Project Blueprint: Secure Data Wiper & Verifier (SDWV)

Version: 1.0 **Date:** September 1, 2025 **Mission:** To design and build a secure, user-friendly, cross-platform data wiping tool that generates a tamper-proof, verifiable certificate of erasure, thereby building trust in IT asset recycling and promoting India's circular economy.

1. High-Level System Architecture

The solution is composed of two main parts: the **Offline Wiping Tool** (a bootable USB) and the **Online Verification Service** (a web portal). They interact indirectly via the generated certificate.

Components:

1. **Bootable Environment:** A customized SystemRescue Live Linux distribution. It provides the base OS and hardware drivers, solving compatibility issues.
2. **Core Application (GUI):** A Python-based graphical user interface built with PyQt5. This is the user-facing component they interact with.
3. **Wiping Engine (`nwipe`):** The trusted, open-source command-line tool that performs the secure data erasure. It runs as a hidden background process controlled by our Core Application.
4. **Certificate Module:** A Python component responsible for generating, digitally signing, and saving the PDF/JSON erasure certificate.
5. **Verification Service:** A lightweight Python Flask web application that exposes a single API endpoint to verify the authenticity of a certificate.

2. Low-Level Design & Implementation Details

2.1 The Bootable Environment (SystemRescue)

- **Base Distribution: SystemRescue**.
    - **Reasoning:** It's lightweight, designed for system administration, has excellent hardware support (BIOS/UEFI), and provides official tools for easy customization.
- **Customization Method:** We will use SystemRescue's `autorun` feature. This avoids complex ISO rebuilding for most of our needs.
- **USB Directory Structure:**

```
/ (Root of USB Drive)
|-- autorun.yml          # The configuration file to auto-launch our app
|-- sysrescue.d/         # Standard SystemRescue directory
|-- EFI/                 # Standard UEFI boot files
|-- boot/                # Standard BIOS boot files
`-- sdwv_app/              # OUR APPLICATION DIRECTORY
    |-- main.py            # The main Python script for our GUI
    |-- gui/               # PyQt UI files, icons, etc.
    |-- libs/              # Self-contained Python dependencies (.whl
files)
```

```
    |-- private_key.pem    # The private key for signing certificates
```

- **Auto-launch Configuration (`autorun.yml`):**

```
---
# This file tells SystemRescue to run our main application script
immediately on boot.
autorun:
  - command: /usr/bin/python /run/archiso/bootmnt/sdwv_app/main.py
```

Note: `/run/archiso/bootmnt/` is the path where the USB drive is mounted when SystemRescue boots.

2.2 The Core Application (GUI)

- **Technology:** Python 3 + PyQt5.
    - **Reasoning:** PyQt5 allows for a modern, responsive, and visually appealing GUI which is superior to standard libraries like Tkinter.
- **User Experience (UX) Flow:**
    - **Screen 1: Welcome & Disk Detection:**
        - The application launches full-screen.
        - It displays a clear title, a brief explanation, and a list of detected storage devices.
        - **Disk Detection Logic:** The `main.py` script will execute the command `lsblk --json -o NAME,MODEL,SIZE,TYPE` and parse the JSON output to identify all devices of `TYPE="disk"`.
        - **Display Format:** Each disk will be listed intuitively: "Seagate ST500 (512 GB)" or "Samsung 970 EVO (1 TB)". The bootable USB itself will be excluded from the list.
    - **Screen 2: Wipe Confirmation:**
        - After the user selects a disk and clicks "Wipe," a large, red confirmation dialog appears.
        - **Warning Text:** "This will permanently destroy all data on the selected drive. This action is irreversible. Are you sure you want to proceed?"
        - Requires the user to type "ERASE" into a text box to enable the final "Confirm Wipe" button.
    - **Screen 3: Wiping in Progress:**
        - The UI shows a prominent progress bar, the current wipe method (e.g., DoD 5220.22-M), and elapsed time.
        - This screen is updated in real-time by parsing the output from the hidden `nwipe` process (see 2.3).

- **Screen 4: Completion & Certificate:**
    - Displays a "Wipe Successful" message.
    - Provides two options for the certificate:
        - "Save Certificate to another USB": Prompts the user to insert another USB drive to save the PDF/JSON files.
        - Displays the QR code on-screen for instant verification via a mobile device.

2.3 The Wiping Engine (`nwipe`) Integration

- **Hiding `nwipe`:** The Core App will use Python's `subprocess` module to run `nwipe` as a non-interactive, headless process. The user will never see the `nwipe` text interface.
- **Command Execution:**

```
import subprocess

# Example command to wipe a disk without showing the nwipe UI
device_to_wipe = "/dev/sda"
command = [
    "nwipe",
    "--autonuke",          # Automatically select and run
    "--method=dodshort",   # Specify the wipe method
    "--noua",              # No user interaction
    device_to_wipe
]
process = subprocess.Popen(command, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT, text=True)
```

- **Real-time Progress Parsing:**
    - The Python script will read the `process.stdout` line by line in a loop.
    - It will use regular expressions (`re` module) to find progress updates in the output stream (e.g., `... (\d+\.\d+)% ...`).
    - The extracted percentage will be used to update the PyQt progress bar in the GUI.

2.4 Certificate Generation & Security

- **Certificate Content:** The certificate (both PDF and JSON) will contain:
    - `certificateId`: A unique UUID.
    - `deviceModel`: e.g., "Samsung 970 EVO".
    - `deviceSerial`: The device's serial number (obtained from `lsblk` or `hdparm`).
    - `deviceSize`: e.g., "1 TB".
    - `wipeMethod`: e.g., "NIST SP 800-88 Purge (DoD 5220.22-M)".
    - `wipeTimestamp`: ISO 8601 format (e.g., "2025-12-15T14:30:00Z").
    - `status`: "Success".

    - `signature`: A digital signature of all the above fields.
- **Technology:**
    - **PDF:** `reportlab` library.
    - **JSON:** `json` standard library.
    - **Cryptography:** `cryptography` library.
- **Digital Signature Process:**
    - A `secp256k1` or `RSA` key pair (private and public) is generated **once** by the development team.
    - The `private_key.pem` is placed inside the `sdwv_app/` folder on the USB.
    - The `public_key.pem` is hardcoded into the Verification Service.
    - After a wipe, the certificate data is concatenated into a single string, and the `cryptography` library is used to sign this string with the private key. The resulting signature is added to the certificate.

2.5 The Verification Service (Web Portal)

- **Technology:** Python + Flask.
    - **Reasoning:** Flask is a micro-framework, perfect for creating a simple, single-purpose API.
- **API Endpoint:** `GET /verify`
    - **Parameter:** `id` (the `certificateId` from the certificate).
- **Workflow:**
    - A user scans the QR code, which directs their browser to `https://your-verifier.com/verify?id=...`.
    - The Flask app receives the request.
    - (Optional but recommended) The app looks up the ID in a simple database to see if it's a known certificate.
    - The user is prompted to upload the JSON certificate file.
    - The server receives the JSON, separates the data from the signature, and uses the hardcoded **public key** to verify if the signature matches the data.
    - It returns a simple page: "✅ Verified Authentic" or "❌ Verification Failed".

3. Phased Development Plan & Roadmap

Phase 1: Proof of Concept (Internal Hackathon)

- Manually boot SystemRescue from a standard USB.
- Manually run `nwipe` from the command line on a test drive.

- Write a separate Python script on your development machine that can:
    - Generate a hardcoded JSON certificate.
    - Sign it using the `cryptography` library.
    - Write a second script to verify the signature with the public key.
- **Goal:** Prove that the core cryptographic and wiping components work.

Phase 2: Core Product Development (Submission)

- Set up the USB directory structure (`sdwv_app/`, etc.).
- Develop the full PyQt5 GUI with all four screens.
- Integrate disk detection logic using `lsblk`.
- Implement the `subprocess` logic to run `nwipe` headlessly and parse its output for the progress bar.
- Integrate the certificate generation and signing module.
- Implement the "Save to USB" and "Display QR Code" features.
- Create the `autorun.yml` file to tie everything together.
- **Goal:** A fully functional, bootable, one-click wiping tool.

Phase 3: Verification Ecosystem (Pre-Finale)

- Develop the Flask web application for verification.
- Design the simple upload and verification status page.
- Deploy the web application to a hosting service.
- **Goal:** A complete, end-to-end user journey from wipe to verification.

Phase 4: Testing & Polish (Finale Prep)

- Test the bootable USB on at least 5 different physical machines (various brands, BIOS/UEFI).
- Refine the GUI for clarity and aesthetics.
- Create a backup video of a successful wipe and verification.
- Practice the final presentation.
- **Goal:** A rock-solid, demo-ready product and a compelling story.