A New Cryptographic Algorithm

Ali Mahdoum

Retired, previously worked as a full-time researcher at the

"Centre de Développement des Technologies Avancées", Algiers, Algeria

ali.mahdoum@gmail.com

ORCID: 0000-0002-9430-195X

Abstract- The advent of quantum computing technology will compromise many of the current cryptographic algorithms, especially public-key cryptography, which is widely used to protect digital information. Most algorithms on which we depend are used worldwide in components of many different communications, processing, and storage systems. Once access to practical quantum computers becomes available, all public-key algorithms and associated protocols will be vulnerable to criminals, competitors, and other adversaries. It is critical to begin planning for the replacement of hardware, software, and services that use public-key algorithms now so that information is protected from future attacks." [1].

For this purpose, we have developed a new algorithm that contributes to deal with the aforementioned problem. Instead to use a classical scheme of encoding / decoding methods (keys, prime numbers, etc.), our algorithm is rather based on a combination of functions. Because the cardinality of the set of functions is infinite, it would be impossible for a third part (e.g. a hacker) to decode the secret information transmitted by the sender (Bob) to the receiver (Alice).

Author contribution information: All the tasks have been achieved by Ali Mahdoum

Conflict of Interest: The author declares that he has no conflict of interest.

A New Cryptographic Algorithm

Ali Mahdoum

Retired, previously worked as a full-time researcher at the

“Centre de Développement des Technologies Avancées”, Algiers, Algeria

ali.mahdoum@gmail.com

Abstract- The advent of quantum computing technology will compromise many of the current cryptographic algorithms, especially public-key cryptography, which is widely used to protect digital information. Most algorithms on which we depend are used worldwide in components of many different communications, processing, and storage systems. Once access to practical quantum computers becomes available, all public-key algorithms and associated protocols will be vulnerable to criminals, competitors, and other adversaries. It is critical to begin planning for the replacement of hardware, software, and services that use public-key algorithms now so that information is protected from future attacks.” [1].

For this purpose, we have developed a new algorithm that contributes to deal with the aforementioned problem. Instead to use a classical scheme of encoding / decoding methods (keys, prime numbers, etc.), our algorithm is rather based on a combination of functions. Because the cardinality of the set of functions is infinite, it would be impossible for a third part (e.g. a hacker) to decode the secret information transmitted by the sender (Bob) to the receiver (Alice).

1. Introduction

The aim of cryptography is to enable exchanging secure messages even in the presence of hackers. As our electronic networks grow increasingly open and interconnected, it is crucial to have strong, trusted cryptographic standards and guidelines, algorithms and encryption methods that provide a foundation for e-commerce transactions, mobile device conversations and other exchanges of data. The National Institute of Science and Technology (NIST) has fostered the development of cryptographic techniques and technology for 50 years through an open process which brings together industry, government, and academia to develop workable approaches to cryptographic protection that enable practical security [2]. Fig.1 shows the relation between FIPS 140-2 and CMVP & CAVP. Unfortunately, this strong procedure did not prevent cryptographic systems to be hacked even if they were FIPS 140-2 level 3 validated! This was the case for UTIMACO HSM (Fig.2) and YUBIKEY (Fig.3). Those attacks need stronger algorithms that aim at replacing the current set of public-key cryptographic algorithms so that to feature high resistance against both current and quantum computer-based attacks.

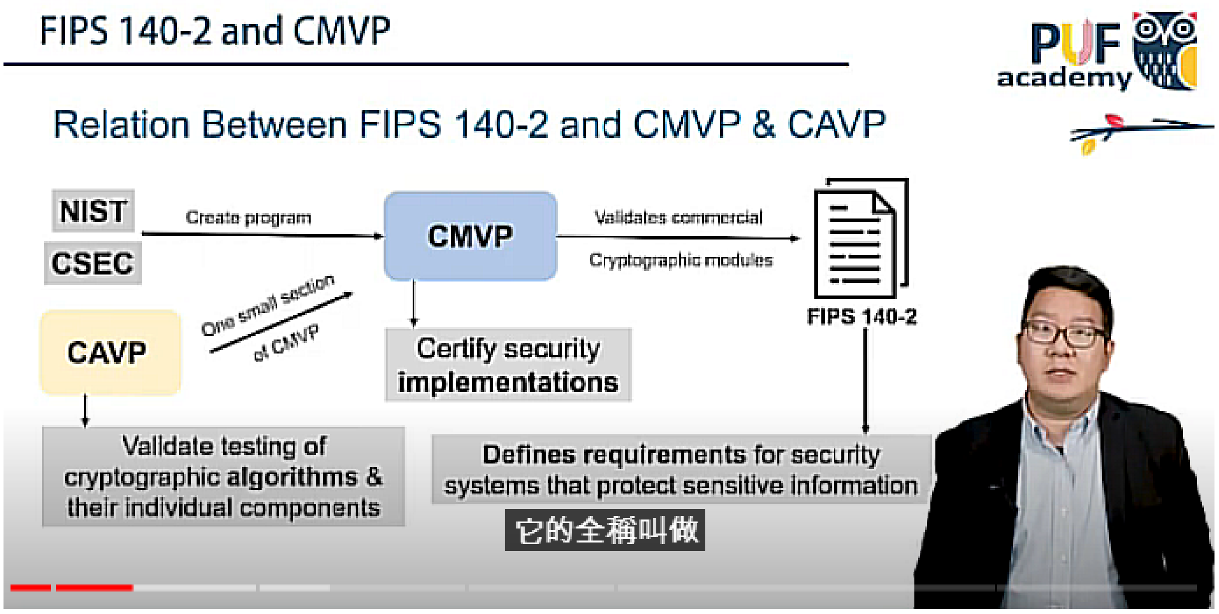
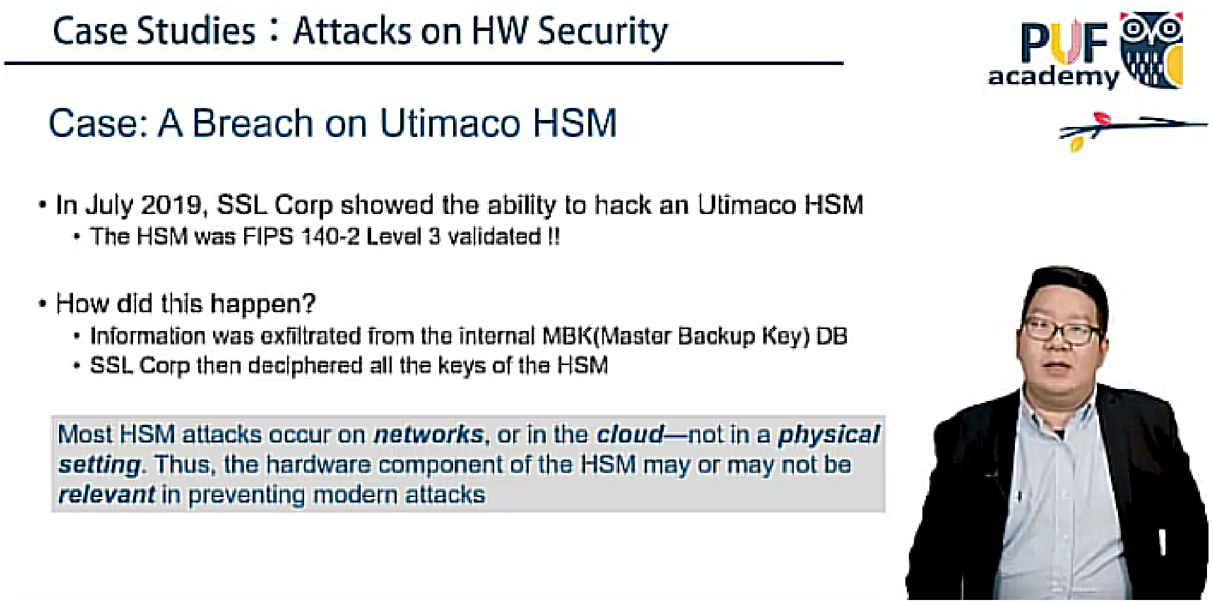

FIPS 140-2 and CMVP
Relation Between FIPS 140-2 and CMVP & CAVP
NIST
CSEC
Create program
CMVP
Validates commercial
Cryptographic modules
FIPS 140-2
CAVP
One small section of CMVP
Certify security implementations
Validate testing of cryptographic algorithms & their individual components
Defines requirements for security systems that protect sensitive information
它的全稱叫做
PUF academy

Fig.1. Relation between FIPS 140-2 and CMVP & CAVP (source: [3])



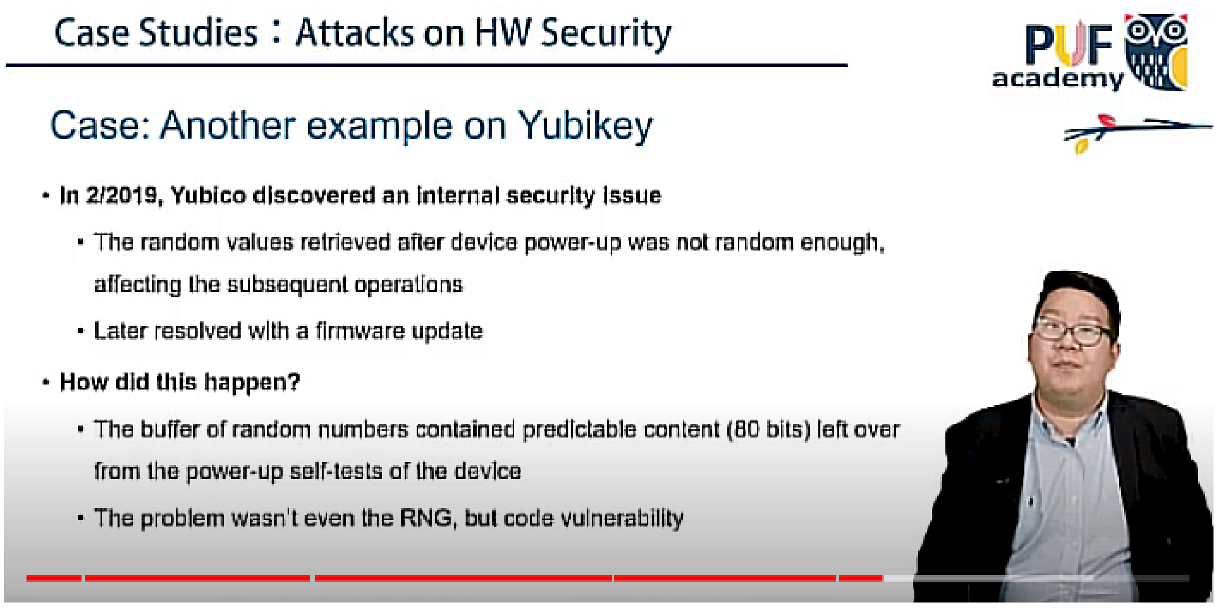
Case Studies：Attacks on HW Security

Case: A Breach on Utimaco HSM

- In July 2019, SSL Corp showed the ability to hack an Utimaco HSM
  - The HSM was FIPS 140-2 Level 3 validated !!
- How did this happen?
  - Information was exfiltrated from the internal MBK(Master Backup Key) DB
  - SSL Corp then deciphered all the keys of the HSM

Most HSM attacks occur on ***networks***, or in the ***cloud***—not in a ***physical setting***. Thus, the hardware component of the HSM may or may not be ***relevant*** in preventing modern attacks

Fig.2. Example1: attacks on Ultimaco HSM (source: [4])


Case Studies：Attacks on HW Security

Case: Another example on Yubikey

- In 2/2019, Yubico discovered an internal security issue
  - The random values retrieved after device power-up was not random enough, affecting the subsequent operations
  - Later resolved with a firmware update
- How did this happen?
  - The buffer of random numbers contained predictable content (80 bits) left over from the power-up self-tests of the device
  - The problem wasn't even the RNG, but code vulnerability

Fig.3. Example2: attacks on Yubikey (source: [4])

In this context, we have developed a new cryptographic algorithm that overcomes the use of a public key. Instead, it is rather based on a combination of functions. Because the cardinality of the set of functions is infinite, it would be impossible for a third part (e.g. a hacker) to decode the secret information transmitted by the sender (Bob) to the receiver (Alice). This will be shown in the next section. In section 3, we present our obtained results then we conclude this paper in section 4.

2. Presentation of our algorithm

The aim of the proposed algorithm is to either encode or decode a message. Instead to use a classical scheme of encoding / decoding methods (keys, prime numbers, etc.), our algorithm is rather based on a combination of functions. Because the cardinality of the set of functions is infinite, it would be impossible for a third part (e.g. a hacker) to decode the secret information transmitted by the sender (Bob) to the receiver (Alice).

2.1. Encoding methodology

Let us consider the following simple example:

Bob wants to transmit to Alice the character 'A'. The ASCII code of 'A' is 65. Of course, instead to transmit 65, our algorithm sends Y = F(65) where F if an encoding function such as the following one:

$$Y = F(\text{'A'}) = \text{sqrt}(13.*X) + \log((2*X+1)/\log(X)) / \log(5.) - (7+\text{pow}(X, 1./3.)) / (X+24)$$

$$= 3.108632\text{e+}001$$

In the above equation, X is the ASCII code of the character one wants to encode (in this case, 65).

Thus, instead to send to Alice the integer value `65`, Bob sends the REAL value `3.108632e+001`

It is obvious that the encoding function could be AS COMPLICATE AS WE WANT since the set of functions is of INFINITE cardinality (the above equation shows a quite simple function).

Using the above equation, the encoding of the message found in the file "TEXT.txt" results in the encrypted message included in the file "CODED_TEXT.txt", the third one is "DECODED_TEXT.txt" which is the result of the decoding operation performed by Alice. Obviously, the file "DECODED_TEXT.txt" has the SIMILAR content than the original file, "TEXT.txt".

The robustness of our encoding methodology is that there exists an infinite number of functions:

$X-1$, $\log_{51}(4+X^{7/4})$, $2*\text{sqrt}(X)$, $(97+X*3)$, etc.

Among this infinite number of functions, one can deduce an infinite number of function combinations such as the following simple one:

$$Y = (29.*X*\log(X^{7/4}) - (2.*X^{2/3}+1))/(X*\log_9 X)*(2.*X^{12}+1.)*\log_3(5.))$$

where X is the ASCII code of the character to encode

A well chosen function combination used to encode some character will yield a hard task for a third part (e.g. a hacker) to deduce X from Y. Furthermore, our encoding algorithm uses a DIFFERENT encoding function for some characters like the space one. Thus, even if a hacker can observe that, for example, `1.917480e+001` corresponds to the ASCII code of the space character (which is 32), he would not be able to retrieve the other original characters because they are encoded with a DIFFERENT function.

Please note that our present version uses 2 different encoding functions: one for the space character and a second one for the remaining characters. However, this version could be easily modified so that a specific encoding function is used for EACH ASCII code (in other words, there are as many encoding functions as there are ASCII codes).

It is then clear that our encoding methodology yields a huge size of the solution space. In other words, the exploration of such a solution space makes the hacker facing an intractable problem for which he wont be able to solve even using a quantum algorithm.

2.2. Decoding methodology

Let us again consider the following encoding function:

Y = F(X) = sqrt(13.* X)+ log((2*X+1)/log(X)) / log(5.) - (7+pow(X, 1./3.)) / (X+24)

In order to retrieve the ASCII code X, one has to solve $X = F^{-1}(Y)$. This is not easy if function F is a combination of many difficult functions. However, this is not an issue. Indeed, let us assume that we are encoding the character 'A'. Then, the value of Y would be 3.108632e+001. Thus, we have: **F(X) - 3.108632e+001 = 0.**

To obtain X, we will simply use some numerical method achieving that goal. The Newton-Raphson method is well indicated for that purpose. Rapidly speaking, this method operates as follows (Fig. 4):

- One starts from some X_0. In our case, the solution is an ASCII code ($0 \leq$ ASCII code ≤ 127). So, one can choose $X_0 = 64$
- From X_0, one determines X_1 which is the intersection of the X-axis with the tangent of the curve of G at the point $(X_0, G(X_0))$
- One recursively determines X_2, X_3, etc.

Note that the tangent of G at the point $(X_n, G(X_n))$ is $Y = G'(X_n) * (X-X_n) + G(X_n)$

This tangent intersects the X-axis when Y = 0, namely when:

$$X = X_n - G(X_n) / G'(X_n)$$, i.e.:

$$X_{n+1} = X_n - G(X_n) / G'(X_n)$$

In order to be able to use this nice method, one should be able to calculate the derivate function of the encoding function for each abscissa $X_0, X_1, \ldots, X_{n+1}$

The iteration process stops when the condition $|G(X_i) / G'(X_i)| < 10^{-P}$ (e.g. P=5).

Note that in our context, G(X) = F(X) – C ; C is the encrypted value of an ASCII code (e.g. 3.108632e+001).

3. Results

Our method that was presented in the previous section has been implemented on both Linux and Windows7 operating systems (OS), using the C language. The results that are depicted in Table 1 are obtained under Linux 11.0 OS on a machine that has the following features:

- Dual core processors (AMD C-50) running at 1 GHz
- 4-Gb RAM

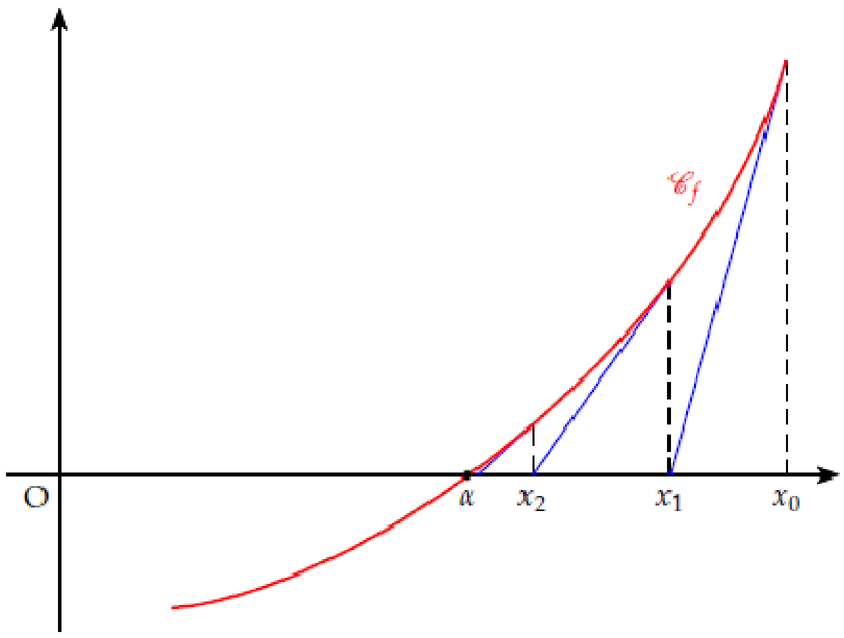


Fig. 4. The Newton-Raphson method to numerically solve G(X) = 0

Table 1 – Results obtained under Linux 11.0 OS using the C language
(with 2 core processors AMD C-50 @1GHz, 4-Gb RAM)

# Instance	ENCODING		DECODING	
	Size (bytes)	CPU Time (s) *	Size (bytes)	CPU Time (s) *
1	2331	0	30303	0
2	4662	0	60606	0
3	6993	0	90909	0
4	11655	0	151515	0
5	18648	0	242424	1
6	30303	0	393939	2
7	74592	0	969696	5
8	149184	1	1939392	10
9	298368	2	3878784	21
10	596736	5	7757568	43

*CPU Time = 0 s means CPU Time < 1 s (it would actually be some ns)

Although this machine presents weak features, the CPU times are very interesting even for the last sample (596736 bytes is the size of the text to encode, 7 757 568 bytes is the size of the encoded text obtained for the same instance). One can observe the following:

- For the same instance, the CPU time for the decoding is greater than that of the encoding because the size of the encoded text is greater than that of the original text and because the Newton-Raphson method used in the decoding needs, for each real value, some iterations to retrieve the original character

- The encoding is very fast (5s for a quite large instance – we recall that the machine presents weak features-)
- Considering the sizes and the obtained CPU times, our algorithm scales perfectly (the CPU time nearly grows linearly with the size –Fig.5-)

The CPU times are determined thanks to the last instruction of our C-code, the instruction system("ps –elf | grep Mahdoum_crypto") –please see Fig.6 in APPENDIX A- (Mahdoum_crypto being the name of the executable code).

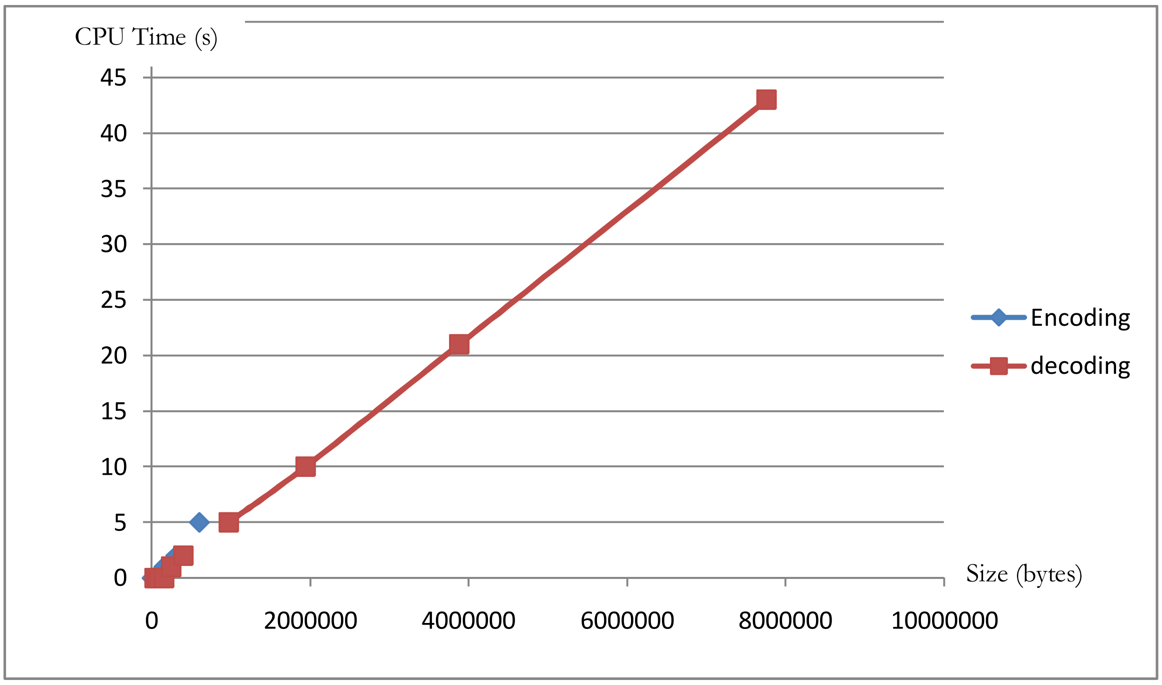


Fig.5 CPU time VS the size (bytes) of the file for both encoding and decoding techniques

The command is **Mahdoum_crypto if of** where *if* and *of* are respectively the names of the input file (the text to encode) and the output one (the encoded text). Fig.7 (in APPENDIX A) depicts a sample of a text to encode (included in file TEXT.txt), Fig.8 (in APPENDIX A) depicts the result of the encoding, included in file CODED_TEXT.txt

(obviously, the command is:
Mahdoum_crypto TEXT.txt CODED_TEXT.txt).

When the user starts running our program, he is asked to enter:

- either 'e' or 'E' to perform the encoding part or
- either 'd' or 'D' to perform the decoding part

In case none of these characters is entered, our program delivers an error message and exits.

An error message is also delivered in case:

- no file (e.g. TEXT.txt) is specified as a parameter for the encoding part
- the 2 file names (e.g. CODED_TEXT.txt and DECODED_TEXT.txt) are not specified. Note that CODED_TEXT.txt is an input file that contains the encoded message while DECODED_TEXT.txt is an output file that will contain the result of the decoding part.

4. Conclusion

In this paper, we have shown that some cryptographic components that were FIPS 140-2 level 3 validated get hacked. Furthermore, with the advent of quantum computing, it is critical to begin planning for the replacement of hardware, software, and services that use public-key algorithms now so that information is protected from future attacks. For this purpose, we have developed a new cryptographic algorithm that is not based on the classical scheme (keys, prime numbers, etc.). Instead, our algorithm is rather based on functions combination because the cardinality of the functions set is INFINITE, which makes the data corruption an intractable problem for which the hacker will not be able to solve even using a quantum algorithm. The present version of our algorithm uses only 2 encoding functions (one for the space character, the 2nd one for the remaining characters). In order to make our encoding technique stronger, we plan to use a SPECIFIC encoding function for EACH ASCII code. We formally believe that such an encoding will prevent any correlation, interpolation, etc. from the encoded data.

References

[1] https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms

[2] https://www.nist.gov/cryptography

[3] https://youtu.be/w8iQsgkiQ9I

[4] https://youtu.be/w_cjOjdN1bI

APPENDIX A

```
/*******************************************************************************************/
/*                         1rst version of text encoding / decoding,    May 2023                         */
/*                        Author: Dr. Ali Mahdoum       ali.mahdoum@gmail.com                            */
/*******************************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define faire
#define fait
#define then
#define endif

float Fct(float), Fct_Spec(float);
float Fct2(float, float), Fct_deriv(float), Fct2_Spec(float, float), Fct_deriv_Spec(float);
void coding(FILE *, FILE *);

int main(int argc, char *argv[])
{
  FILE *fp, *fp1;
  char c, C1;
  float Y, Y1, Y2, X, Z;
  int P= 5;
  int X1;
  printf("\n Please enter either e or E for encoding, either d or D for decoding\n");
  scanf("%c", &c);
  if(c != 'e' && c != 'E' && c != 'd' && c != 'D')
  then {printf("\n Command error\n");
          exit(1);
         }
  endif
  if(c == 'e' || c == 'E')
  then {if(!(fp=fopen(argv[1], "r")))
          then {printf("\nERROR: Cannot open file %s\n", argv[1]);
                   exit(1);
                  }
          endif
          if(!(fp1=fopen(argv[2], "w")))
          then {printf("\nERROR: Cannot open file %s\n", argv[2]);
                   exit(1);
                  }
         endif
         coding(fp, fp1);
        }
    else {Z=pow(10., -P);
           if(!(fp=fopen(argv[1],"r")))
           then {printf("\n ERROR: Could not open file %s\n", argv[1]);
                   exit(1);
                  }
           endif
           if(!(fp1=fopen(argv[2],"w")))
           then {printf("\n ERROR: Could not open file %s\n", argv[2]);
                     exit(1);
                    }
            endif
            while((c=getc(fp)) != EOF)
            faire {ungetc(c,fp);
                     fscanf(fp,"%f ",&Y);
                     Y1=4.0e+0;
                     Y2=2.0e+0;
                     X=1.e+2;
                    while(fabs(Y1 / Y2) >= Z)
                    faire {X = X - (Y1 / Y2);
                           if(fabs(Y - 19.17480) <= 1.e-6)
                           then {Y1=Fct2_Spec(X,Y);
                                  Y2 = Fct_deriv_Spec(X);
                                 }
                           else {Y1 = Fct2(X,Y);
                                 Y2 = Fct_deriv(X);
                                }
                           endif
                          }
                    fait
                    X1= (int) X;
                    if(X-X1 < (X1+1-X))
                    then C1= (char) X1;
                    else C1= (char) (X1+1.);
                    endif
                    fprintf(fp1,"%c", C1);
                   }
             fait
             fclose(fp);
             fclose(fp1);
            }
      endif
      system("/bin/ps -elf | grep Mahdoum_crypto");
     }


/*****************************************************************************************/
/*                                        Function FCT()                                 */
/*****************************************************************************************/
float Fct(float X)
{float Y;

  Y = sqrt(13.* X)+ log((2*X+1)/log(X)) / log(5.) - (7+pow(X, 1./3.)) / (X+24);
  return Y;
 }


/*****************************************************************************************/
/*                                      Function FCT_Spec()                              */
/*****************************************************************************************/
float Fct_Spec(float X)
{float Y;

  Y = 16+pow(X, 1./3.);
  return Y;
}


/*****************************************************************************************/
/*                                      Function CODING()                                */
/*****************************************************************************************/
void coding(FILE *fp, FILE *fp1)
{
  char c;
  int X1;
  float X, Y1;

  while((c=getc(fp)) != EOF)
  faire {ungetc(c,fp);
         fscanf(fp,"%c", &c);
```

```
          X1= (int) c;
          X=(float) X1;
          if(X1 == 32)
          then Y1=Fct_Spec(X);
          else Y1 = Fct(X);
          endif
          fprintf(fp1,"%le ",Y1);
         }
  fait
  fclose(fp);
  fclose(fp1);
 }


/*******************************************************************************************/
/*                                    Function FCT_DERIV()                                  */
/*******************************************************************************************/
float Fct_deriv(float X)
{float Y;

 Y = 13./(2.*sqrt(X))+ (2.*X*log(X)-(2.*X+1))/(X*log(X)*(2.*X+1.)*log(5.)) - ((((1./3.)*pow(X,-
     2./3.)*(X+24.))-(7.+pow(X,1./3.)))/pow(X+24.,2.));
 return Y;
 }


/*******************************************************************************************/
/*                                  Function FCT_DERIV_Spec()                               */
/*******************************************************************************************/
float Fct_deriv_Spec(float X)
{float Y;

 Y = 1./3. * pow(X, -2./3.);
 return Y;
}



/*******************************************************************************************/
/*                                      Function FCT2()                                     */
/*******************************************************************************************/
float Fct2(float X, float A)
{float Y;

 Y = sqrt(13.* X)+ log((2*X+1)/log(X)) / log(5.) - (7+pow(X, 1./3.)) / (X+24) - A;
 return Y;
}


/*******************************************************************************************/
/*                                    Function FCT2_Spec()                                  */
/*******************************************************************************************/
float Fct2_Spec(float X, float A)
{float Y;

 Y = 16+pow(X, 1./3.)- A;
 return Y;
 }
```

Fig.6 The C-code implementing our cryptographic algorithm

```
ABCD Bob is sending a message to Alice.
AZERTY
QWERTY
1234567890-= ][poiuytrewqasdfghjkl;'\/.,mnbvcxz!@#$%^&*()_+}{POIUYTREWQASDFGHJKL:"|?>
<MNBVCXZ
```

Fig.7 The text to encode (e.g. in file TEXT.txt)

```
3.108632e+001 3.131737e+001 3.154661e+001 3.177409e+001 1.917480e+001 3.131737e+001 4.029624e+001
3.793445e+001 1.917480e+001 3.922493e+001 4.099398e+001 1.917480e+001 4.099398e+001 3.849313e+001
4.011979e+001 3.830787e+001 3.922493e+001 4.011979e+001 3.886086e+001 1.917480e+001 3.774627e+001
1.917480e+001 3.994252e+001 3.849313e+001 4.099398e+001 4.099398e+001 3.774627e+001 3.886086e+001
3.849313e+001 1.917480e+001 4.116646e+001 4.029624e+001 1.917480e+001 3.108632e+001 3.976441e+001
3.922493e+001 3.812164e+001 3.849313e+001 2.628497e+001 1.250596e+001 3.108632e+001 3.640005e+001
3.199986e+001 3.479361e+001 3.520253e+001 3.620338e+001 1.250596e+001 3.458722e+001 3.580661e+001
3.199986e+001 3.479361e+001 3.520253e+001 3.620338e+001 1.250596e+001 2.710363e+001 2.737072e+001
2.763505e+001 2.789672e+001 2.815579e+001 2.841234e+001 2.866644e+001 2.891817e+001 2.916757e+001
2.683370e+001 2.600596e+001 3.014323e+001 3.698338e+001 3.659559e+001 4.047187e+001 4.029624e+001
3.922493e+001 4.133818e+001 4.201762e+001 4.116646e+001 4.082073e+001 3.849313e+001 4.167936e+001
4.064670e+001 3.774627e+001 4.099398e+001 3.830787e+001 3.867746e+001 3.886086e+001 3.904335e+001
3.940563e+001 3.958545e+001 3.976441e+001 2.965968e+001 2.425985e+001 3.679002e+001 2.656085e+001
2.628497e+001 2.572372e+001 3.994252e+001 4.011979e+001 3.793445e+001 4.150914e+001 3.812164e+001
4.184885e+001 4.218567e+001 2.236799e+001 3.085343e+001 2.301706e+001 2.333438e+001 2.364717e+001
3.717566e+001 2.395560e+001 2.514908e+001 2.456008e+001 2.485644e+001 3.736689e+001 2.543813e+001
4.268562e+001 4.235301e+001 3.437952e+001 3.417046e+001 3.288641e+001 3.540511e+001 3.620338e+001
3.520253e+001 3.479361e+001 3.199986e+001 3.580661e+001 3.458722e+001 3.108632e+001 3.499871e+001
3.177409e+001 3.222393e+001 3.244636e+001 3.266718e+001 3.310411e+001 3.332029e+001 3.353498e+001
2.941472e+001 2.269500e+001 4.251966e+001 3.061864e+001 3.038193e+001 2.990250e+001 3.374823e+001
3.396004e+001 3.131737e+001 3.560646e+001 3.154661e+001 3.600558e+001 3.640005e+001 1.250596e+001
```

Fig.8 The encoded text (e.g. in file CODED_TEXT.txt)