Collatz Cipher paper

Marc Lecointre

March 5, 2024

I do not provide any warranty that this cipher is in any way reliable. DO NOT USE WITH CRITICAL DATA ! This is document is not an official research paper but rather a technical documentation. You are welcome to contribute in any way !

Stay tuned, new features may be added through time ! (check the date of the document to see if you are up-to-date). Please open an issue or contact me directly, should you have any suggestion or bug/security issue/feature to submit.

Made with LaTeX.

Abstract

A polyalphabetic cipher that derivate a key from Hasse's algorithm. Noise and randomness is added to make cryptanalysis more difficult. Works only on text. However, any file can be encrypted by converting binary stream to base64 text. This, however, drastically increases the size of the encrypted file.

Contents

Introdution

First, a bit of context and reminder.

Symmetric cipher

Symmetric ciphers use symmetric algorithms to encrypt and decrypt data. They use the same key to encrypt and decrypt data. They differ from asymmetric ciphers, which use two keys : one to encrypt, and the other one to decrypt. The former reduces the data to send via a secure channel, while the latter allow to establish a secure channel without having any.

Collatz conjecture

AKA :

- $3n + 1$ conjecture
- Ulam conjecture
- Hasse's algorithm
- Syracuse problem
- ... and a bunch of other names.

One of the most famous unsolved problems in mathematics. Consider the following operation on an arbitrary positive integer :

- If the number is even, divide it by two.
- If the number is odd, triple it and add one.

Doing this repeateadly always gives 1, and then goes in an infinite loop ($1 \to 4 \to 2 \to 1 \to ...$), no matter the number you start with (for any positive integer, zero excluded).

Let's define this mathematically.

We consider the function f such as :

$$f(n) = \begin{cases} \frac{n}{2} \text{ if } n \equiv 0 \mod 2 \\ \\ 3n + 1 \text{ if } n \equiv 1 \mod 2 \end{cases}$$

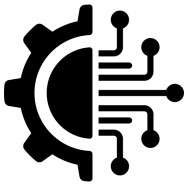
And we can form the following sequence, with any integer $n \geq 1$:

$$a_i = \begin{cases} n & \text{if } i = 0 \\ f(a_{i-1}) & \text{if } i > 0 \end{cases}$$

The Collatz conjecture is : this process will eventually reach the number 1, regardless of which positive integer is chosen initially.

$$\forall n \geq 1 / a_0 = n, \exists \varepsilon \geq 0 / a_\varepsilon = 1 \tag{1}$$

How it works



Note : the secrets library is used, a library provided for cryptographic use.

Key

The key is made of three things :

- A mixed set of chars (randomly swapped) coming from DEFAULT_CHARSET defined in params.py, which the message should limit to (chars not in this set will not be usable in the message !).
- A key field, which is the "seed" that will be used for Hasse's algorithm.
- A split char, that will be used to split the message and noise. If present in the message, it is removed.
- A set of "null chars", that should NOT be used in the message (doing so will stop the program).

Key generation

- The unused chars defined in params.py are splited in 3 : one char is defined as the split char, and the rest is randomly distributed among null chars and charset. The split char is part of the charset.
- The 'key' field is filled with secret's function token_hex (actually makes a 2×nbytes long hex number).

Key storage

Keys are nothing more than dictionaries (content described just above). They are stored as json with additional informations. Each key is stored in one file, named after the fingerprint (SHA256) of the key (in json format).

 The key is stored IN CLEAR, meaning anyone having access to it can decrypt any message encrypted with it !

However, it is possible to only store a random string, thereafter called salt, which combined with a password and hashed, can be used as a seed to regenerate the key each time it is neeeded. This is recommendend, especially if your system and backups are not encrypted !

Please note that the key is not stored in encrypted form, but rather regenerated each times. Its hash is used to determine if the password is valid.

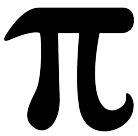
Message encryption

- The message is splited in blocks (of 4096 chars by default).
- Noise is added to the begining and the end of the message as well as between all the blocks, using the spliting char as a separator. This is to make each encryption of the message unique (see the section below) and cryptanalysis difficult/confusing.
- Hasse's algorithm is applied to generate a sequence (as long as the message) of numbers modulus the size of the charset. If the sequence is too short (compared to the message size), it is repeated.
- Each character i is swapped with the character $i + key[i]$ modulus the size of the charset. Plus, there are between 1 and 45% of chance that a null char would be added at each character encoding. (this percentage is "decided" at each message encryption)
- Optionally, the enciphered message can be encoded in base64 (so that the message can be transmitted via channels that does not support UTF-8).

Message decryption

- Null chars are removed.
- Each character i is swapped with the character $i - key[i]$ (+ the size of the charset, if the result is negative).
- The noise is removed using the split char. (by ignoring odd blocks)

Let's do some math



Number of possible keys

Let's note Ψ the number of possible keys. Then :

$$\Psi(n, C, u) = K_n \times C! \times (2^{u-2} - 1) \tag{2}$$

With :

- K_n the number of possible keys with n "bytes", e.g. K_{100} = fff...f (200 times) - 100...0 (1, and 0 199 times)
- C the number of chars in the charset (not including the null chars, thus this part should be bigger in reality)
- u is the number of unused chars. This part comes from here and we do not count the split char

Number of possible encrypted message with the same message and key

Let's note $\mathcal{M}(C)$ the number of possible encrypted messages with the same key. Then :

$$\mathcal{M}(C) = (\beta + 1) \sum_{k=0}^{m} C^k \tag{3}$$

With :

- C the number of chars in the charset (not including the null chars, so this part may be slightly bigger)
- m the upper bound (maximum of noise lenght)
- β the number of blocks in the message

$$\beta = \left\lfloor \frac{\sigma}{\omega} \right\rfloor \tag{4}$$

With :

- σ the number of chars in the message
- ω the size of one block

Version 2 (older version)

Number of possible encrypted message with the same message and key

Let's note $\mathcal{M}(C)$ the number of possible of keys. Then :

$$\mathcal{M}(C) = 2 \sum_{k=0}^{m} C^k \tag{5}$$

With, again, C the number of chars in the charset, and m the upper bound.

Security considerations

Frenquency analysis

Each character is swapped to another one with a different indice. This, plus the null chars, defeat frequency analysis, even to figure out which language is used.
The coincidence rate (calculated with another tool of mine)) looks to tend to 0.01, no matter French or English.

It is around :

- 0.0778 for French
- 0.0762 for German
- 0.0738 for Italian
- 0.0667 for English
- 0.0385 for randomly generated text

Brute forcing the key

As seen above, there are 10^{1200} possible keys with nbytes = 100. With nbytes = 1,000 it increases to 10^{3368}.
Brute forcing is not possible in acceptable time with our present knowledge and computer power.

Hashing algorithm

Algorithm used :

- SHA-256 which is considered secure. It is also used in Bitcoin, for example. (ID, integrity check, legacy password hashing)
- bcrypt and SHA-512 are also considered secure. (password hashing)

Bibliography

Collatz conjecture - Wikipedia

Icons credits

Caesar cipher icons created by Pixelmeetup - Flaticon
Warning icons created by Freepik - Flaticon
Safe icons created by Assia Benkerroum - Flaticon
Idea icons created by Good Ware - Flaticon
Technology icons created by Smashicons - Flaticon
Pi icons created by Freepik - Flaticon