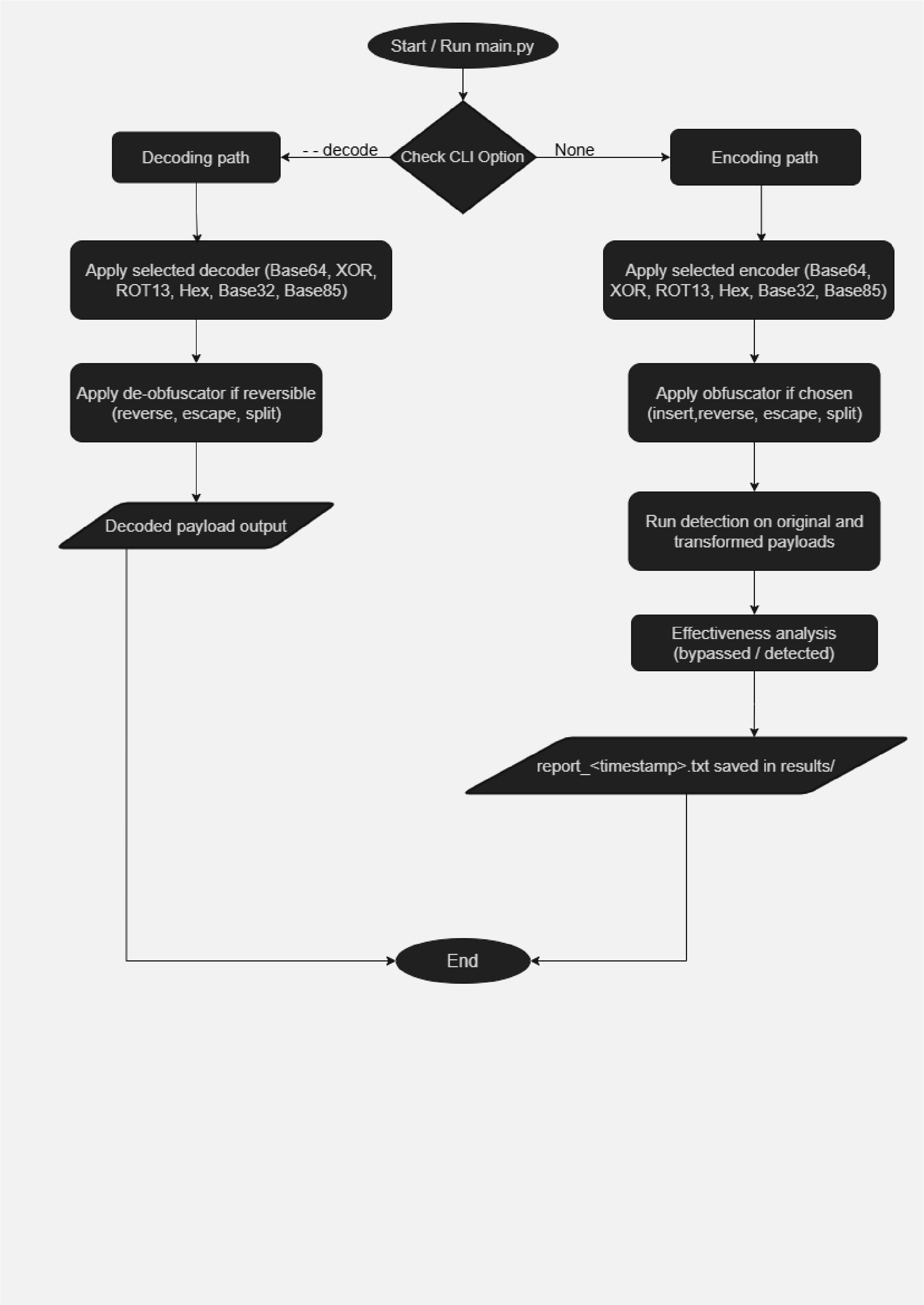
Flowchart - Cloak : Payload Encoder, Obfuscation, Detection, Reporting, and Decoding Framework


Start / Run main.py
Check CLI Option
- - decode
None
Decoding path
Encoding path
Apply selected decoder (Base64, XOR, ROT13, Hex, Base32, Base85)
Apply selected encoder (Base64, XOR, ROT13, Hex, Base32, Base85)
Apply de-obfuscator if reversible (reverse, escape, split)
Apply obfuscator if chosen (insert,reverse, escape, split)
Decoded payload output
Run detection on original and transformed payloads
Effectiveness analysis (bypassed / detected)
report_<timestamp>.txt saved in results/
End

Explaination :

The **Cloak flowchart** represents the operational sequence of the framework, showing how a payload is processed based on the user's command-line selection. It demonstrates two main execution paths - **Decoding Path** and **Encoding/Obfuscation Path** - both ending with structured output and termination.

1. Start / Run main.py

- The framework execution begins when the main script is launched.
- Initial configuration and argument parsing are performed.

2. Check CLI Option (Decision Node)

- The system evaluates the command-line argument provided by the user.
- This decision determines whether the framework will:
  - **Decode an existing payload**, or
  - **Encode and obfuscate a payload** for detection testing.

Decoding Path (–decode option)

3. Apply Selected Decoder

- The framework applies the chosen decoding technique to reverse earlier transformations.
- Supported reversible decoders typically include:
  - Base64
  - XOR
  - ROT13
  - Hex
  - Base32 / Base85

4. Apply De-Obfuscator (If Reversible)

- Additional cleanup steps such as **reverse**, **escape removal**, or **string splitting resolution** are applied.
- This stage restores the payload closer to its original readable or executable form.

5. Decoded Payload Output

- The recovered payload is displayed or exported.
- No detection analysis is required in this branch since the focus is restoration rather than evaluation.

Encoding / Obfuscation Path (Default / No –decode option)

3. Apply Selected Encoder

- The payload is transformed using one or more encoding algorithms (Base64, XOR, ROT13, Hex, Base32, Base85).
- This step alters the payload's appearance without necessarily changing its behavior.

4. Apply Obfuscator (Optional)

- Structural or string-level obfuscation is introduced using methods such as **insert**, **reverse**, **escape**, or **split**.
- The purpose is to simulate evasion of signature-based detection systems.

5. Run Detection Simulation

- The framework tests both the **original** and **transformed** payloads against predefined signature rules.
- This simulates how simple detection engines might respond to altered payload structures.

6. Effectiveness Analysis

- Results are compared to determine whether the transformations **bypassed** or were **detected** by the rule set.
- This stage measures the success of encoding and obfuscation strategies.

7. Report Generation

- A structured report file (e.g., report_<timestamp>.txt) is automatically saved in the **results/** directory.
- The report typically includes:
    - Techniques used
    - Detection outcomes
    - Transformation summary
    - Effectiveness metrics

End

- After output or report generation, the framework execution terminates.outcomes while maintaining clear reporting and reproducibility.