Dragon Crypto - An innovative cryptosystem

Awnon Bhowmik
Department of Mathematics and Computer Science, CUNY York College
90-24 Guy R. Brewer Blvd
Jamaica, NY 11451
awnon.bhowmik@yorkmail.cuny.edu

Unnikrishnan Menon
Department of Electrical and Electronics Engineering, Vellore Institute of Technology
Vellore Campus, Tiruvalam Rd
Katpadi, Vellore, Tamil Nadu 632014
unnikrishnanr.menon2017@vitstudent.ac.in

May 8, 2020

Abstract

In recent years cyber-attacks are continuously developing, therefore security specialists must stay busy in the lab inventing new schemes to keep attackers at bay. In this paper we outline the details of a new crypto system based on the dragon curve fractal. The security level of this scheme is based on multiple private keys, that are crucial for effective encryption and decryption of data. This paper covers how core concepts emerging from fractal geometry can be used as a trapdoor function for a new cryptosystem.

keywords: dragon curve, dragon fractal, dragon curve fractal, heighway dragon curve, heighway dragon fractal, cryptography, cryptosystem, crypto system, secure encryption, Iterative Function System, IFS, iteration, precision, trapdoor function

1 Introduction

Fractals are just a self replication of a particular pattern. There are many fractals that are found in nature. One such fractal curve is known as *Heighway Dragon*, or simply the *Dragon Curve*. Because this is a fractal, hence it requires an IFS (Iterated Function System). In simple words, we make a simple pattern, put it into a recursive function, before running a loop and calling the function over and over again until our objective is accomplished.

2 The Dragon Fractal

A fractal is a repetition of an initial geometric shape. This is generated by something known as the Iterative Function System (IFS). The dragon curve is one such fractal. There are multiple ways to generate this fractal.

2.1 Generating the Dragon Fractal

This dragon curve fractal, or its generation algorithm forms the basis of this cryptosystem, and hence the proposed name *dragon crypto*. Here, we take an arbitrary string of characters, pass each character through the *Koblitz encoder* to obtain the starting point for the IFS.

The iterations of the Dragon curve can easily be generated by folding a strip of paper n number of times. We take a strip of paper and fold it in half to the right.[1] We fold it in half again, to the right. We continue this process as many times as we want (of course it will be hard to fold after a certain point). Let's say we do it 2 times.

Now we unfold the strip and relax it, and we'll notice that every bend/corner of the strip has become a 90° turn. What we have now is the second iteration of the Dragon Curve. If we fold the strip of paper in half for a third time and then open it up, we would get the third iteration and so on...

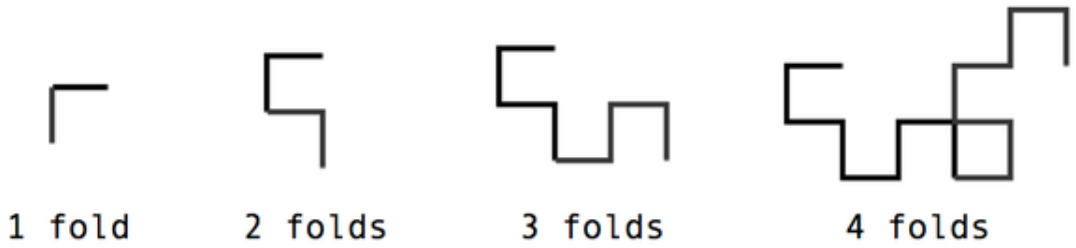

Fig 1. **Paper folding method**

Now all we have to do is visualize a turtle walking along these lines and it should be able to predict how to move. Let F = forward, L = left, R = right. The following sequences are associated with the respective iterations...

- 1st iteration- F L F L
- 2nd iteration- F L F L F R F L
- 3rd iteration- F L F L F R F L F L F R F R F L ⋮

If we know the n^{th} iteration, we can predict the $(n+1)^{\text{th}}$ iteration in the following way. Let's say we are trying to derive the third iteration from the second iteration.

- We ignore the last element for now (F L F L F R F). In the remaining elements, if there's a R, it is replaced by L. And if there is a L, it is replace by a R. The sequence now becomes (F R F R F L F R).
- Now, we flip these elements about the mid point. The sequence turns into (R F L F R F R F).
- Add the last element which we had ignored in the first step. The sequence becomes (R F L F R F R F L).
- Appending the result of step 4 to the 1st iteration and we will have our third iteration ready.

Using this same logic, we can predict the next iterations. A fractal curve generated with 15 iterations is as follows...

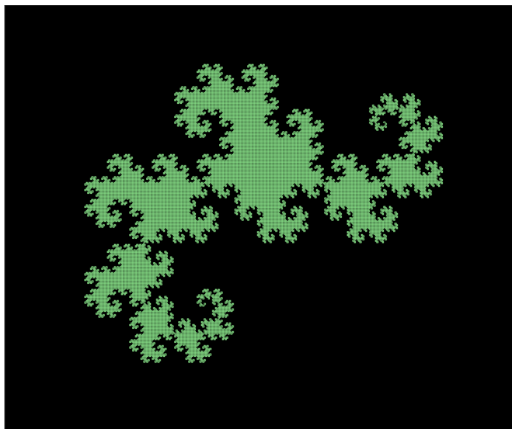

Fig 2. **Dragon Curve Fractal**

3 Koblitz Encoding & Decoding Algorithm

The encoding algorithm[2] is as follows...

- Given a message M, convert each character m_k into a number a_k using Unicode, where $b = 2^{16}$ and $0 < a_k < 2^{16}$
- Convert the message M into an integer using

$$m = \sum_{k=1}^{n} a_k b^{k-1}$$

  . In practice we choose an n to be less than or equal to 160 such that m satisfies $m \leq 2^{16 \cdot 160} < p$.
- Fix a number d such that $d \leq \frac{p}{m}$. In practice we choose the prime p large enough so that we can allow $d = 100$.
- For integers $j = 0, 1, 2, \cdots d-1$ we do the following loop
  - Compute the x coordinate of a point on the elliptic curve as $x_j = (dm + j) \mod p$ where $m = \left\lfloor \frac{x_j}{d} \right\rfloor$
  - Compute $s_j = (x_j^3 + Ax + B) \mod p$
  - If $(s_j)^{\frac{p+1}{2}} \equiv s_j \mod p$, then define y coordinate of a point on the elliptic curve as $y_j = (s_j)^{\frac{p+1}{4}} \mod p$. Return the point (x_j, y_j).

Thus we are able to encode our message M, as an element of the Abelian group $G = E(\mathbb{F}_p)$.
The decoding algorithm is as follows...

- Consider each point (x, y) and set m to be the greatest integer less than $\frac{x-1}{k}$. Then the point (x, y) decodes as the symbol m.

4 Trapdoor Function

A trapdoor function is a function that is easy to compute in one direction, yet difficult to compute in the opposite direction without special information, called the *trapdoor*. Trapdoor functions came to prominence in cryptography in the mid-1970s with the publication of asymmetric (or public-key) encryption techniques. Several function classes have been proposed, and it soon became obvious that trapdoor functions are harder to find than was initially thought.

The main novelty of our cryptosystem is the use of **The Heighway Dragon Fractal** as a trapdoor function. The algorithm starts off with a secret message that needs to be encrypted (called the plainText).

The Koblitz encoder accepts the plainText along with the curve parameters (obtained from the private key). The koblitz encoder spits out an encoded point in 2D Cartesian space for each character present in the plainText with the help of curve parameters.These points are now the starting point of the dragon curve fractals. For each character, a corresponding starting point and a dragon fractal are generated.

Now the components of the private key (including size,iterations,angle) are used to generate the the fractal for each character from their corresponding starting points.

Private key parameters

- The size defines the length of each forward step.
- The Iterations defines the number of iterations for generating the fractal.
- The Angle defines the starting angle for the fractal.
- Elliptic curve parameters a and b for the function $y^2 = x^3 + ax + b$

Once the fractals are generated for each encoded starting points, the corresponding ending points are noted and stored.

$$p_{\text{start}} = \{(x_i, y_i)\} \qquad i = 1, 2, \cdots n$$
$$p_{\text{end}} = \{(x_j, y_j)\} \qquad j = 1, 2, \cdots n$$

5 Proposed Algorithm

1. Input an arbitrary string, and split into list of characters, spaces and special characters
2. Encryption
   (a) Pass each character's ASCII representation through the Koblitz encoder (using Koblitz Algorithm in section 3). Every character is then represented by a point (x_i, y_i) on the Cartesian coordinate. Let's call these set of points as initial starting points.
   (b) For a fixed length l, an angle θ and a fixed number of iterations n for all characters (or points) in the set, we generate a dragon fractal curve. The end point (x_j, y_j) for each starting point (x_i, y_i) is stored into another list.
   (c) A special padding is applied on the list of end points as follows

   $$"Xx_1Xx_2 \cdots Xx_nXYy_1Yy_2 \cdots y_nY"$$

   . This turns it into a string. This string is the encrypted cipher text.
3. Decryption
   (a) Remove padding and regenerate points. Parse through each element of the cipher text string and split it at "XY" into 2 sequences. So now we have 2 strings:

   $$"Xx_1Xx_2 \cdots Xx_nX" \qquad \text{and} \qquad "Yy_1Yy_2 \cdots y_nY"$$

   Next, we parse through the 2 lists and recover the ordered set of end point coordinates (x_j, y_j)
   (b) The list of end point coordinates are taken and based on the (size, iterations, angle) parameters present in the private key, a dragon curve fractal is generated in reverse by initializing the turtle at an endpoint facing the correct direction based on the angle. If the private key parameters are correct, the reversed dragon fractal will end at the initial starting point. This process is repeated for all characters. The points obtained are stored in a list.
   (c) The list of points obtained from the previous step is passed through the **Koblitz Decoder**. This returns the ASCII values of each character in original string. This is finally converted to string to retrieve original message.

6 Example demonstrating the algorithm

```
----------------------------------Dragon_Crypto-------------------------------------

Enter Message: HELLO

----------------------------------Private Key---------------------------------------
Size of Dragon: 5
No. of Iterations: 3
Angle: 0
Curve Parameters
Enter A: -1
Enter B: 7

----------------------------------ASCII Values--------------------------------------

[72, 69, 76, 76, 79]

----------------------------------Koblitz Points------------------------------------

[(1441, 173), (1382, 247), (1522, 59), (1522, 59), (1581, 204)]

---------------------------------Padded Encryption----------------------------------

Encrypted Message:  X1431X1372X1532X1532X1571XY183Y237Y49Y69Y214Y

---------------------------------Split String---------------------------------------

X1431X1372X1532X1532X1571X     and     Y183Y237Y49Y69Y214Y

--------------------------Padding Removed and Regenerate--------------------------

[(1431, 183), (1372, 237), (1532, 49), (1532, 69), (1571, 214)]

-----------------------------------Decryption-------------------------------------

Decrypted message:  HELLO
```

Fig 3. **Sample Test**

7 Conclusions

Cryptography is a 40 year old topic where a lot has been discovered but a lot more yet remains unknown. The essence of any traditional cryptosystem relies on three hard mathematical problems: the integer factorization problem, the discrete logarithm problem or the elliptic curve discrete logarithm problem. Shor's algorithm can be used to easily compromise the security of these conventional cryptosystems. So for a secure future, it is critical to come up with innovative trapdoor functions that can be incorporated in the heart of the encryption scheme.

In this paper, we proposed a new encryption algorithm based on the well known dragon curve fractal, due to the fact that not many people have worked on an approach that is closely related to the work discussed here. Whether on not this system is truly usable at the industry level is out of scope of this paper for the moment, but hopefully we will be able to build up on our present research work in the near future.

We have shown that our proposed algorithm is working based on the appropriate parameters. It has been noticed that the run time complexity can be drastically reduced by using smaller number of iterations while increasing the length to compensate for the precision of the endpoint as well as the Euclidean distance from the start point. So far no edge cases have been found, but we suspect the presence of one or more. This is dependent on the calculation precision of the machine. Our work so far is available here.[3]

References

[1] A. Bhowmik and U. Menon, "Taming the dragon," May 2018. [Online]. Available: https://www.quora.com/q/rrqrrlbcxwyhoqay/The-Dragon-Curve-Fractal-in-Python

[2] N. D. Rene´e Brady and A. Tracy, "Encrypting with elliptic curve cryptography," July 2010. [Online]. Available: https://www.math.purdue.edu/~egoins/notes/Encrypting_Text_Messages_via_Elliptic_Curve_Cryptography.pdf

[3] A. Bhowmik and U. Menon, "Dragon-crypto," 2020. [Online]. Available: https://github.com/awnonbhowmik/Dragon-Crypto