Simulating the Lorenz SZ42 Cipher Machine in Python

A Historical and Computational Exploration of WWII Cryptography

Robert Gravelle

July 2025

"THE BREAKING OF LORENZ WAS ONE OF THE GREATEST INTELLECTUAL ACHIEVEMENTS OF THE WAR" — B.J. COPELAND

1. Introduction

Imagine decrypting Hitler's secret orders without ever seeing the machine—a feat achieved by Bletchley Park's codebreakers in 1941. The Lorenz SZ42 cipher, developed by C. Lorenz AG, was a cryptographic system far more complex than Enigma, securing teleprinter messages for the German High Command during WWII. Its breaking, led by mathematician Bill Tutte and engineer Tommy Flowers, produced "Ultra" intelligence that shaped D-Day and shortened the war by an estimated two years [1]. The effort birthed Colossus, the world's first programmable electronic computer. This project, part of my cryptography portfolio alongside AES, RSA, and Enigma simulations, implements the Lorenz cipher in Python to educate users about its historical and technical significance while showcasing my software engineering skills. Explore the full implementation at [cryptography_portfolio/lorenz-cipher at main · Rob-Gravelle/cryptography_portfolio].

2. Cipher Mechanics

The Lorenz SZ42 is a stream cipher that encrypts 5-bit ITA2 (Baudot) code, a teleprinter standard using 5-bit sequences for characters (e.g., A = 00011), enabling compact transmission. It generates a pseudorandom keystream by XORing bits from 12 wheels:

- 5 χ (chi) wheels (lengths 41, 31, 29, 26, 23): Step every character.
- 5 ψ (psi) wheels (lengths 43, 47, 51, 53, 59): Step only if μ1=1, adding irregularity.
- 2 μ (mu) wheels (lengths 61, 37): Control ψ stepping.

Each wheel holds a fixed bit sequence (0s and 1s). The χ and ψ outputs (when ψ steps) are XORed to produce a keystream bit, which is XORed with each plaintext bit. My simulation uses 8-bit ASCII for modern compatibility, simplifying the historical 5-bit ITA2 encoding.

Figure 1: Lorenz Wheel Structure

[χ1-χ5 (41,31,29,26,23)] → Step every bit → XOR

[ψ1-ψ5 (43,47,51,53,59)] → Step if μ1=1 → XOR

[μ1,μ2 (61,37)] → Step every bit

Output: Keystream bit (XOR with plaintext)

Caption: Lorenz SZ42 wheel structure, showing χ, ψ, and μ wheel interactions.

3. Cryptanalysis

In Bletchley Park the cryptanalysis teams broke the Lorenz cipher without seeing the machine, working under intense secrecy and tight deadlines. A German operator's error in August 1941—reusing wheel settings for a 4,000-character message—allowed John Tiltman to recover plaintext. Bill Tutte deduced the 12-wheel structure using statistical "delta" methods, exploiting ψ wheel stuttering patterns [2]. Tommy Flowers' Colossus, operational by December 1943, used 1,500 vacuum tubes to test wheel settings at 5,000 characters/second, reducing the χ wheel search space from ~22 million to ~1,898 combinations. Flowers, facing skepticism, personally funded parts to deliver Colossus in just 10 months. This effort provided critical intelligence for battles like D-Day and pioneered electronic computing. See the cover page for a Colossus replica image.

4. Python Simulation Overview

This Python project simulates the Lorenz SZ42 with historical wheel lengths, demonstrating skills in object-oriented programming, CLI development, and JSON handling:

- Modular Design: The Wheel class models individual wheels, and LorenzMachine handles encryption/decryption (see lorenz.py).
- Keystream Logic: χ wheels step every bit, ψ wheels step if μ1=1, and the keystream bit is the parity of XORed χ and ψ outputs.
- Unique Feature: The wheel_config.py module saves/loads wheel states to JSON, replicating historical "key sheets" for secure, reproducible encryption (see Figure 2).
- CLI and Testing: cli.py provides a command-line interface, and test_lorenz.py validates functionality with tests for encryption and edge cases.

Code Example (from lorenz.py's generate_keystream):

python

```
chi_bits = [w.current() for w in self.chi_wheels]

psi_bits = [w.current() for w in self.psi_wheels] if mu1_bit == 1 else [0] * 5

combined = [c ^ p for c, p in zip(chi_bits, psi_bits)]

keystream_bit = sum(combined) % 2
```

Figure 2: JSON Wheel Configuration

```
json

{
  "chi": [{"bits": [1, 0, ...], "position": 0}, ...],
  "psi": [{"bits": [0, 1, ...], "position": 0}, ...],
  "mu": [{"bits": [1, 1, ...], "position": 0}, ...]
}
```

Caption: JSON configuration from wheel_config.py, mimicking historical key sheets.

4. Example Output

Figure 3: CLI Output

```
bash

$ python cli.py --mode encrypt --message "HELLO" --seed 1234
Encrypted Bits (binary): 0001010101101011001001001010010001011100
Encrypted Bits (hex): 0x1564a485c
$ python cli.py --mode decrypt --message "00010101..." --seed 1234
Decrypted: HELLO
```

Caption: Sample output from cli.py, showing encrypted bits in binary and hex formats. See full implementation at [cryptography_portfolio/lorenz-cipher at main · Rob-Gravelle/cryptography_portfolio].

6. Modern Relevance

The Lorenz cipher's pseudorandom keystream inspired modern stream ciphers (e.g., RC4, Salsa20) and chaotic encryption systems. A 2024 study used Lorenz-like chaotic systems for image encryption, achieving 7.9975 entropy [3]. The cipher's reliance on wheel-based

randomness parallels modern challenges with secure pseudorandom number generators, explored in my AES project. This simulation could be extended into an educational tool, enhancing its impact.

7. Acknowledgments

- Bill Tutte: Deduced the cipher's structure via statistical analysis.
- Tommy Flowers: Built Colossus, funding components himself [2].
- Bletchley Park Team: Enabled critical WWII intelligence.
- National Museum of Computing (UK): Preserves Colossus replicas.

"The breaking of Lorenz was one of the greatest intellectual achievements of the war" [2].

References

- Wikipedia. (2025). Lorenz Cipher. Retrieved from https://en.wikipedia.org/wiki/Lorenz_cipher
- Copeland, B. J. (2006). Colossus: The Secrets of Bletchley Park's Codebreaking Computers. Oxford University Press.
- Zhang, X. (2024). Improved Lorenz Chaotic System for Image Encryption. Journal of Cryptographic Engineering, 14(3), 123–130.