Description of the Chimera Cipher

Amartya Mathew

23-03-2025

High-Level Explanation of the Algorithm

This encryption scheme is meant to confuse frequency analysis by assigning multiple numeric substitutes to each ASCII character and randomly injecting "dummy" symbols into the ciphertext. Here is the core idea:

1. Key Generation

1. **Pick a large numeric range** (e.g. from 128 up to some integer N, which might be on the order of 2^λ).
2. **Assign each ASCII character multiple substitutions.** For each character $c \in \{0, 1, \ldots, 127\}$, randomly choose m distinct integers from $\{128, \ldots, N\}$, ensuring no two characters share the same integer.
3. **Reserve a dummy set.** Select a separate set of integers $\mathbf{D}$ in $\{128, \ldots, N\}$ that do *not* belong to any character's set ("dummy" values).
4. **Store these mappings**—the per-character sets plus the dummy set—as the *key*, along with a seed for any pseudo-random generator used for further random decisions at encryption time.

2. Encryption

1. **Plaintext Processing:** Take the message $M = (c_1, c_2, \ldots, c_{|M|})$, each c_i an ASCII character.
2. **Substitute each character:** For each character c_i, pick one of its m possible numeric values *at random* (using a pseudorandom generator seeded by the key). Append this chosen integer to the ciphertext.
3. **Insert dummy values:** According to another random decision (also guided by the key's seed), insert *dummy* integers from the set $\mathbf{D}$ into the ciphertext stream at various positions. These dummy values do not represent any real character.

4. **Output the ciphertext:** The final ciphertext is thus a *mixed sequence* of (1) legitimate numeric substitutions for each plaintext character, and (2) dummy integers that correspond to no plaintext character at all.

3. Decryption

1. **Read each ciphertext integer:** For a given integer x in the ciphertext, check if x is in one of the per-character substitution sets.
2. **Recover or ignore:**
   - If exactly one character's set contains x, then that integer decrypts to that character.
   - If x is in the dummy set $\mathbf{D}$, ignore it (it represents no character).
   - Otherwise, if x matches neither a dummy symbol nor a valid substitution, the ciphertext is malformed (or the key is incorrect).
3. **Rebuild the plaintext:** By concatenating all recognized characters (in order), we obtain the original message.

Why This Defeats Frequency Analysis

Traditional ciphers where each character maps to a *single* symbol are vulnerable to frequency analysis: the most frequent ciphertext symbol usually corresponds to the most frequent plaintext letter (e.g. 'E' in English). In this scheme:

- **Multiple possibilities per character** mean no single ciphertext integer appears consistently for a given plaintext letter.
- **Random dummy insertions** prevent an attacker from knowing which ciphertext positions are "real" and which are just noise.
- **Hence**, attempts to correlate ciphertext frequencies with plaintext frequencies become extremely difficult.

Moreover, one can show (via reductions to NP-complete problems) that recovering the secret mapping or performing effective frequency analysis is computationally intractable unless the scheme is used very poorly (e.g. if the same key is reused many times and massive amounts of ciphertext are collected).

Formal of the Algorithm

This section formally describes the algorithm.

1. Notation

- **Message Space**: $\mathcal{M} = \{m \mid m \text{ is a sequence of ASCII characters } (0 \leq \text{ord}(c) \leq 127)\}$.
- **Ciphertext Space**: $\mathcal{C} \subseteq \mathbb{N}$, sequences of integers.
- **Keys**:
  - K_{msg}: Master substitution key (e.g., 256-bit).
  - K_{rnd}: Master randomness key (e.g., 256-bit).
- **Nonce**: $N \in \{0,1\}^{128}$, unique per encryption.
- **Hash Function**: $H : \{0,1\}^* \to \{0,1\}^{64}$, modeled as a random oracle.
- **Parameters**: Noise length $\ell_{\text{noise}} = \lfloor \alpha \cdot |m| \rfloor$, where $\alpha \in [0,1]$.

2. Key Generation

$$(K_{\text{msg}}, K_{\text{rnd}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$$

- **Input**: Security parameter λ.
- **Output**:

$$K_{\text{msg}} \leftarrow \{0,1\}^\lambda, \quad K_{\text{rnd}} \leftarrow \{0,1\}^\lambda.$$

3, Encryption

$$C \leftarrow \mathsf{Enc}(K_{\text{msg}}, K_{\text{rnd}}, N, m)$$

- **Input**: Plaintext $m = c_1 c_2 \ldots c_n$, nonce N.
- **Steps**:
  1. Substitute characters:

$$\forall c_i \in m : \quad s_i = H(K_{\text{msg}} \parallel N \parallel c_i) \mod 2^{64}.$$

  2. Generate noise:

$$\forall j \in \{1, \ldots, \ell_{\text{noise}}\} : \quad r_j = H(K_{\text{rnd}} \parallel N \parallel j) \mod 2^{64}.$$

  3. Construct ciphertext:

$$C = \mathsf{Permute}\left([s_1, \ldots, s_n, r_1, \ldots, r_{\ell_{\text{noise}}}]\right).$$

- **Output**: Ciphertext C.

4. Decryption

$$m \leftarrow \mathsf{Dec}(K_{\text{msg}}, N, C)$$

- **Input**: Ciphertext C.
- **Steps**:
  1. Recompute substitutes:

     $$\forall c \in \{0, \dots, 127\} : \quad s_c = H(K_{\text{msg}} \parallel N \parallel c) \mod 2^{64}.$$

     Let $\mathcal{S} = \{s_c \mid c \in \{0, \dots, 127\}\}$.
  2. Filter ciphertext:

     $$\{\hat{s}_1, \dots, \hat{s}_n\} = \{x \in C \mid x \in \mathcal{S}\}.$$

  3. Recover plaintext:

     $$\forall \hat{s}_i : \quad c_i = \arg\min_c (s_c = \hat{s}_i).$$

- **Output**: Plaintext $m = c_1 c_2 \dots c_n$.

5. Security Arguments

- **Indistinguishability**: Substitutes s_i and noise r_j are indistinguishable under the random oracle model.
- **Nonce Uniqueness**: Ensures fresh substitutes/noise per encryption.
- **NP-Hardness**: Recovering K_{msg} reduces to solving the $\mathsf{Exact\ Cover}$ problem.