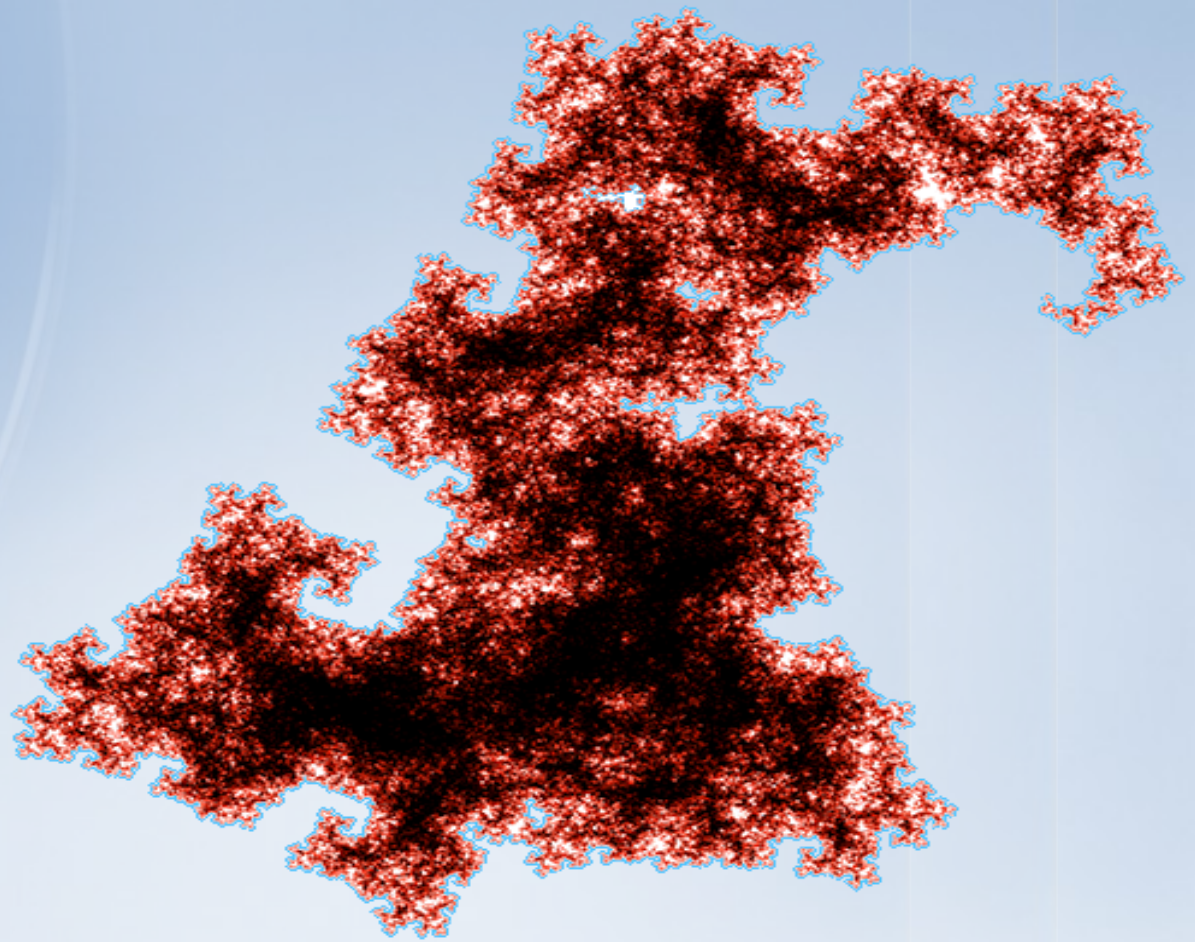
988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

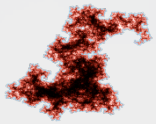


Crypt::FNA & MySQL

How to make a fractal encrypted database

988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

SECTION I



Definition of {F} fractal curves

Crypt::FNA & MySQL

Definition of {F} fractal curves

IT'S FANTASTIC! FRACTAL CURVE AND SYMMETRIC ENCRYPTION

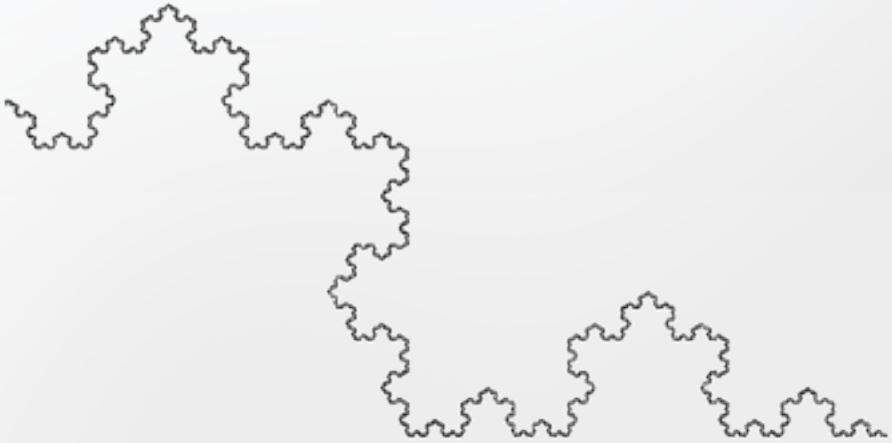
- Acronym di Fractal Numerical Algorithm
- What is Crypt::FNA
  - It is the Perl implementation of **two algorithm**

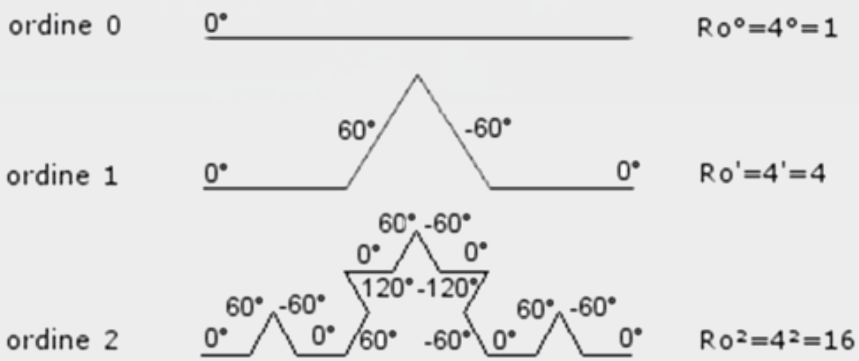
1. to define the set of fractal curves {F}
2. Apply {F} curves on a symmetric cryptosystem

Crypt::FNA & MySQL

Definition of {F} fractal curves

Start from the linear fractal Koch curves then generalize





Here are the directions of the various orders in a building in a triangle :

0

0, 60, -60, 0

0, 60, -60, 0, 60, 120, 0, 60, -60, 0, -120, -60, 0, 60, -60, 0

Crypt::FNA & MySQL

Definition of {F} fractal curves

- The property that interests us is that any number of the triangle is obtained as combination of the above amount in row

- We observe that we can express the properties of self-similarity of the Koch curve through a similar construction, combining the values of the base with those derived from the combination and so on iterating the procedure.

Crypt::FNA & MySQL

Definition of {F} fractal curves

Writing the sequence of angles as elements of a vector, we read the main property: the first addend is the group or branch on which iterates the construction process. The second term is the location of which we are calculating, within that branch.

```
a(0) =    a(0)  +    a(0)
a(1) =    a(0)  +    a(1)        I GRUPPO
a(2) =    a(0)  +    a(2)
a(3) =    a(0)  +    a(3)
-----------------------------------------------
a(4) =    a(1)  +    a(0)
a(5) =    a(1)  +    a(1)        II GRUPPO
a(6) =    a(1)  +    a(2)
a(7) =    a(1)  +    a(3)
-----------------------------------------------
a(8) =    a(2)  +    a(0)
a(9) =    a(2)  +    a(1)        III GRUPPO
a(10)=    a(2)  +    a(2)
a(11)=    a(2)  +    a(3)
-----------------------------------------------
a(12)=    a(3)  +    a(0)
a(13)=    a(3)  +    a(1)        IV GRUPPO
a(14)=    a(3)  +    a(2)
a(15)=    a(3)  +    a(3)
```

Mad Thinker

Crypt::FNA & MySQL

Definition of {F} fractal curves

The group that is the direction k-th :

$$G(k) = int(k/Ro)$$

The location of the k-th group is rather :

$$P(k) = k\text{-}int(k/Ro) = k\text{-}G(k)$$

In conclusion, the value of the k-th direction is :

$$\mathbf{a(k)=a(G(k)) + a(P(k))}$$

Note that this equation is general and independent of the number of parameters under the curve: the fractal Koch have a base of cardinality equal to 4 but with this formula we can express the basis of curves with a desired cardinality

Crypt::FNA & MySQL

Definition of {F} fractal curves

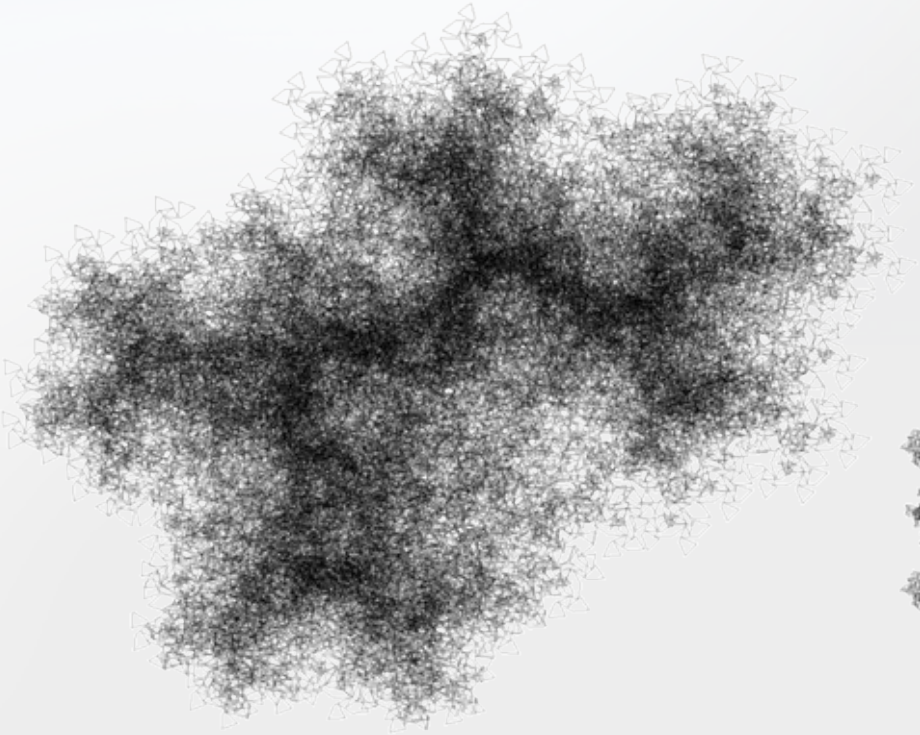
With this equation is very simply plot the curve, being able to calculate the direction of successive approximants segments and then implementing a system of turtle graphics for hes chart:

```
while ($k<$Ro**$r) {
    $a [$k]=$a[int($k/$Ro)]+$a[$k-int($k/$Ro)];
    $k++
}
```

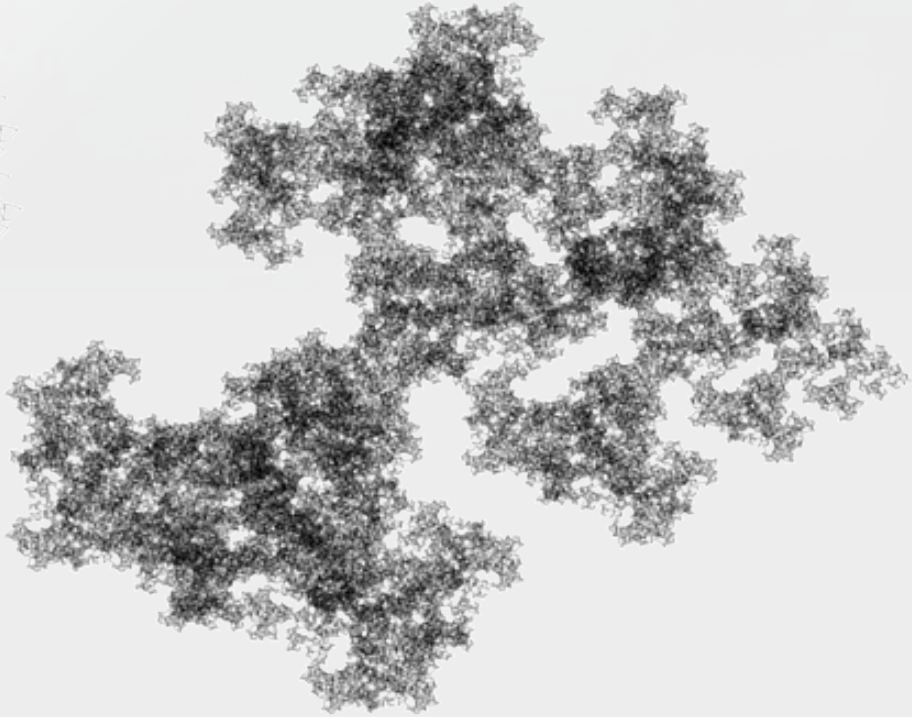
Next, many curves ∈ {F}

Crypt::FNA & MySQL

Drawing {F} with make_fract method



(56,-187, 215, 64)

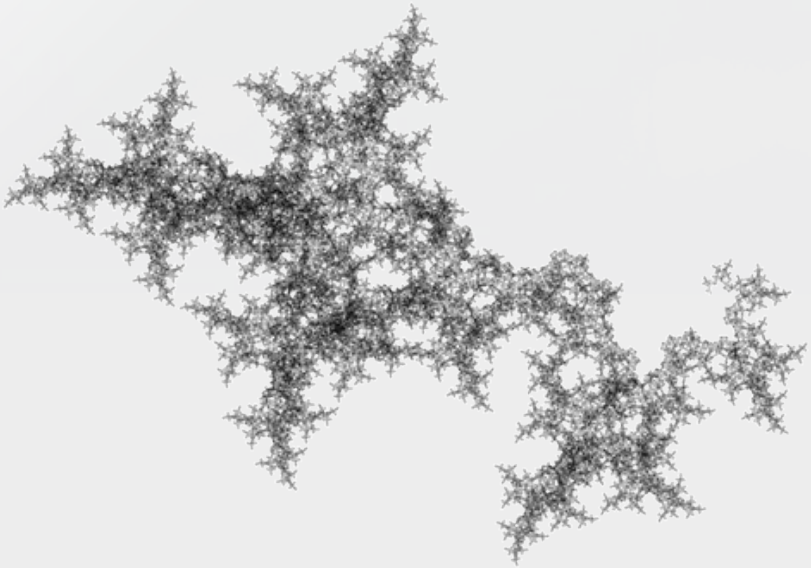


(0,90,-60,-90,60)

Crypt::FNA & MySQL

Drawing {F} with make_fract method

(56,-177,225,-164)

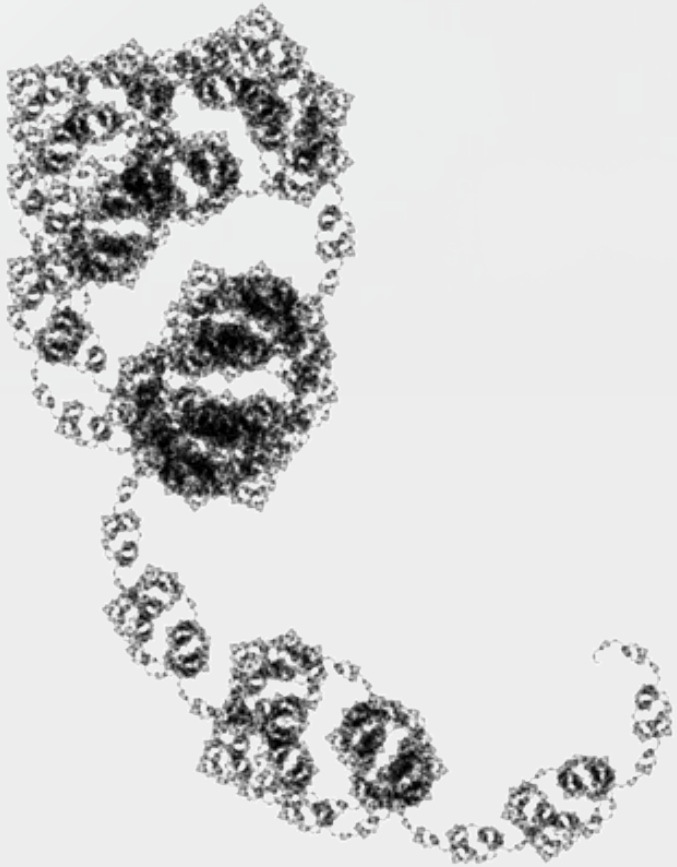


(56,-77,215,-64,60)

Crypt::FNA & MySQL

Drawing {F} with make_fract method

(0,90,0,-90)

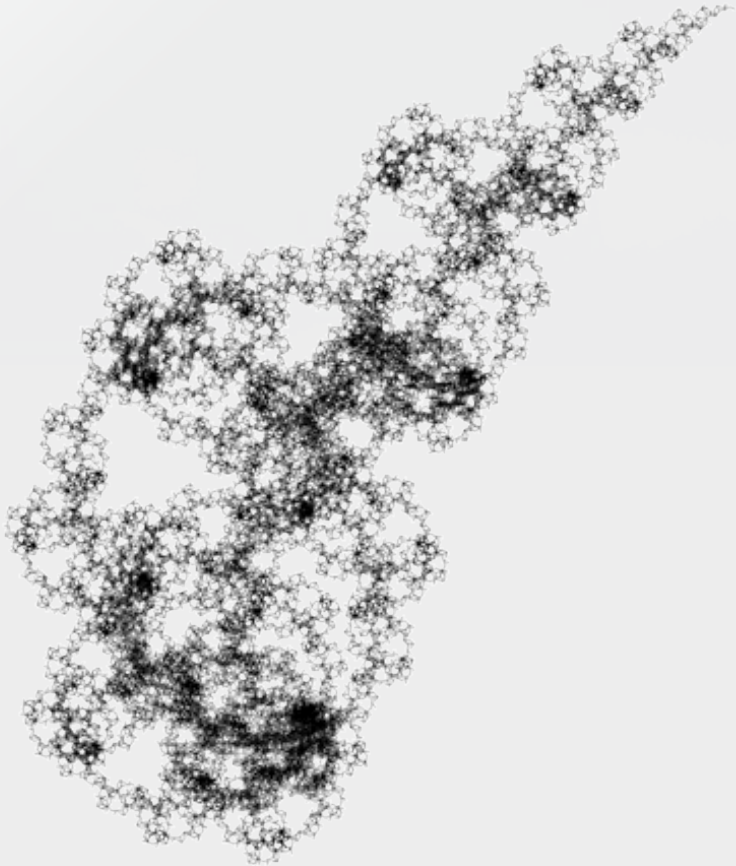


(0,90,60,-90,120)

Crypt::FNA & MySQL

Drawing {F} with make_fract method

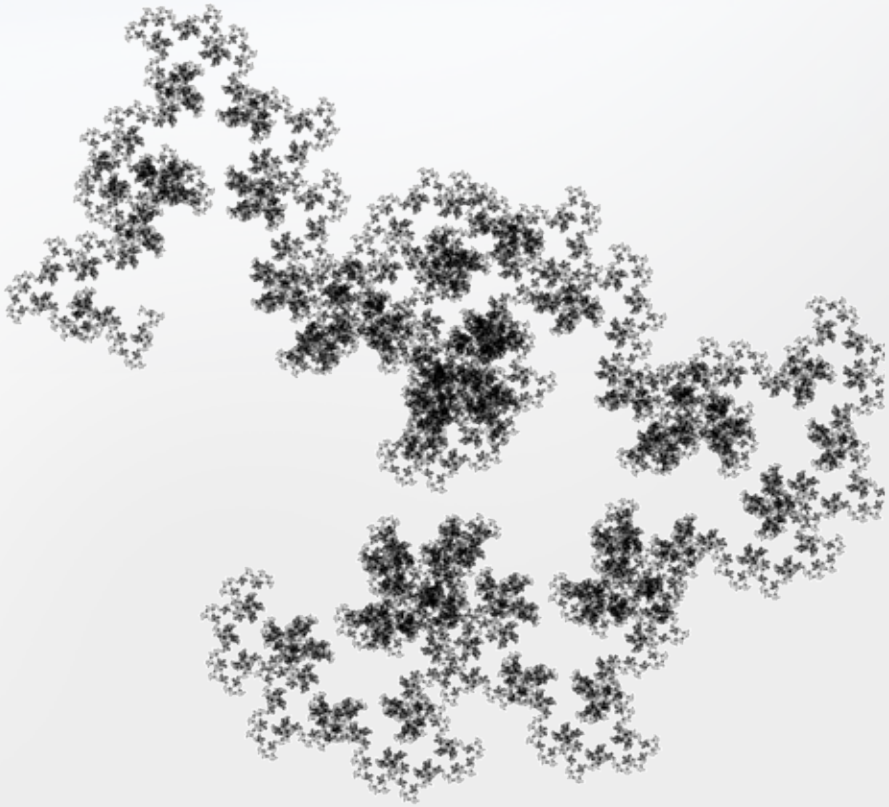
(56,-177,225,164)



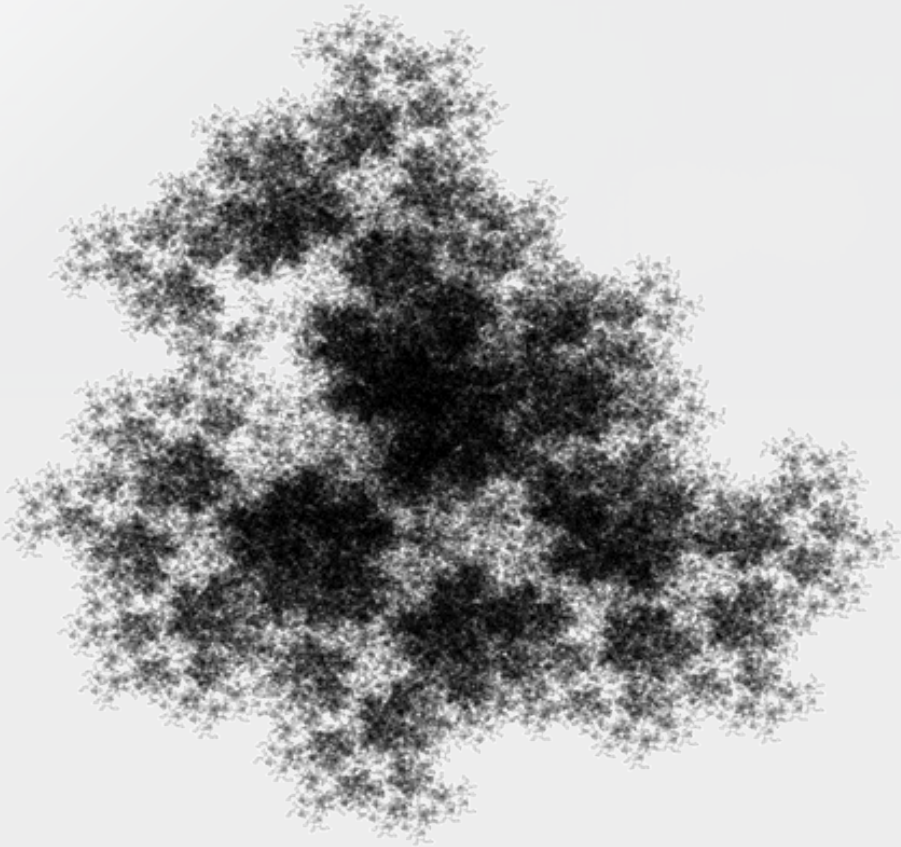
(21,-31,100,-79)

Crypt::FNA & MySQL

Drawing {F} with make_fract method



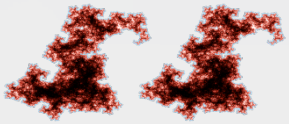
(56,-67,215,-64,60,45)



(56,-67,210,-64,60,70)

988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

SECTION II



Crypt::FNA methods & attributes

Theory

1 Data is stored in bytes: any type of files we open, its content is certainly a precise sequence of bytes.

A byte is 8 bits, its value must belong to all the integers between 0 and 255 (256 items total).Then follow the curve fractal choice of the set (F), for a number of vertices equal to the value of bytes to encrypt. The Cartesian coordinates of that summit are the cryptogram of that precise byte.

Theory

2 The curves {F} have a development that, in general, is known only computers but it can be calculated if we know the parameters Ro parents are elementary parts of the key: it is at the heart of this cryptosystem.

Like other symmetric encryption systems, such as DES and AES, FNA has a secret key, but unlike those of the Data Encryption Standard (which has a 56-bit key) and Advanced Encryption Standard (which has a key range between 128 and 256-bit), **Fractal Numerical Algorithm has a key in bits as long as you want**: there are no restrictions on the number and values of the basic directions of Ro.

Theory

3 to encrypt

Each byte is encrypted using the coordinates of the top of fractal curve, obtained starting from the next than previously estimated, jumping a further number of vertices equal to the magic number increased the value of bytes to encrypt.

4 to decrypt

Follow the curve occurring fractal, from summit to summit, that the coordinates match those of the cryptogram. The value of the original byte is reconstructed having counted how many summits have succeeded for the equality of two values, the last equality found. The number of vertices, reduced the magic number added to the unit, represents the value of the nth byte.

Crypt::FNA & MySQL

Crypt::FNA methods & attributes

Crypt::FNA methods

Crypt::FNA->new

Crypt::FNA->make_fract

Crypt::FNA->encrypt_file
Crypt::FNA->decrypt_file

Crypt::FNA->encrypt_scalar
Crypt::FNA->decrypt_scalar

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD NEW

This method has two modes for the object instance:

```
my $krypto=FNA->new();

my $krypto=FNA->new(
        {
            r=> 7,
            angle =>  [56,-187, 215,-64],
            square => 4096,
            background => [255,255,255],
            foreground => [0,0,0],
            magic => 3
        }
```

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD NEW

"r" attribute: order of {F} curve

It's the depth in the calculation of the curve. It 's a number greater than zero, not necessarily integer. Indicated by the number of corners Ro basic self-similar structure, the number of segments forming the curve "complete" is given by *Ro ** r*

Default value: 7

"angle" attribute: direction of the segments of the {F} curve base

Are the angles covered by the recursion algorithm: these angles determines the basic structure of the self-similar curve (F). Angles are expressed in sessadecimale system, with values between -360 and 360 (or from 0 to 360).

Default value: (56,-187, 215,-64)

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD NEW

"square" attribute: side of the square where it will be designed/calculated {F} curve

It 's the length of the side of a square container of the curve. Square has not only important for the (possible) graphical representation, but also for encryption, because it is used to calculate the length of the side of the curve (*di* it's proportional to *square/Ro**r*)

Default value: 4096

"background" attribute: background color PNG {F}

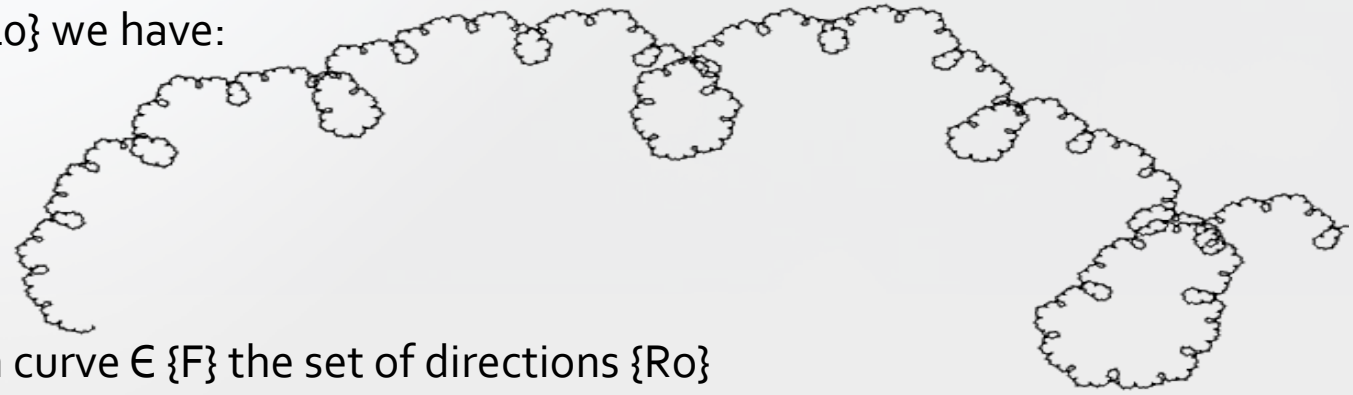
It's the RGB color of the bottom of the PNG file containing the design curve. The notation is decimal, with values ranging from 0 to 255.

Default value: (255,255,255)

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: ANGLE ATTRIBUTE: lacci di {F}

There is a non-zero probability that two (and thus infinite) vertices can overlap, making it impossible to decode the encrypted file. For example, base Ro={-30,60,45,110} we have:



Definition 1

It defines the basis of a curve Є {F} the set of directions {Ro}

Definition 2

It defines branch of a curve Є {F}, the broken line calculated by the algorithm of construction applied on the base {Ro}

Theorem

Hypothesis: is given the base of {F}, Ro={x1, x2,..., xn} : max(x)-min(x) < π/4

Thesis: the hypothesis is sufficient for all points of the curve (F) is in correspondence with a subset of points of the plan

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD NEW

"foreground" attributes: foreground color for drawing the curve (F)

RGB color tract in the PNG file containing the design curve. The notation is decimal, with values ranging from 0 to 255.

Default value: (0,0,0)

"magic number" attribute: for a discrete encryption

Indicates the number of vertices of the curve to be skipped during encryption and decryption: the algorithm, being a continuous function on the vertices, skipping some, this becomes a function on isolated points (hence "discrete").

Default value: 3

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD MAKE_FRACT

This method is undoubtedly the most impressive and can "touch" the curves are then used in cryptographic algorithms.

The graphics file output format is PNG (Portable Network Graphic), accessible from any browser as the most diverse graphics software.
The syntax is:

$krypto->make_fract($pngfile,$zoom)

$pngfile it's the name of png file – without "png" extension, automatically inserted.

$zoom is the drawing scale - greater than zero. Default value: 1

The image produced is contained in the square of side *"square"*.

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD ENCRYPT_FILE

The methods encrypt_file and decrypt_file, are summa: make profit, by applying the mathematics of {F} curves. This method encrypts the plain input file to output file.

Syntax:

$krypto->encrypt_file($name_plain_file,$name_encrypted_file)

The input file of any format, is read and encrypted, through the {F} curve.

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD ENCRYPT_FILE

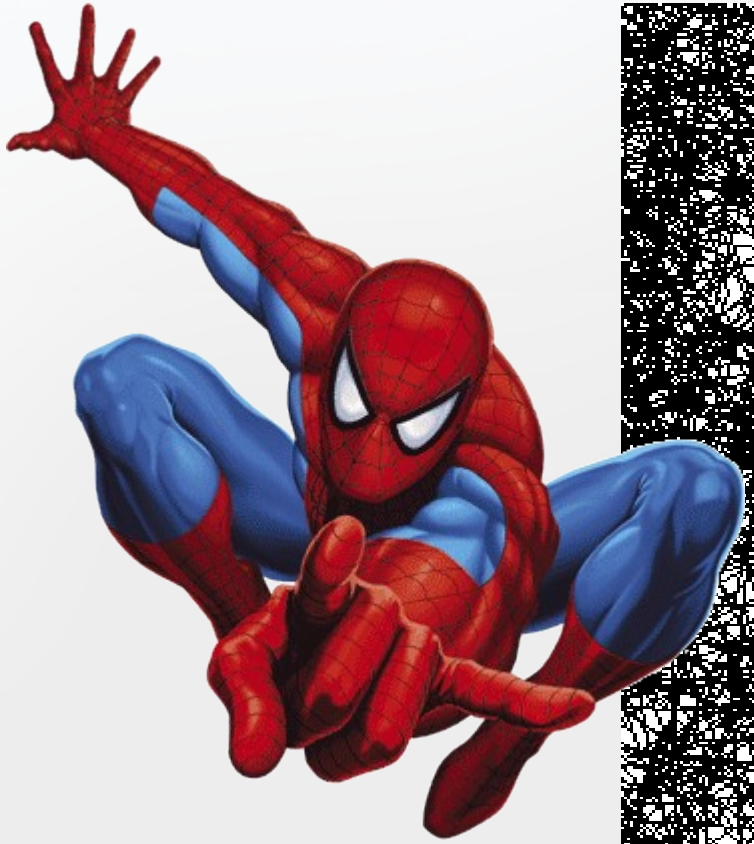
Here the appearance of an encrypted FNA file

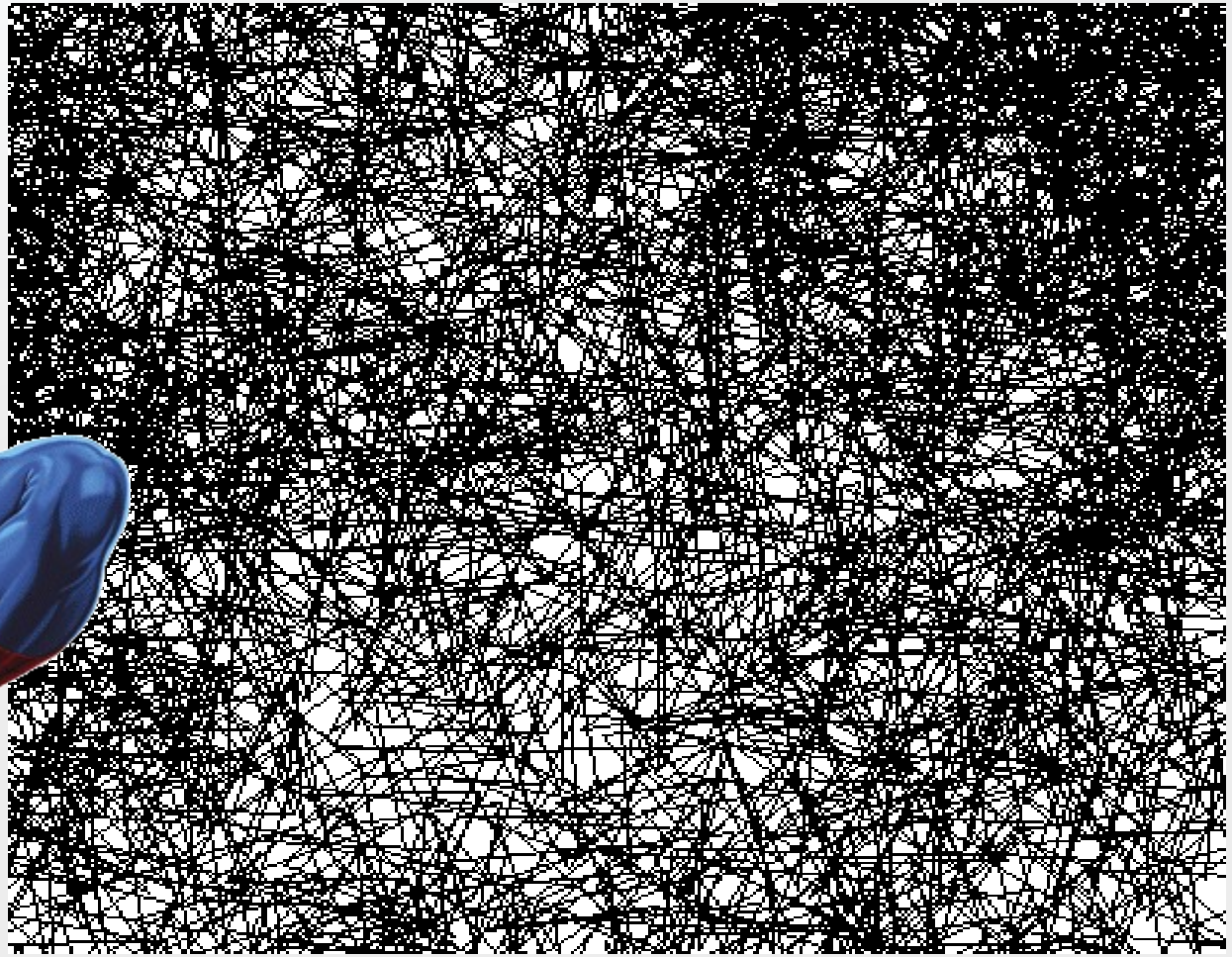
-806.16701617 4296.950584 -1163.3897453 4378.30613408 -1253.81513894 4361.33265404 -1502.80711437
4636.89514523 -1371.10557976 4745.56050632 -1230.07749379 4968.48069209 -1338.39851924
5248.88785964 -917.21821497 5429.36645491 -773.44592091 5696.62911696 -692.72801005
5885.46154004 -988.27897105 5885.418198 -1248.99379997 6171.71101067 -830.48330143 6377.55135044
-768.07453852 6493.40995382 -290.38619797 6703.79926248 -101.38261857 6641.39653224 329.01095794
6547.35282987 491.23460593 6672.15350589 682.15153937 6767.07332641 951.17643798 7125.45527124
844.47157379 7301.13742586 616.45930112 7293.99200882 844.26353513 7262.78340711 1211.3200562
7315.25004987 1474.41515451 7121.21394711 1951.75973992 7224.47233263 2176.20365976 6962.04147204
2547.88708591 6998.13655185 2781.82594976 6972.85084038 3056.52905252 7371.28466715
3037.53030053 7569.06437014 3048.49593738 7320.32093005 3389.66342779 7357.81470144 3676.23526579
7708.87987244 3755.43863759 7814.8354795 3435.5290489 8296.58426972 3441.10117125 8627.97877198
3412.2773365 8623.6058585 3362.87465115 8767.32280898 3260.65143202 8583.97947961 2890.71868372
8474.68032897 2726.83436885 8650.05588533 2718.8481018 9045.95222039 2669.00976899
9254.66114943 2644.06562016 9103.68182141 3127.66020707 9113.43039278 3191.47856428
9188.88465234 3207.82184971 9202.57034881 3478.33454467 8945.6121183 3832.00806714 8945.62804071
4080.86384299 9320.62189286 4289.2595779 9439.78195562 4021.13116501 9644.36385638
4311.34336432 9554.3477728 4679.21568268 9563.22563256 4833.53132591 9641.37582295 4740.32174942
9910.49435765 4448.89751812 10157.37473936 4273.26989922 10265.73224722 4218.00573474
10553.33210292 4076.79496626 10732.34891747 3830.35537312 10613.81591903 3785.18217462
10386.70855427 3666.99726881 10332.12423113 3476.25444621 10694.76481321 3296.35920314
10804.77625983 3060.88089069 11346.01346391 3007.91070428 11444.10666595 2765.46825422
11911.74931522 2771.84792598 12217.75488876 2730.08778903 12432.33422506 2649.22698242
12307.67655488 2179.40416992 12145.89439835 2279.94226546 12105.79701773 2047.78623478
12604.70024151 2134.4739565 12762.57334939 1895.30449332 12619.14996241 1526.25794611
12313.79872918 1561.04359063 12060.9258984 1204.52077789 11904.48474151 1011.49806809
11625.32850092 896.84643331 11430.88088124 1209.72754463 11427.67243264 1445.63793588
11243.03320502 1007.30448881

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD ENCRYPT_FILE

And here is the file smoothie: without the key is very difficult rebuilding it





Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD DECRYPT_FILE

The methods encrypt_file and decrypt_file, are summa: make profit, by applying the mathematics of {F} curves. This method decrypt the crypted input file to output plain file.

Syntax:

```
$krypto->decrypt_file($name_encrypted_file,$name_decrypted_file)
```

The input file (FNA format), is read and decrypted, through the {F} curve.

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD ENCRYPT_SCALAR

The method encrypt_scalar encrypt strings: the result of encryption is a vector containing the cryptogram.

Syntax:

@encrypted_scalar=$krypto->encrypt_scalar($this_string)

The developer that provides to save FNA encrypted password, must implement what is called "salt". A "salt" is a string, usually random, adding to the data to be encrypted, so that a brute force dictionary to not produce results.

Crypt::FNA & MySQL

Crypt::FNA methods & attributes: METHOD DECRYPT_SCALAR

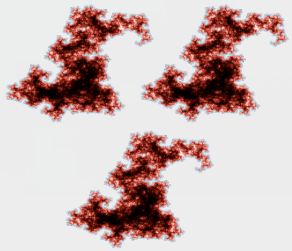
"decrypt_scalar" method, reconstructs clear data from the cryptogram created with encrypt_scalar.

Syntax:

```
@decrypted_scalar=$krypto->decrypt_scalar(@encrypted_scalar)
```

988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

SECTION III



Attack to FNA

Crypt::FNA & MySQL

Attack to FNA

FNA is a special encryption system, based on the replacement of bytes / characters with complex numbers, (*n* ordered number, in this case the pair of coordinates) through the algorithm of fractal generator {F}. The transformation generally takes place by replacing the ordinal value in his alphabet, with the coordinates of a vertex of the curve.

FNA can be considered a particular polyalphabetic, whose peculiarity it's in the fact of having a virtually unlimited number of alphabets, this is because each encoding depends directly from all previous and following "the butterfly effect." Referring explicitly to deterministic chaos, a small variation in the data in clear, produces large differences in the resulting ciphertext.

Basically, what happens is that each byte encryption, affects the coding sequence of possible cryptograms, because it restarts from zero (the number of cryptograms depends on cardinality of the alphabet the clear data) what is interesting is that for any cryptographic sequence of 256 vertices of curve (F) (in case you encrypt bytes), the alphabet used to encrypt that particular byte is different from the previous and the value of bytes to encrypt influence subsequent alphabets.

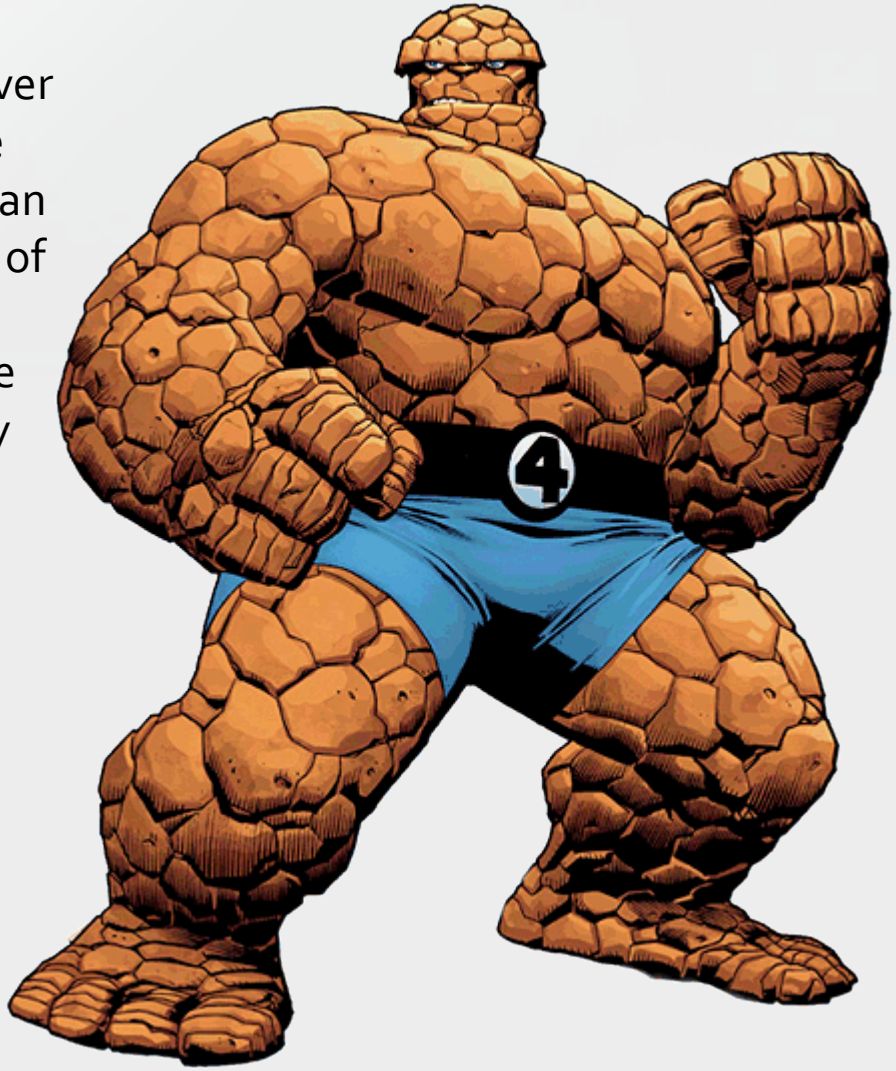
Crypt::FNA & MySQL

Attack to FNA

FNA key, strictly, is given by directions of initialization **Ro**, **di** and the **magic number**. However observed it as polyalphabetic, and considering the fractal algorithm as a generator of alphabets We can say that, in that point of view, has the advantages of a key long of clear data and random (in the sense that it is significantly irregular, non-linear) as in the case of disposable block cipher. Also has a virtually unlimited number of cipher alphabets.

Also does not suffer from inherent difficulties in applying the system to lock disposable and can be implemented very simply, **hypercryptography** operations (encrypt an already encrypted data).

See why, in my opinion, this observation is legitimate:



Alphabet

The alphabet used is long and variable and have length, at least, as the cardinality of the alphabet of the clear data. In the case of bytes is at the 256 pairs of coordinates of vertices. Alphabets are also apparently random, since the succession of vertices, branch fractal, is irregular and each byte influence, significantly, the subsequent alphabets.

Once encrypted one byte, it proceeds to encrypt the following: the alphabet "re-start", because once encrypted the byte, the coordinates of the next vertex of (F), is the first symbol of new alphabet of cardinality, at most, equal to the cardinality of the alphabet with which it is expressed as to be encrypted.

All following alphabet are always different and depend, strictly, on all previously encrypted plaintext data.

Key

The key, seen as the ordinal sequence of alphabets used, is as long as the message:

key 1, first data to encrypt: firts alphabet -> next alphabet depend on this
key 2, second data to encrypt: second alphabet -> next alphabet depend on this
...
key n-1, n-1 data to encrypt: n-1 th alphabet -> n-th alphabet depend on this

In this point of view, We have therefore a key to encrypt the data, long how plaintext and a number of alphabets equal to number of parts which the plaintext.

Crypt::FNA & MySQL

Attack to FNA

Note that any **brute force attack** would take more time than necessary to the heat death of our Universe.

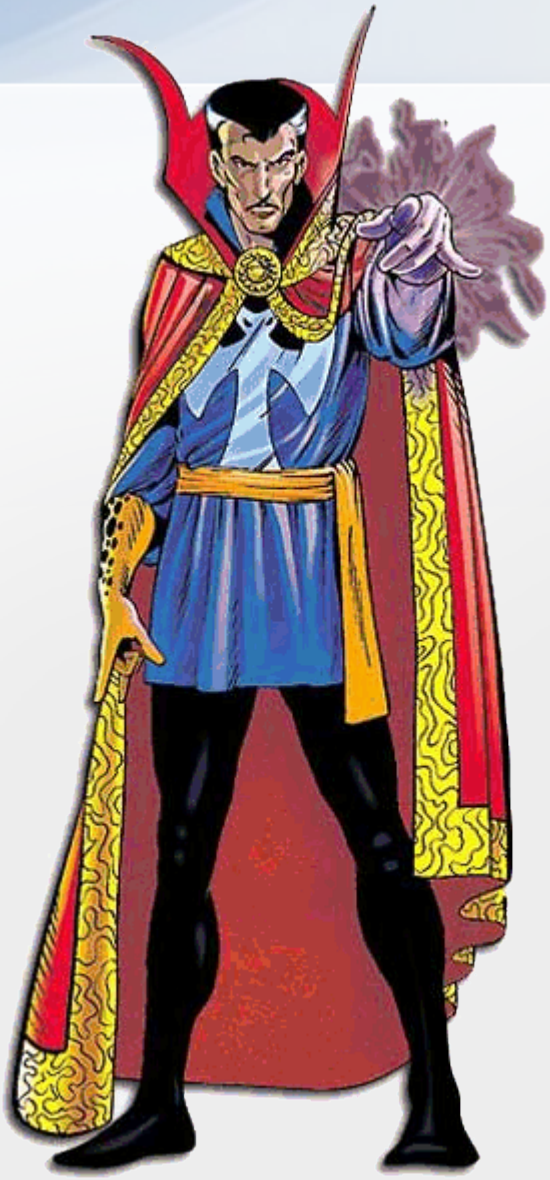
If we assume a number of basic directions of ro = 3, however we do not have any idea on his values. Consider that the value of an angle is a number between 0 and 360, whereas not We have no idea how many decimal places were used for those directions. Had eight decimal number for directions we have:
(99'999'999 * 360)**3 =
= 46655998600320013996799953344000 possibility

If we could see a combination per second (very optimistic assumption), it would take a number of years equal to:
1 479 452 010 410 959 347 945 204

And if directions were 4? Omit the calculation...

Crypt::FNA & MySQL

Attack to FNA



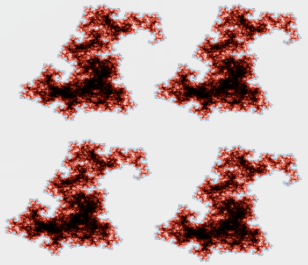
Also there are other variables, such as the **magic number**, which make it very difficult to identify the next corner (in an attempt to discover the base) over the order of the curve where you go to encrypt.

We also consider the possibility of **hypercryptography** that increase exponentially the difficulty from having to identify combinations, as seen, even with a few Ro direction based...

988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

Crypt::FNA & MySQL

SECTION IV



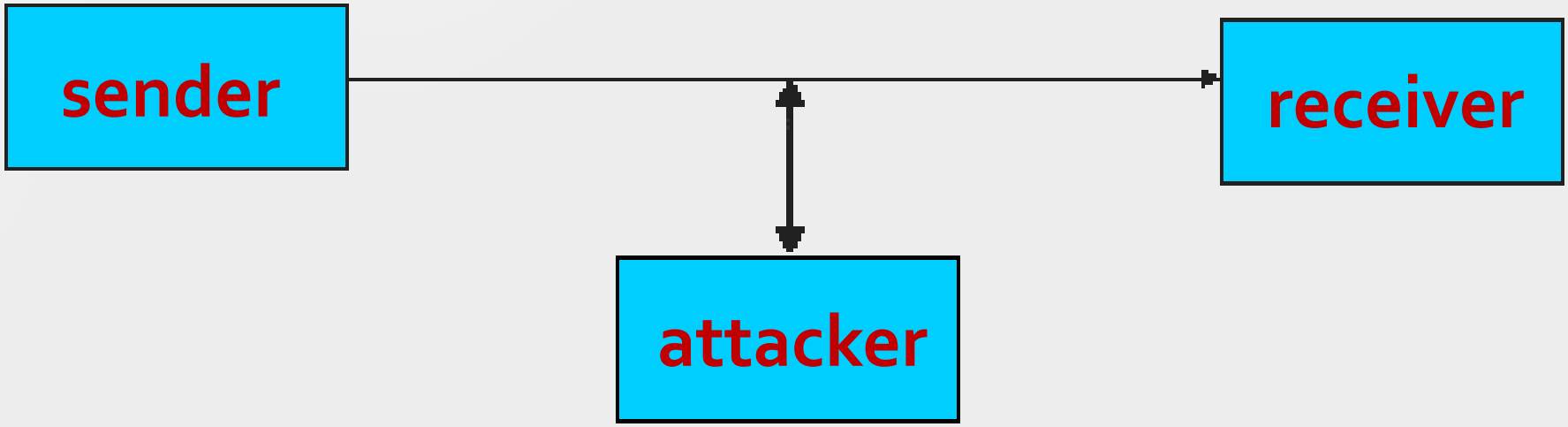
Applications

Crypt::FNA & MySQL

Applications: MAC implementation

MAC -> Message Authentication Code

Message authentication ensures the integrity of information even in the presence of an active adversary that sends data meaningful



Crypt::FNA & MySQL

Applications: MAC implementation

MAC -> Message Authentication Code

Said:

K FNA key selected for the authentication

A Application of FNA algorithm on Sender data

V Application of FNA algorithm on Receiver data

m data to authenticate

Compute

$\mathbf{A_k(m)}$ – coordinates last FNA vertex

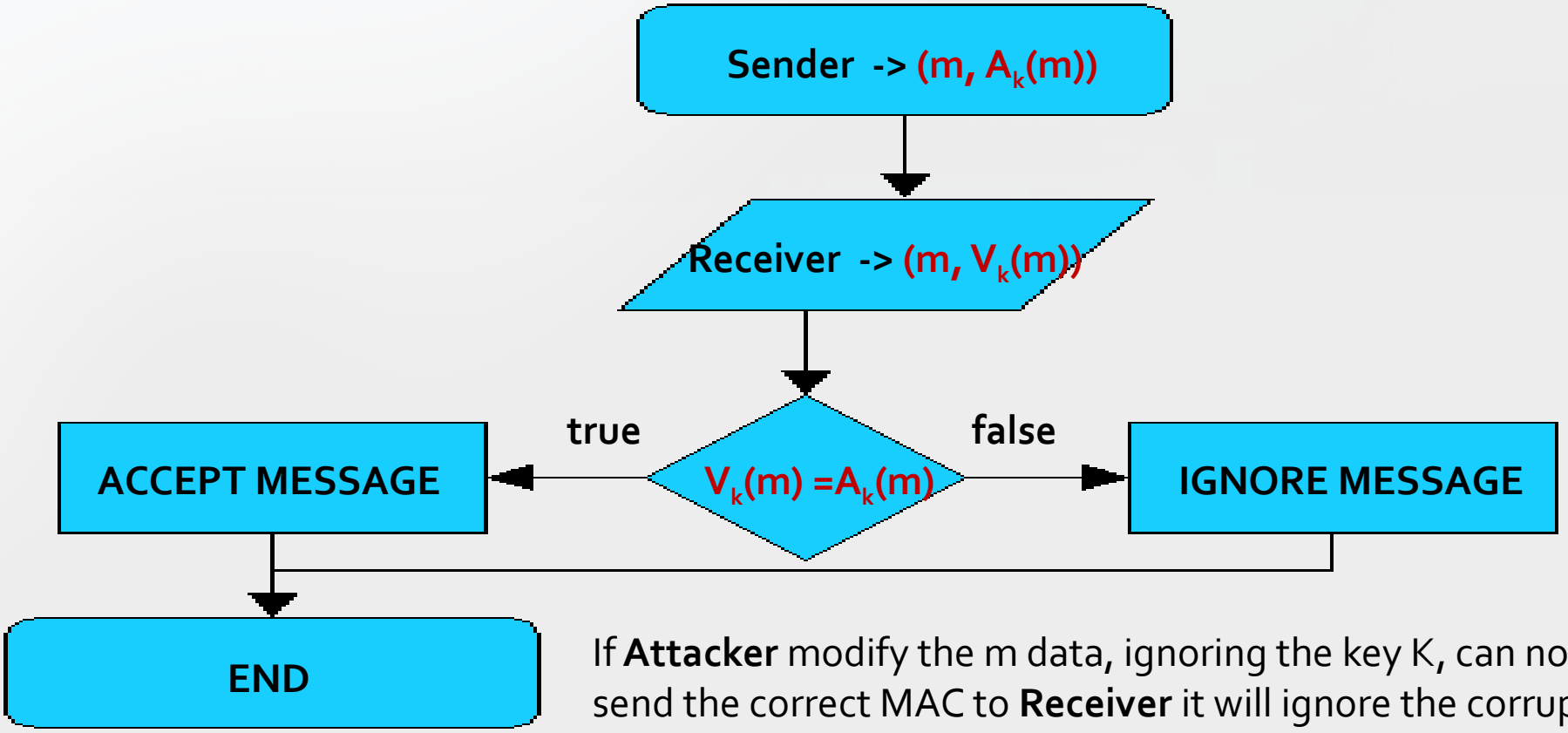
1. **Sender** send to **Receiver** a pair $\mathbf{(m, A_k(m))}$

2. **Receiver** – know **K** and have received "his" **m** – compute $\mathbf{V_k(m)}$

Crypt::FNA & MySQL

Applications: MAC implementation

MAC -> Message Authentication Code



If **Attacker** modify the m data, ignoring the key K, can not send the correct MAC to **Receiver** it will ignore the corrupt message.

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

Target: develop a client in Perl to interact with an encrypted database

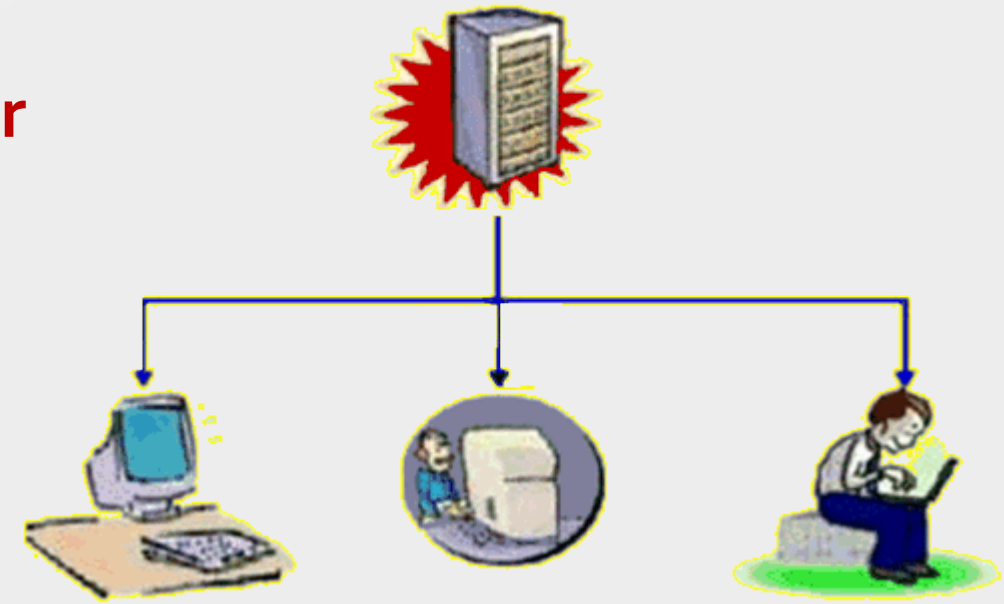
Example

Requirements:

MySQL server

DBI

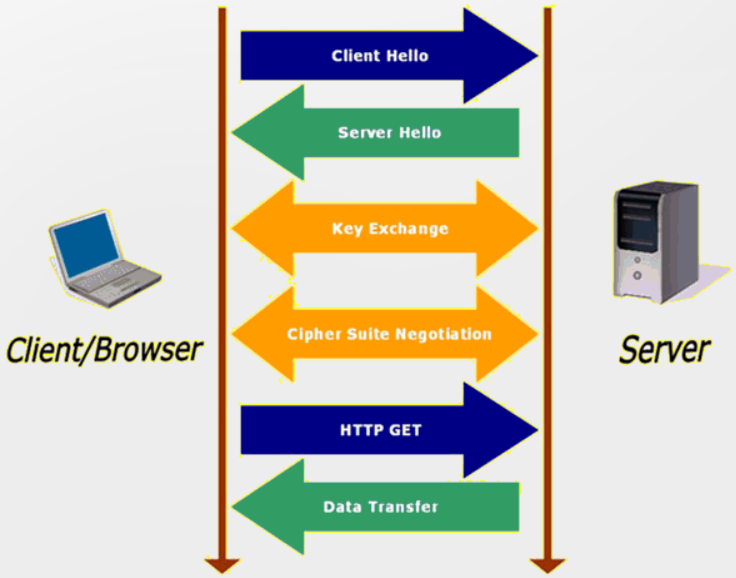
Crypt::FNA



Crypt::FNA & MySQL

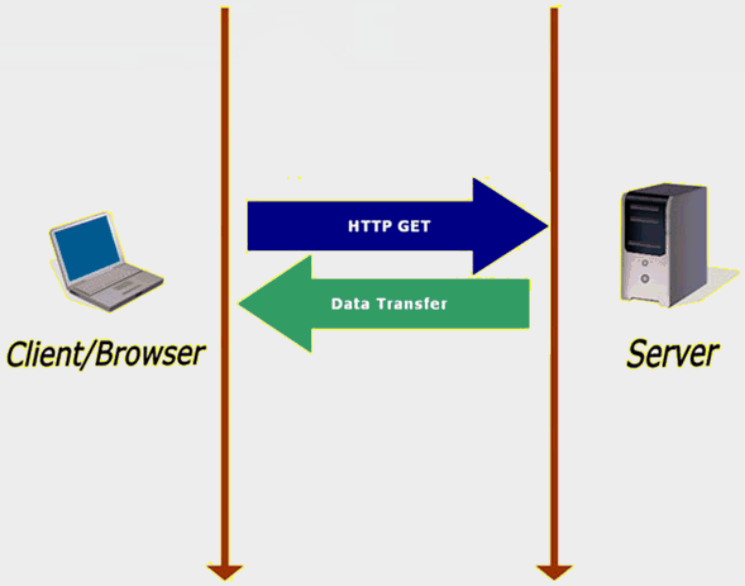
Applications: how to make a fractal encrypted database

In a typical encrypted connection, such as SSL, the pattern is as follows



SSL

The situation with FNA significantly different, and the number of steps is less

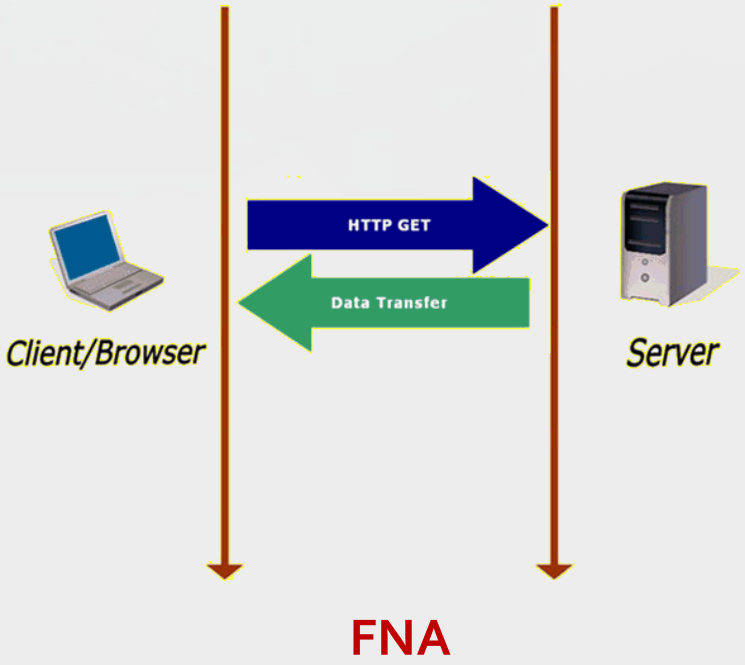


FNA

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

The difference is that :

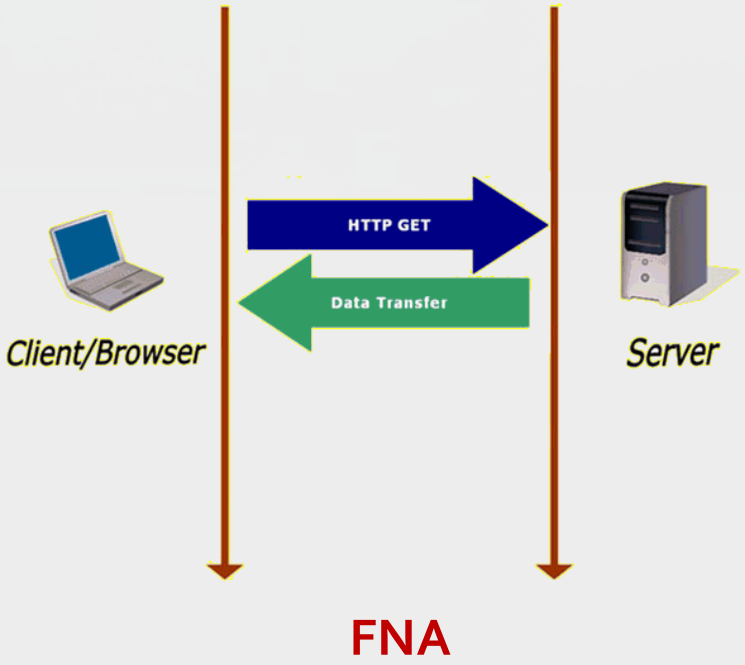


Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

The difference is that :

with **SSL**, the **communication** is **encrypted** but the **database** on server **is plain**



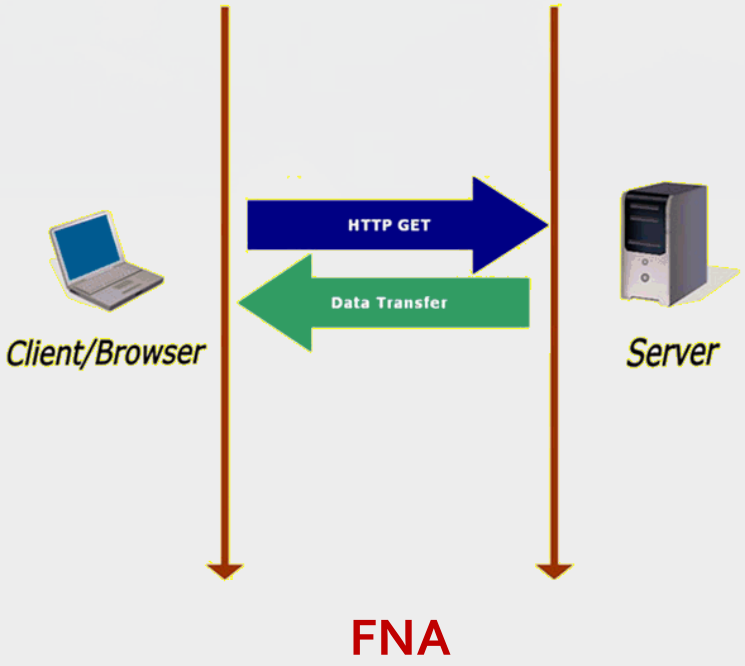
Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

The difference is that :

with **SSL**, the **communication** is **encrypted** but the **database** on server **is plain**

with **FNA**, the **communication** is **plain** but the **data is encrypted**



Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

Example with DBI on MySQL

CONNECTION

```
use Crypt::FNA;
use DBI;

my $krypto=Crypt::FNA->new();

my $user="anakadmin";
my $pass="thewolf";

my $dbname="dataCrypted";
my $host="anak_mob";

my $dsn="DBI:mysql:database=$dbname;host=$host";
my $dbh=DBI->connect($dsn,$user,$pass,{RaiseError => 1 }) or
die (
      "Couldn't connect to database: ".DBI->errstr
);
$dbh->{AutoCommit} = 0;
```

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

WRITING

Suppose We want to populate a database table using data collected from a CSV file (or other database in the clear - is an example to show writing), containing customers data

```
# prepare data insert
	my $sth = $dbh->prepare(q
		{
			INSERT INTO customers (name,surname,email) VALUES (?,?,?)
		}
	) or die $dbh->errstr;

# end prepare data insert
```

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

WRITING

```
my $fh_csv;
my $csv_filename="customers.csv";

open $fh_csv,'<',$csv_filename or die "Unable to open $csv_filename: $!";
while (!eof($fh_csv)) {
        my ($name,$surname,$email)=split(/,/,<$fh_csv>);

        my @encrypted_name=$krypto->encrypt_scalar($name);
        my $encrypted_name=join('\n',@encrypted_name);

        my @encrypted_surname=$krypto->encrypt_scalar($surname);
        my $encrypted_surname=join('\n',@encrypted_surname);

        my @encrypted_email=$krypto->encrypt_scalar($email);
        my $encrypted_email=join('\n',@encrypted_email);

        $sth->execute($encrypted_name,$encrypted_surname,$encrypted_email)
}
close $fh_csv;
$dbh->commit;
```

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

READING

It will make accessing the database and decrypt values

```
my $sth = $dbh->prepare(q
        {
                SELECT * FROM customers
        }
) or die $dbh->errstr;

$sth->execute;

while (my $row=$sth->fetchrow_hashref()) {
        my $name=$krypto->decrypt_scalar($row->{name});
        my $surname=$krypto->decrypt_scalar($row->{surname});
        my $email=$krypto->decrypt_scalar($row->{email})
}
```

☑	✎	✕	11	1042 09332726\n711.9962719\n989.19358468\n510.4162...	1092.5021146\r725.54587292\n1005.71791345\n437.404...	1155.90607766\n638.030305\n1120.3619648\n319.48
☐	✎	✕	12	979.55459067\n742.85730367\n988.94492988\n496.1710...	1106.53302732\n725.42341784\n971.03446438\n455.805...	1134.91186603\n676.98167152\n1089.48277823\n41
☑	✎	✕	13	1004 7571464\n754.67828303\n1003 25885566\n516.974...	1014.41942614\n734.20786854\n966.90128026\n492.120...	1143.30834969\n650.35139132\n1064.27613789\n413
☐	✎	✕	14	1004 7571464\n754.67828303\n988.94492988\n496.1710...	1029.22724181\n720.67456606\n994.26206815\n482.961...	1155.90607766\n638.030305\n1091.37483662\n316.2
☐	✎	✕	15	1068 78512953\n709.84796939\n966.90128026\n492.120...	1068.78512953\n709.84796939\n980.21014663\n485.189...	1155.90607766\n638.030305\n1108.27954308\n327.0

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

CONCLUSIONS

1. The rate turns out to be good, especially without complication by the server because the encryption and decryption process is borne by the client. From a viewpoint of data size increases substantially, since each byte is to occupy about 16 (the coordinates of points). Of course you could implement an algorithm for compression / decompression that would significantly limit the size.

2. An interesting application is saving encrypted files in LONGTEXT type fields (for MySQL): pictures, executable, spreadsheets, etc.

3. Network communication between the client and the server will be encrypted data, so any communication channels also "clear" will not lead to the revelation of the contents

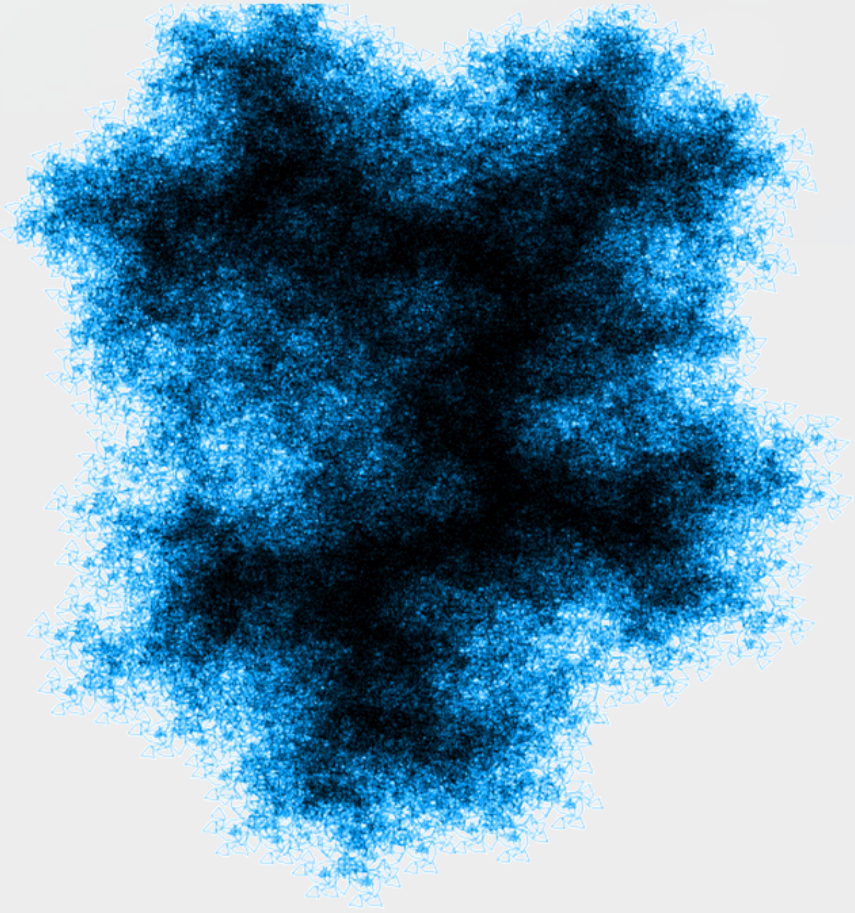
    1. In this way the possible violation of the server and reading of data by unauthorized third parties will not lead to dissemination of information, because the keys are remote to the server.

Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

CONCLUSIONS

- Here was used DBI for example, but you can use the interface that you want if you **encrypted before** each **write** operatione **and decrypt after** any **read** operation



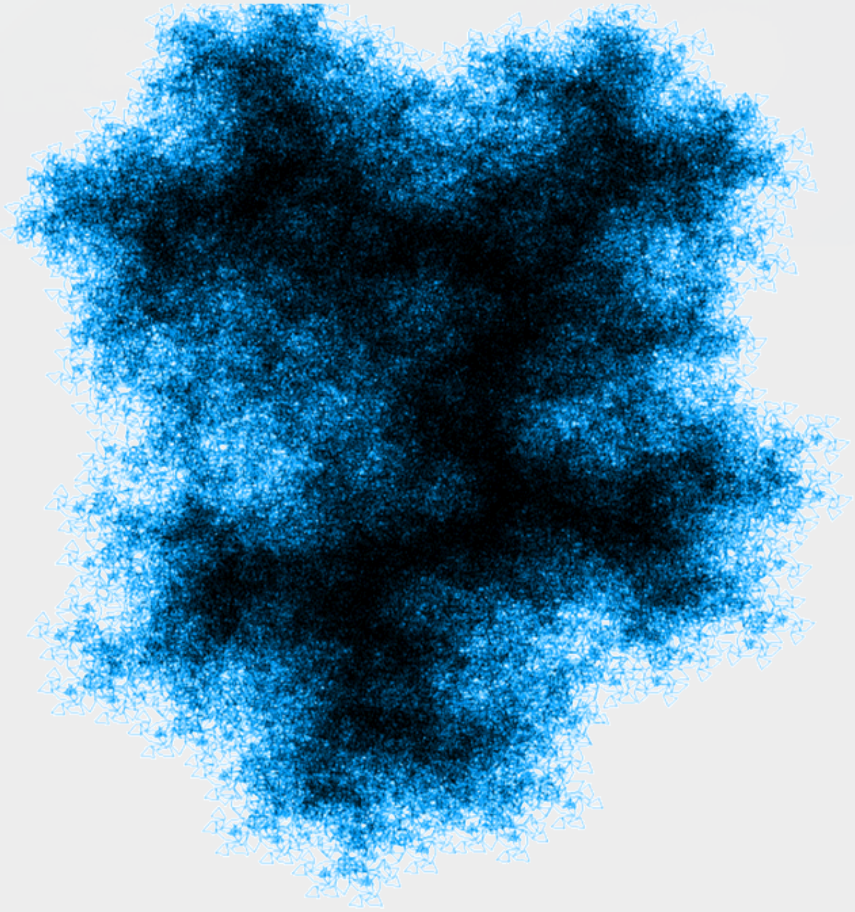
Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

CONCLUSIONS

- Here was used DBI for example, but you can use the interface that you want if you **encrypted before** each **write** operatione **and decrypt after** any **read** operation

- In any case I remember that FNA is an algorithm very young and still not adequately tested, so I would not save data to access a server containing **credit cards** or the **recipe for pizza** only after careful testing, and recalling that Anak I had warned ☺



Crypt::FNA & MySQL

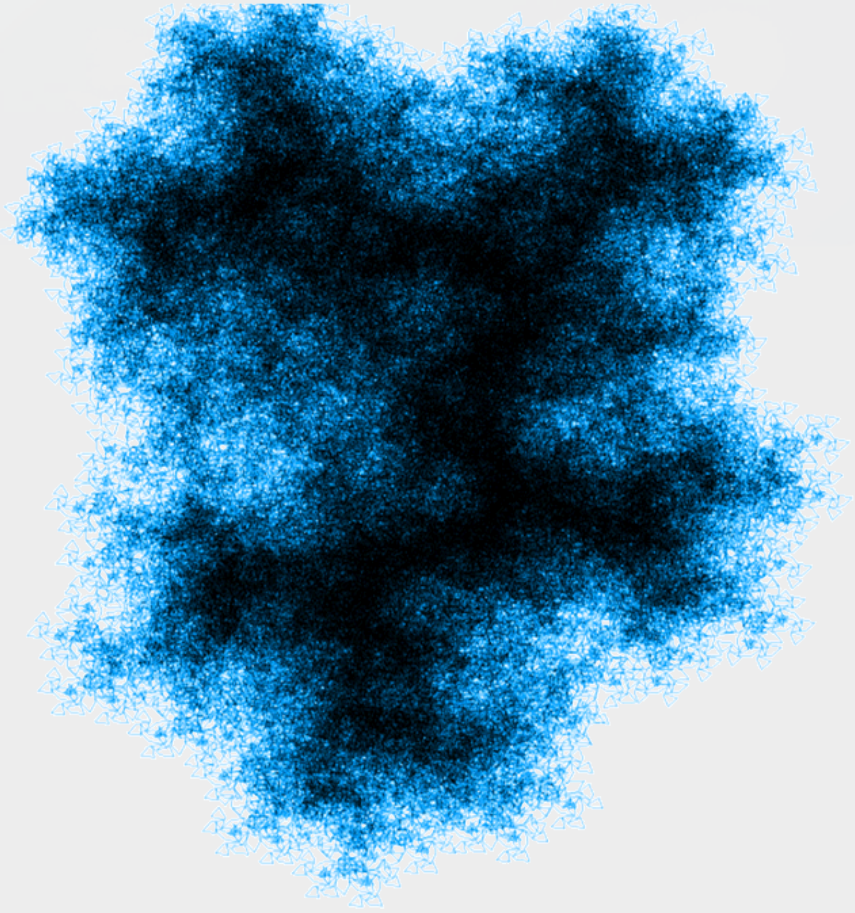
Applications: how to make a fractal encrypted database

CONCLUSIONS

- Here was used DBI for example, but you can use the interface that you want if you **encrypted before** each **write** operatione **and decrypt after** any **read** operation

- In any case I remember that FNA is an algorithm very young and still not adequately tested, so I would not save data to access a server containing **credit cards** or the **recipe for pizza** only after careful testing, and recalling that Anak I had warned ☺

- Among the future developments, the implementation of an asymmetric encryption system **aFNA** on the timing of which can not be precise because I am going to do in the little free time: apologize, not my main job although I would :/



Crypt::FNA & MySQL

Applications: how to make a fractal encrypted database

References

- CPAN : Crypt::FNA
  http://search.cpan.org/~anak/
- Article:
  http://www.perl.it/documenti/articoli/2010/04/anakryptfna.html

988.44587991
864.00639226
912.57423224
810.6751627
1004.9465835
856.86126075
1138.2745474
728.29664537
1079.07469229
695.76234167
930.351318
826.07044723
900.19866958
855.11766404
835.59726026
631.79907235
845.59106749
607.51514839
826.07053756
730.67818328
719.40812543
712.90110561
878.12528707
606.87693617
844.95287635
529.89996398
800.51021841
627.67375558
859.71016379
739.56662699
932.56196132

Crypt::FNA & MySQL

End talk – Thank'you all ☺