Alten Labs

BMI

-

How Algorithm Works

-

How to use Algo DiDIN

Table of Contents

[1 - Introduction 4](#_Toc162270984)

[2 - DiDIN Enrollment Algorithm 5](#_Toc162270985)

[2.1 - Input data: 5](#_Toc162270986)

[2.2 - How enrollment works: 6](#_Toc162270987)

[3 - DiDIN Identification Algorithm 7](#_Toc162270988)

[3.1 - Input data: 7](#_Toc162270989)

[3.2 - How the algorithm works 7](#_Toc162270990)

[4 - Reed-Solomon Correction Code 8](#_Toc162270991)

[4.1 - How Reed-Solomon Correction Works 8](#_Toc162270992)

[4.1.1 - ENCODING 8](#_Toc162270993)

[4.1.2 - DECODING 8](#_Toc162270994)

[4.2 - Example 8](#_Toc162270995)

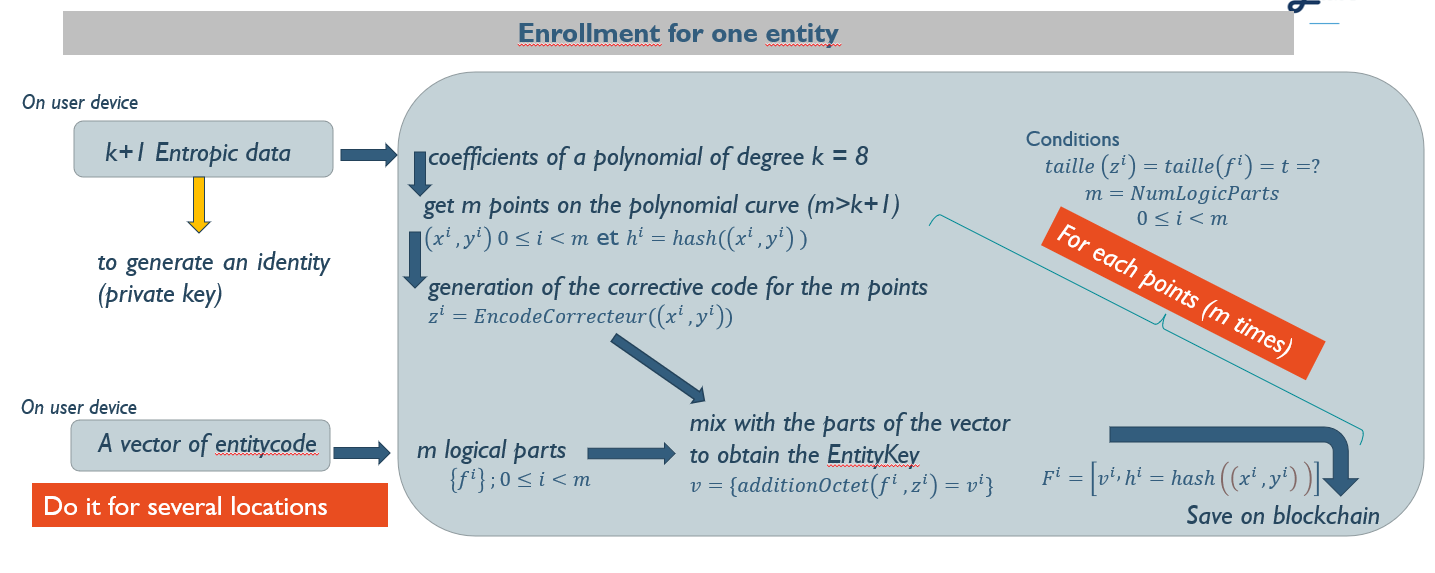
# Introduction

The **DiDIN**  algorithm is a data encryption algorithm with a Reed-Solomon-based correction code, optimized for the processing of identifying data of any entity. It is based on two elements: **entropy**, represented by coefficients forming a polynomial, and the EntityCode vector, which structures the data allowing identification.

*Entropy*, which can be random or based on specific data, is converted into a polynomial to retrieve points from it. The *EntityCode* vector is divided into subvectors for data management and segmentation for future identification.

The Reed-Solomon correction code is used to add redundant data, correcting potential errors during data transmission. This method ensures that identification data is retrieved even in the presence of transmission errors. This document allows you to understand how the algorithm works and what data to put as input to make it work.

# DiDIN Enrollment Algorithm



## Input data:

ENTROPY :

* + **Description:** Taking a value corresponding to a precise identity or a random numerical value on 128 bits (8 coefficients of a polynomial that are each on 16 bits).
  + **Example data:** 
    - 8 coefficients on 16 bits, i.e. values between 0 and 65536.
    - Can correspond to the identity of an object, an identity... (Imagine the transformation of a block *Last name+first name+date of birth* in numerical value or the reference of an aircraft part. However, the value must be numeric and a maximum size of 128 bits.
    - **Example: Random coefficients** 
      * Tables of values of 8 random coefficients on 16 bits: [25655, 5067, 38246, 57434, 39229, 51710, 51782, 54046].
      * Polynôme obtenue : 25655 + 5067x + 38246x2 + 57434x3 + 39229x4 + 51710x5 + 51782x6 + 54046x7

VECTEUR OF ENTITYCODE :

* + **Description:** Identification data of the entity. Variable size for the dataset. However, it is necessary to be able to divide the data set into n logical parts corresponding to subvectors.
  + **Example data:** 
    - Maximum size not fixed. It seems possible that there is more data at enrolment because we want to have the most complete data when we do the first enrolment to have as much information as possible.
    - Size of the **n** logical parts sets **between 20 and 40 values coded on 1 byte** to avoid padding and leave room for correction as well as dot size. Can correspond to a footprint with numerical minutiae data corresponding for example to the X, Y, Z axis, angle, type or peaks corresponding to piezoelectric information.
    - **Example: Size 25 minutiae**
      * X : 96.77, Y : 95.81, Z : 271.75, Angle : 190, Type : Bifurcation, 1
        + [9, 6, 7, 7, 9, 5, 8, 1, 2, 7, 1, 7, 5, 1, 9, 0, 8, 5, 1, 0, 0, 0, 0, 0, 0]
      * X : 86.29, Y : 43.77, Z : 249.93, Angle : 34.65, Type : Terminaison, 2
        + [8, 6, 2, 9, 4, 3, 7, 7, 2, 4, 9, 9, 3, 3, 4, 6, 5, 2, 0, 0, 0, 0, 0, 0]
      * X : 8.93, Y : 73.63, Z : 356.54, Angle : 284.63, Type : Terminaison, 2
        + [8, 9, 3, 7, 3, 6, 3, 5, 6, 5, 4, 2, 8, 4, 6, 3, 2, 0, 0, 0, 0, 0, 0, 0]
      * X : 9.69, Y : 13.97, Z : 50.29, Angle : 235.28, Type : Bifurcation, 1
        + [9, 6, 9, 1, 3, 9, 7, 5, 0, 2, 9, 3, 5, 2, 8, 1, 0, 0, 0, 0, 0, 0, 0, 0]
      * Full Vector: [9, 6, 7, 7, 9, 5, 8, 1, 2, 7, 1, 7, 5, 1, 9, 0, 8, 5, 1, 0, 0, 0, 0, 0, 0, 8, 6, 2, 9, 4, 3, 7, 7, 2, 4, 9, 9, 3, 3, 4, 6, 5, 2, 0, 0, 0, 0, 0, 0, 8, 9, 3, 7, 3, 3, 3, 7, 3, 6, 3, 5, 6, 5, 4, 2, 8, 4, 6, 3, 2, 0, 0, 0, 0, 0, 0, 0, 9, 6, 9, 1, 3, 9, 7, 5, 0, 2, 9, 3, 5, 2, 8, 1, 0, 0, 0, 0, 0, 0, 0, 0]. You can add as much data as you want as long as the size of the subvector is standardized and identical. For example, the size of minutiae remains fixed (size of 25 numeric values)

The importance is to indicate the size of the subvectors to check the consistency of the data.

## How enrollment works:

* *Step 1:* Take the two data mentioned above as input.
* *Step 2:* Creation of a polynomial using the coefficients corresponding **to the entropy k**.
* Step 3: Split the EntityCode vector into m equal subvectors
* *Step 4:* Taking **m** points on the polynomial corresponding **to the m subvectors of *Vectors of EntityCode*** (Adaptable for any type of entity, not just minutiae).
* Step 5: Encoding the dots (x on one byte and y on 8 bytes) using the Reed-Solomon correction algorithm according to the number of correction bytes **c** desired. The set of encoded points are the same size as the **m subvector of EntityCode** and create a hash for each of them.
* Step 6: Mixing the points taken on the polynomial with the subvectors of EntityCode
* Step 7: Recording on the blockchain of the mix and hash corresponding to the point for future comparison to enrollment.

# DiDIN Identification Algorithm

## Input data:

cf [vector of entityCode](#Donnéesdentrée). The expected data are more or less the same. However, it may seem logical that there is less data at the time of identification than at the time of enrolment because we have less precision.

## How the algorithm works

* Step 1: Search by the surname/first name/date of birth of the potential *EntityKey* available in the blockchain.
* Step 2: Split the *EntityCode* vector for identification into equal subvectors.
* Step 3: Extraction of the m parts of the *EntityCodeAuth* subvector recorded on the blockchain as well as the hash for each of these subvectors.
* Step 4: Loop over the m subvectors of EntityCodeAuth and the p subvectors of EntityCode to try to find enough points to reconstruct the polynomial.
  + Step 1: Subtraction between a subvector m of *the EntityCodeAuth* and p of *the EntityCode.*
  + Step 2: Correction with *ReedSolomon* of the value obtained possibly corresponding to the point of a polynomial and creation of a hash of this new value.
  + Step 3: Comparison of the hash created and the hash recorded on the blockchain **for the subvector p** (if a point of the polynomial is equal).
* Step 5: If enough points are found **(k+1 points),** we reconstruct the polynomial allowing us to find the identity. Otherwise, we go to step 1 and redo the algorithm for another name/surname/address if it has determined other possibilities.

# Reed-Solomon Correction Code

## How Reed-Solomon Correction Works

The Reed-Solomon corrector code is an error-correcting coding system based on the theory of polynomials and finite fields, specifically Galois fields. This system consists of two main phases: encoding and decoding. The correcting power of the Reed-Solomon code is related to the number of error correction bytes added. For example, with 10 bytes of correction, it is possible to correct up to 5 bytes of error.

### ENCODING

* **Creating redundant data during encoding:** The Reed-Solomon algorithm does not add random bits. This data is generated according to mathematical rules on polynomial and finite field theory.
* **Polynomial transformation** : The message (in the form of numbers) is transformed into a polynomial. Each element of the message becomes a coefficient in this polynomial.
* **Redundant Data Calculation:** From this polynomial, the algorithm generates redundant data by evaluating the polynomial at different points. These additional values are the redundant data added to the message.

### DECODING

* **Polynomial point check:** When the message is received, the algorithm uses the same points that were used to calculate the redundant data to determine where each point in the polynomial should theoretically be. It then reconstructs the polynomial from the received message.
* **Comparison of results:** By evaluating the reconstructed polynomial at the corresponding points, the results should theoretically correspond exactly to the redundant data received. The algorithm then compares these results with the redundant data actually received. If the values match, it means that the message is probably intact. If they don't match, it indicates that there were errors during transmission, and the algorithm attempts to correct them using the redundant data**.**

## Example

Let's take the example of the vector X = [3,1,4,1,5]. In our case, we can consider that each value corresponds to a byte for the *Reed-Solomon algorithm*.

Encoding

* Step 1: Transformation of the vector into a polynomial:
* Step 2: We choose a point to evaluate to obtain redundant data. Let us take x = 1,2,3
  + We obtain the encoded vector [3,1,4,1,5,14,57,164]

Error Transmission

During transmission, the vector becomes [3,1,4,2,5,14,57,164], so we have the fourth digit that is altered.

Decoding and Correction:

* Step 1: The algorithm tries to reconstruct the polynomial from the received data.
* Step 2: By evaluating the reconstructed polynomial x=1,2,3, he compares the results with the redundant parts. He finds that the values do not match.
* Step 3: Using the redundant values, the algorithm determines that the error is in the 4th coefficient of the polynomial.
* Step 4: He corrects the number 2, it becomes 1 again.

So we get the corrected vector [3,1,4,1,5] which is our original vector. Since we had 3 redundant values, we could have corrected a maximum of one value of the vector (*nombre\_octets\_correction\_possible = nombre\_octet\_données\_redondantes / 2*).

## Byte-by-byte error correction limits

A limitation of the algorithm, using the Reed-Solomon code for error correction, is its ability to correct errors byte by byte. Because this method is effective in correcting isolated errors in numerical data, it runs into difficulties when errors cause subtle but global changes in a dataset.

For example, if a numeric value such as 16.99 is altered to 17.01 during transmission, the Reed-Solomon code will not be able to effectively correct this error because all the bytes representing the value will have changed. This is due to the nature of the correction which is digital and not physical, i.e. the code corrects errors based on changing individual bytes rather than understanding or interpreting the data in its overall context. Thus, for data where small variations are significant, the byte correction method will not be able to correct this error.

In addition, if the data is too inaccurate, the correction can lead to significant changes that do not accurately reflect the original information. Conversely, if the data is extremely accurate, there may not be enough redundant bytes to allow for effective correction in case of small variations. We will not find that they are the same data when they are more or less the same:

**Data that is too inaccurate** : If the measurements are rounded to the nearest centimeter (e.g., 15 cm, 16 cm), a transmission error changing 15 cm to 16 cm will be corrected by the algorithm. However, this correction may not be representative of the true value.

**Too precise data:** If the measurements are now extremely accurate, for example, 15.2427 cm, correcting a small error, such as changing 15.2427 cm to 15.2389 cm, could require a significant number of bytes to correct the value when these are extremely close values. The algorithm would therefore never be able to find common points in the identification when there are some.

It is therefore very important to choose something consistent and to keep a standardised size (which does not change for each value at enrolment and identification: *data is changed to the hundredth ready when it was at the tenth ready at the time of enrolment to suit the ReedSalomon correction code*) for the data.