

# Internship Report: PDF Protection Tool Using Python

**Intern Name:** Prem Kumar

**Project Title:** PDF Protection Tool Using Python

**Internship Organization:** Inlighn Tech

## Abstract

The project titled “**PDF Protection Tool Using Python**” was developed to create a command-line-based utility to protect PDF files using password encryption. PDF files often contain sensitive personal, academic, or business information, and securing them with a password provides an essential layer of protection against unauthorized access. This internship project primarily focused on leveraging Python’s **PyPDF2** library to add encryption and automate PDF file protection processes. Through this project, I gained deep insights into Python scripting, file handling, and basic cryptographic implementation in a practical context.

## 1. Introduction

In the digital world, documents are often shared in PDF format for their consistency and professionalism. However, sensitive documents such as financial reports, academic certificates, contracts, and identification proofs can easily be exposed if left unprotected. The objective of this project is to address this security concern by developing a Python tool that applies password-based encryption to PDF files.

This tool was developed as part of my internship at **Inlighn Tech**, where I was encouraged to design secure automation solutions. The script accepts a source PDF file, an output filename, and a password as command-line arguments and produces a protected version of the original file. The project involved key areas such as **command-line interface development**, **exception handling**, and **PDF encryption** using PyPDF2.

## 2. Tools and Technologies

- **Language:** Python 3.x
- **Library:** PyPDF2
- **IDE Used:** VS Code / PyCharm
- **Platform:** Windows/Linux (Cross-platform)
- **Terminal/CLI:** Command Prompt, Linux Terminal

## 3. Objectives

- Create a tool that protects PDF files using password encryption.
- Use Python's built-in and external libraries for handling PDF files.
- Accept command-line arguments to enhance usability.
- Handle errors and exceptions effectively.
- Provide feedback to the user about successful encryption.

## 4. How the Tool Works

The core functionality of the PDF Protection Tool is broken into these steps:

### 1. Input Handling:

Takes three command-line arguments – input file name, output file name, and password.

### 2. Reading the Input PDF:

Opens the file using PyPDF2 and creates a PDF reader object.

### 3. Creating a New PDF Writer Object:

Copies all pages from the original file into a new PdfWriter instance.

### 4. Encryption:

Encrypts the file using the `encrypt()` method with a user-provided password.

## 5. Saving the Output:

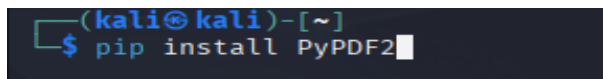
Writes the encrypted file to the specified output location.

## 6. Error Handling:

Catches file not found or permission-related issues to ensure a smooth user experience.

# 5. Implementation Steps

## *Step 1: Installing Required Library*

A terminal window with a dark background. The prompt is `(kali㉿kali)-[~]` in blue. Below it, the command `$ pip install PyPDF2` is entered in blue, followed by a white cursor block.

```
(kali㉿kali)-[~]  
$ pip install PyPDF2
```

## Step 2: Full Python Script

```
$ import PyPDF2
import sys

def encrypt_pdf(input_file, output_file, password):
    try:
        # Open the input PDF in binary read mode
        with open(input_file, 'rb') as pdf_file:
            reader = PyPDF2.PdfReader(pdf_file)
            writer = PyPDF2.PdfWriter()

            # Copy each page to the new writer
            for page in reader.pages:
                writer.add_page(page)

            # Apply encryption
            writer.encrypt(user_password=password)

            # Write to the output file
            with open(output_file, 'wb') as output_pdf:
                writer.write(output_pdf)

            print(f"✓ Successfully encrypted '{input_file}' as '{output_file}' with password protection.")

    except FileNotFoundError:
        print("✗ Error: Input file not found.")
    except Exception as e:
        print(f"✗ Unexpected error occurred: {e}")

# Accept command-line arguments
if len(sys.argv) != 4:
    print("Usage: python pdf_encryptor.py input.pdf output.pdf password")
else:
    input_pdf = sys.argv[1]
    output_pdf = sys.argv[2]
    user_password = sys.argv[3]
    encrypt_pdf(input_pdf, output_pdf, user_password)
```

## 6. Command-Line Usage

### Syntax:

```
(kali@kali)~$ python pdf_encryptor.py source.pdf protected.pdf mysecurepassword
```

### *Example:*

```
python pdf_encryptor.py report.pdf report_protected.pdf pass1234
```

## 7. Sample Output Screenshot

✓ Successfully encrypted 'report.pdf' as 'report\_protected.pdf' with password protection.

## 8. Exception Handling and Validations

To make the script robust and user-friendly, multiple exceptions were handled:

- **FileNotFoundError:** If the input file doesn't exist
- **PermissionError:** If the script doesn't have write permissions
- **General Exception:** For any unknown issue like unsupported PDF versions

Sample error output:

✗ Error: Input file not found.

## 9. Security Perspective

While this tool applies a **user password** to protect the file, it does not use advanced encryption mechanisms like AES-256. PyPDF2 uses **RC4 or AES-128**, which is sufficient for basic password protection but **should not be used for high-security environments**.

For highly sensitive documents, consider using pikepdf or external tools like **qpdf** or **Adobe Acrobat Pro**, which support more advanced cryptographic options.

## 10. Limitations and Improvements

### *Current Limitations:*

- Doesn't support batch encryption
- Doesn't apply owner-level permissions (e.g., restrict printing or copying)
- PyPDF2 has limited AES support
- GUI not available (CLI only)

### *Future Improvements:*

- Add support for directory/batch processing
- Create a **Tkinter-based GUI**
- Include owner permissions and metadata editing
- Add decryption functionality

## 11. Learning Outcomes

By completing this project, I have:

- Learned how to manipulate PDF files programmatically using Python
- Understood the role of encryption and access control in document security
- Developed a functional command-line utility with robust exception handling
- Enhanced my understanding of **PyPDF2**, Python scripting, and system I/O
- Learned how to **structure user input via command-line arguments**

## 12. Real-World Use Cases

- Securing PDF invoices before sending to clients
- Encrypting academic certificates or ID scans
- Automating document protection in enterprise workflows
- Adding password control for internal reports

## 13. Conclusion

The PDF Protection Tool Using Python demonstrates how simple, yet powerful automation can enhance document security. While tools like Adobe Acrobat offer similar features, building such a tool from scratch allowed me to understand how **encryption, file handling, and command-line scripting** work under the hood.

This project enhanced my confidence in applying Python for cybersecurity-related tasks and introduced me to real-world issues such as exception handling, usability, and security trade-offs. As I continue my journey in tech, I plan to expand this tool into a GUI version with added features and enterprise-level security options.