwords
* Olhanzilla. (2017, 12 27). *What encription method is used in logins.json?'*. Retrieved from forums.mozillazine.org: https://forums.mozillazine.org/viewtopic.php?f=8&t=3036866
* Óscar. (2010). *How does Google Chrome store passwords?* Retrieved from superuser.com: https://superuser.com/questions/146742/how-does-google-chrome-store-passwords
* Phiter. (2016, 6 11). *Where in the filesystem does Firefox store saved passwords?* Retrieved from stackoverflow.com: https://stackoverflow.com/questions/37685932/where-in-the-filesystem-does-firefox-store-saved-passwords
* pycryptodome. (2024). *Welcome to PyCryptodome’s documentation*. Retrieved from pycryptodome.readthedocs.io: https://pycryptodome.readthedocs.io/en/latest/
* Richard Baskerville, . (2009). *Fashion Waves in Information Systems Research and Practice*. Retrieved from www.researchgate.net: https://www.researchgate.net/publication/220260003\_Fashion\_Waves\_in\_Information\_Systems\_Research\_and\_Practice
* setevoy. (2019, 10 12). *Chromium: Linux, keyrings && Secret Service, passwords encryption and store*. Retrieved from rtfm.co.ua: https://rtfm.co.ua/en/chromium-linux-keyrings-secret-service-passwords-encryption-and-store/
* Simson L. Garfinkel. (2010). *Digital forensics research: The next 10 years*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287610000368?via%3Dihub
* Simson L. Garfinkel, Michael McCarrin . (2015). *Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb*. Retrieved from www.sciencedirect.com: https://www.sciencedirect.com/science/article/pii/S1742287615000468?via%3Dihub
* SQLite Documentation. (2025). *Documentation*. Retrieved from www.sqlite.org: https://www.sqlite.org/docs.html
* Texas Tech Security Group. (2013). *How Browsers Store Your Passwords (and Why You Shouldn't Let Them)*. Retrieved from raidersec.blogspot.com: https://raidersec.blogspot.com/2013/06/how-browsers-store-your-passwords-and.html
* Toksöz, İ. (2011). Retrieved from www.researchgate.net: https://www.researchgate.net/publication/221670097\_Ilhan\_Toksoz
* Unode. (2024). *firefox\_decrypt*. Retrieved from github.com: https://github.com/unode/firefox\_decrypt
* Vidyut. (2013, 5 23). *How to retrieve your passwords saved in Google Chrome or Chromium on ubuntu linux*. Retrieved from vidyut.net: https://vidyut.net/how-to-retrieve-your-passwords-saved-in-google-chrome-or-chromium-on-ubuntu-linux/
* Viezelyte, K. (2024 , 2 1). *How to view, change, or delete saved passwords on Firefox*. Retrieved from nordpass.com: https://nordpass.com/blog/view-edit-delete-saved-passwords-firefox/
* Zakuskina, N. (2023, 8 15). *How to store passwords securely*. Retrieved from www.kaspersky.com: https://www.kaspersky.com/blog/how-to-store-passwords-securely/48784/

# APPENDICES

## Appendix A

A document with text on it

AI-generated content may be incorrect.

48 Title and Introduction in the PPF

A paper with text on it

AI-generated content may be incorrect.

49 Problem Statements in the PPF

A paper with text on it

AI-generated content may be incorrect.

50 Aims and Objectives in the PPF

A white paper with black text

AI-generated content may be incorrect.

51 Target Users in the PPF

A screenshot of a computer

AI-generated content may be incorrect.

52: Submission in FYPbank

A screenshot of a computer

AI-generated content may be incorrect.

53 Submission in FYPbank

## Appendix B

A screenshot of a computer

AI-generated content may be incorrect.

54 Ethics Form (Fast Track) Particioant Confidentiality

A screenshot of a computer

AI-generated content may be incorrect.

55 Ethics Form (Fast Track) ‎Nature of Research

A screenshot of a survey

AI-generated content may be incorrect.

56 Ethics Form (Fast Track) ‎ Nature of Research

A screenshot of a chat

AI-generated content may be incorrect.

57 Ethics Form (Fast Track) target Participants

A screenshot of a computer

AI-generated content may be incorrect.

58 Ethics Form (Fast Track) ‎Support Information

## Appendix C A screenshot of a computer AI-generated content may be incorrect.

Proposed Meeting Schedule

## 

A screenshot of a computer

AI-generated content may be incorrect.

60 Meeting Logs

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

61: 1st Meeting

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

62: 2nd Meeting

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

63: 3rd Meeting

## Appendix D

A screenshot of a computer screen

AI-generated content may be incorrect.

64 Gantt Chart

## Appendix E

|  |  |  |  |
| --- | --- | --- | --- |
| What is your name? | Email? | What is your gender? | What is your age group? |
| Aqel | Bnakeel735@gmail.com | Male | Under 25 |
| Tawfik Alhaja | TP071059@mail.apu.edu.my | Male | Under 25 |
| Ahmed Arafat | ahmedarafatjebli@gmail.com | Male | Under 25 |
| Majd Aleslam Al-Samhi | Tp072455@mail.apu.edu.my | Male | Under 25 |
| Abdullah Qanaan | aboody220064@gmail.com | Male | Under 25 |
| osamah | aasshh209077@gmail.com | Male | Under 25 |
| Nour Ahmed | nourahmedabdelmonim@gmail.com | Male | Under 25 |
| Kirollos | TP067904@mail.apu.edu.my | Male | Under 25 |
| Shehab Almanari | TP072060@mail.apu.edu.my | Male | Under 25 |
| Hatem | R0508005281@gmail.com | Male | Under 25 |
| Amanullah Ghauri | tp071215@mail.apu.edu.my | Male | Under 25 |
| Ebrahim Khaled Mohammed ALJABALI | Khalid.ibrahim883@gmail.com | Male | Under 25 |
| Haziq Irfan Radzali | tp072306@mail.apu.edu.my | Male | Under 25 |
| yazen abobakr ahmed al-mehdhar | yazn258@gmail.com | Male | Under 25 |
| Jeniffer Su Kai Li | jsukli20030412@gmail.com | Female | Under 25 |
| Mohammed Murshed | abdulmalekmohammed7@gmail.com | Male | Under 25 |
| Anas wadhah | Skulltrooper766@gmail.com | Male | Under 25 |
| Shafey | abdulshafey99@gmail.com | Male | Under 25 |
| Ibraheem | tp070765@mail.apu.edu.my | Male | Under 25 |
| Ebrahim | barhooooomkhaled@gmail.com | Male | Under 25 |
| Yara | Tp070773 | Female | Under 25 |
| Mohammed Zaid | MohammedZaid213@gmail.com | Male | Under 25 |
| Ali | Aloosh.ahmed@gmail.com | Male | 26-35 |
| Raef AL-kaleed | RaefOmer1990@gmail.com | Male | 36-44 |
| Heba | Hebajameel721@gmail.com | Female | 26-35 |
| Irfan Omer | BioOmer12@gmail.com | Male | 36-44 |
| Rube Ali | Rabab202Rab@outlook.sa | Female | 26-35 |
| Wafaa Binti Abduallh | wb12967205234@outlook.com | Female | 36-44 |
| Sameeha AL-ali | sosoalawe63@gmail.com | Female | Above 44 |
| Ammar Asem | algadeh\_benali@gmail.com | Male | 36-44 |
| Abood | aboodtshe6@gmail.com | Male | Under 25 |
| Assorm | sesomaboomer@outlook.com | Male | 36-44 |
| Tahani Hasen | Tetoalhelwa@gmail.com | Female | 36-44 |
| Wala Gamil | WalabintiGamil@gmail.com | Female | 26-35 |
| Nada | Nadosh18434@gmail.com | Female | Under 25 |
| Jamila Hamdi | JamilaHamdi2003@gmail.com | Female | Under 25 |
| Rabab Mohammed Labi | RababLabiMO@outlook.sa | Female | 36-44 |
| Khlood | Princesskoko@gmail.com | Female | 26-35 |
| Mohammed Omer Saleh ALgamdi | Xbehrreo1348@gmail.com | Female | 26-35 |
| Hatem | Hatemrasheed | Male | Under 25 |
| Ameera | tp070736@mail.apu.edu.my | Female | Under 25 |

Table : Respondence Demographic