PERFORMANCE OF OPTIMIZED POST-QUANTUM HASH BASED CRYPTOGRAPHY USING QPQ-CD ON 5G NETWORK AUTHENTICATION KEY AGREEMENT

RAKOTONDRAMANANANA Radiarisainana Sitraka[1],
RANDRIAMITANTSOA Paul Auguste[2],

[1] *PhD student, TASI, ED-STIII, Antananarivo, Madagascar*
[2] *Thesis director, TASI, ED-STII, Antananarivo, Madagascar*

ABSTRACT

The QPQ-CD (Quantum and PostQuantum CipherKey Dynamic uses a dynamic key after the registration of the mobile at the 5G Network. The algorithm could be divided in two parts : The Quantum Cryptography (QC) and Post Quantum Cryptography (PQC) . The Post Quantum Cryptography will use 12 families of algorithms which are duplicate algorithm of : MD45, MD45xor, MD54, MD54xor, SHA256, SHA256xor for treating the 2 output QHT separately and QAT of the Quantum algorithm. Before the selector of the next key, the 12 keys of 256 bits are optimized of one bit change and selected the best effective probability. It will be based on probability by the extremity, probability of proximity, probability of a bit changed, and probability of disorder or binary entropy and probability of penalties. According to the order of its probabilities, the following key after QPQ-CD thus has the behavior: very far from the ends of the key (00 ... 000 and 11 ... 111); very far from the previous key; several bits changed and very messy from the point of view bit zero and bit one and the penalty of bad extremity proximity. All key chosen keys have a probability by the extremity and proximity between 50% until 100%. The probability of the bit changed is to 50%, the entropy is near 100%. The result is only obtained if the system uses optimization of one bit changed. Any chart representative also permits to include all 12 keys obtained after the Post Quantum algorithm have all chance to be selected by the system. So all varieties of PostQuantum Cryptography is necessary to be implemented on the QPQ-CD. All the result is simulated with Matlab.

Keyword : *QPQ-CD, SHA256, MD5, MD4, PQC*

---

1. Introduction

In the network 5G, the UE and operator use mutual authentication based on the master-key K. A static key K is so vulnerable to the user. The algorithm QPQ-CD uses a dynamic key with optimized selectors with high probability of extremity, probability of proximity, probability of a bit changed and probability in case of entropy.

2.1 Implementation of simplified QPQ-CD

QPQ_CD uses the master key Dynamicity K. This function has as input the previous key K and an activation signal A and a parameter r defining the complexity rule of QPQ-CD. [1] [2][3][4]
The steps of the algorithm are:

- The initialization phase: the goal is to initialize K, r, and i an activation counter and to generate from the Expansion towards the Matrix (E.M) a matrix of $16r \times 16r$ of 8 bits.

- The insertion phase: It consists of periodically inserting while scanning the line of the matrix 16r × 16r a key obtain through the Expansion towards the Linearity (E.L). The insertion is executed only at each activation signal. Since the QPQ-CD algorithm uses 3 16r × 16r matrices, a key-generating function denoted by G makes it possible to generate 3 parts of the key of initializations for each matrix.
- The phase of quantum cryptography : The phase of quantum cryptography uses the method of confusion either by the Hilbert method or by Arnold's method.
- The PQ cryptography phase: it uses several hash algorithm samples to summarize the matrix after confusion in order to have a 256 bits key.
- Phase selectors: it selects the next key K+ appropriate.

The output of the QPQP-CD algorithm is another key generated K+, for other applications especially in the authentication. QPQ-CD is also a family of KDF algorithm. The simplified schema of the QPQ-CD algorithm is represented in Figure 1.

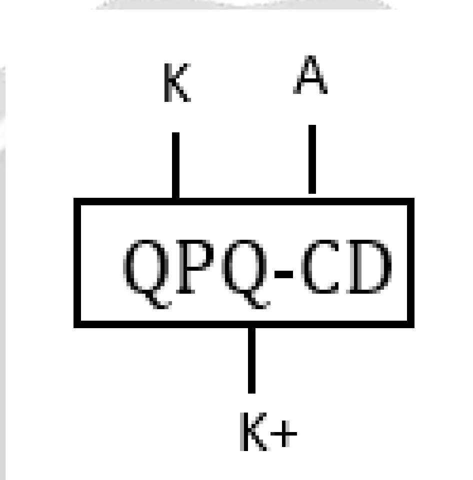

Fig -1 QPQ-CD

2.2 Evaluation of QPQ-CD

For the performance study of the QPQ-CD algorithm, the activation counter will be traversed until the end of the insertion line. Thus, i vary from 1 ... 16r.[5][6][7][8]

The QPQ-CD algorithm will be characterized by the initialization phase that generates r, i, K then JR, JG and JB. The insertion of the keys generated by the Expansion towards the Linearity (E.L) and the key generator G will be repeated at each blur up to 16r. The QC algorithm followed by PQC will be finalized by the optimization selector to obtain the key next K +.

The selection algorithm uses several criteria to identify the best key using the probability of not detecting the key from the previous key by focusing on how opponents think and other relevant criteria. Since the insertion of the matrix is done at each line from 1 to 16r, the authentication sample will be limited to this value 16r.

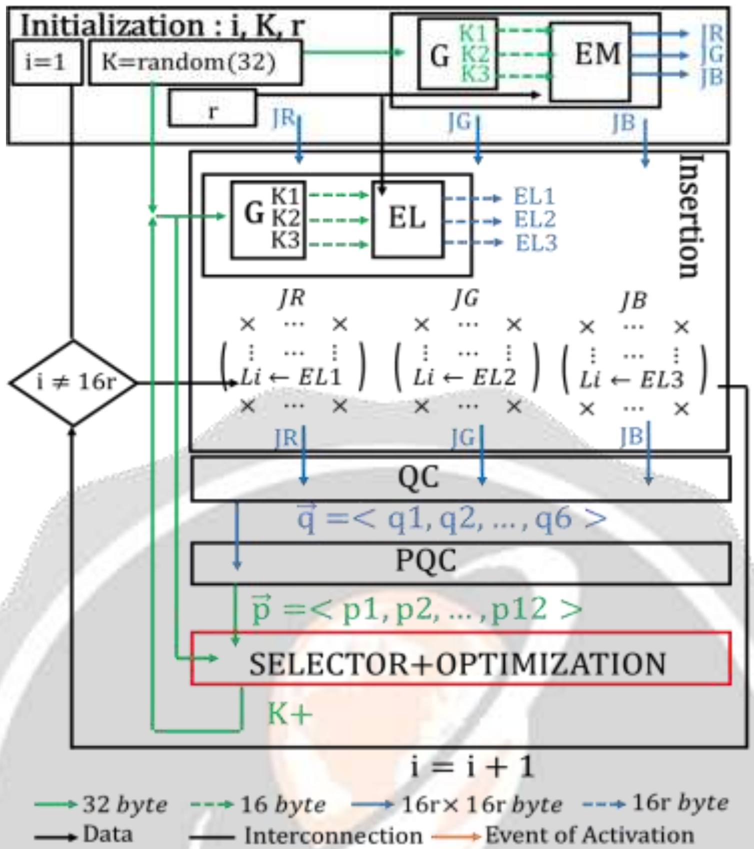

Fig -2 Evaluation of QPQ-CD

2.3 PQ Cryptography based on Hashing

The PQ Cryptography is the family of cryptosystem which is Quantum computer safe. The standard notation for hash algorithms defined by the Table 1 which are left shift, the modulo addition. 2^{32}; Boolean operations XOR, AND, OR, NOT [9][10]

Table -1 Standard Notation algorithms for the hash based cryptography

Symbol	Notation
$[<<<]_s$	rotation of s bit
[+]	addition module 2^{32}
$\oplus$	XOR
$\wedge$	AND
$\vee$	OR
$\neg$	NOT

2.4 Hash based cryptography

The hash function uses a message with arbitrary length at the input and give a fix length result named hash or condensat or fingerprints. Generally, the most family of hash based cryptography are based on the Merkle-Damgärd schema.

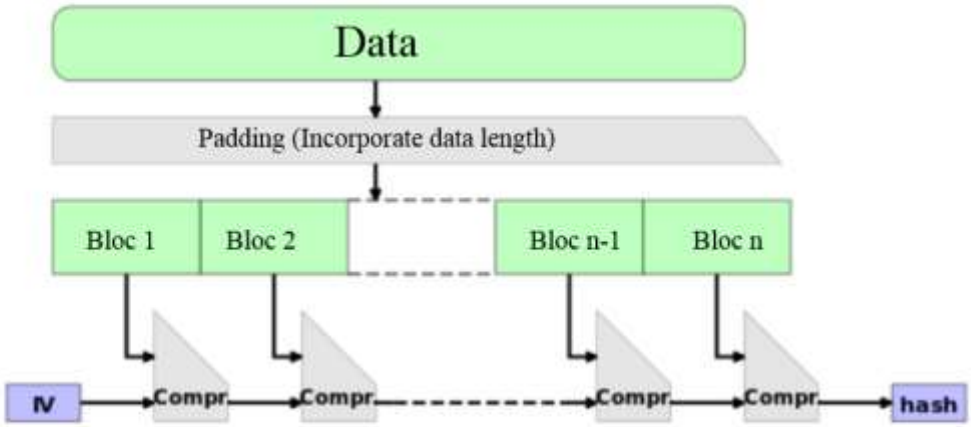

Fig -3 Schema of Merkle-Damgärd

The schema of Merkle-Damgärd uses the variable length data which are split to a fix bloc by using padding algorithms. In their padding is incorporating the data length. The hash of the precedent bloc will be used like initial vector of the next block. The hash bloc use multiple irreversible compression and should have the first initial vector IV.

2.5 MD 5

The MD5 (Message-Digest) algorithm is a widely used hash function producing a 128-bit hash value.

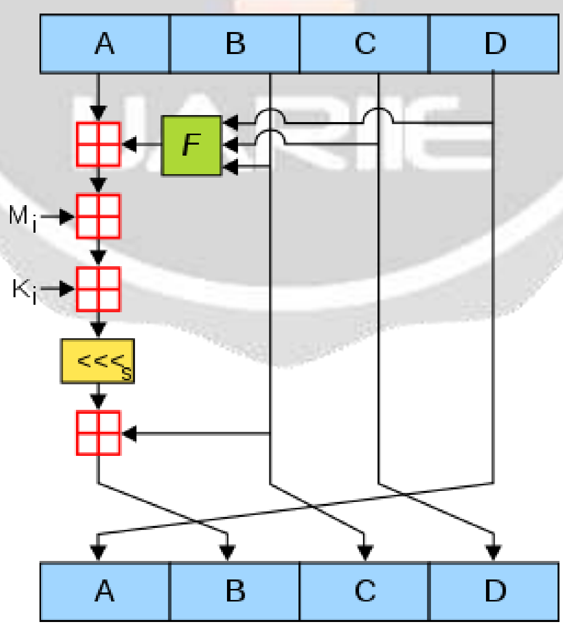

Fig 3- One MD5 Operation schema bloc

One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. *F* is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation. $<<<_s$ denotes a left bit rotation by *s* places; *s* varies for each operation. $\boxplus$ denotes addition modulo 2^{32}.

- Message

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words) : the message is padded so that its length is divisible by 512. The padding works as follows:

- first a single bit, 1, is appended to the end of the message.
- This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512.
- The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *A*, *B*, *C*, and *D*. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a non-linear function *F*, modular addition, and left rotation. Figure 3 illustrates one operation within a round. There are four possible functions; a different one is used in each round:

$$\begin{aligned} F(B,C,D) &= (B \wedge C) \vee (\neg B \wedge D) \\ G(B,C,D) &= (B \wedge D) \vee (C \wedge \neg D) \\ H(B,C,D) &= B \oplus C \oplus D \\ I(B,C,D) &= C \oplus (B \wedge \neg D) \end{aligned} \tag{1}$$

- Constant

MD5 uses 64 constant values of 32-bit words. The constant K is defined by:

$$K = \lfloor |\sin(i+1) \,.\, 2^{32}| \rfloor$$

- Pseudocode

```
//Note: All variables are unsigned 32 bit and wrap modulo 2^32 when calculating
var int[64] s, K
var int i

//s specifies the per-round shift amounts
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22 }
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20 }
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23 }
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21 }

//Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63
    K[i] := floor(2^32 × abs(sin(i + 1)))
end for
```

```
//(Or just use the following precomputed table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }
//Initialize variables:
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D
//Pre-processing: adding a single 1 bit
append "1" bit to message
// Notice: the input bytes are considered as bits strings,
//  where the first bit is the most significant bit of the byte.

//Pre-processing: padding with zeros
append "0" bit until message length in bits ≡ 448 (mod 512)
append original length in bits mod 2^64 to message//Process the message in successive 512-bit
chunks:

for each 512-bit chunk of padded message
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
    //Initialize hash value for this chunk:
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := do //Main loop:
    for i from 0 to 63
        var int F, g
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and D)
            g := i
        else if 16 ≤ i ≤ 31 then
            F := (D and B) or ((not D) and C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47 then
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63 then
            F := C xor (B or (not D))
            g := (7×i) mod 16
        F := F + A + K[i] + M[g]  //Be wary of the below definitions of a,b,c,d
        A := D
        D := C
        C := B
        B := B + leftrotate(F, s[i])
    end for
    //Add this chunk's hash to result so far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for
var char digest[16] := a0 append b0 append c0 append d0 //(Output is in little-endian)

leftrotate (x, c) //leftrotate function definition
return (x << c) binary or (x >> (32-c));
```

2.6 MD4

An operation of MD4. MD4 comprises 48 blocks of Figure 3 , grouped in three rounds of 16 operations. F is a nonlinear function, which varies according to the turn. Mi symbolizes a 32-bit block from the message to be chopped and Ki is a 32-bit constant, different for each operation. The 3 function laps used are:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \quad (2)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$

```
Process each 16-word block. */
      for i = 0 to N/16-1 do
        /* Copy block i into X. */
        for j = 0 to 15 do
          Set X[j] to M[i*16+j].
        end for /* of loop on j */
        /* Save A as AA, B as BB, C as CC, and D as DD. */
        AA = A
        BB = B
        CC = C
        DD = D
        /* Round 1. */
        /* Let [abcd k s] denote the operation
             a = (a + F(b,c,d) + X[k]) <<< s. */
        /* Do the following 16 operations. */
        [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
        [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
        [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
        [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]
        /* Round 2. */
        /* Let [abcd k s] denote the operation
             a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
        /* Do the following 16 operations. */
        [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
        [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
        [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
        [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]
        /* Round 3. */
        /* Let [abcd k s] denote the operation
             a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
        /* Do the following 16 operations. */
        [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
        [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
        [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
        [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]
        /* Then perform the following additions. (That is, increment each
           of the four registers by the value it had before this block
           was started.) */
        A = A + AA
        B = B + BB
        C = C + CC
        D = D + DD
    end for/* of loop on i */
```

2.7. SHA256

SHA (Secure Hashed) 256 is a set of cryptographic hash functions designed by the United States National Security Agency (NSA). One iteration of a SHA-2 family compression function. The blue components perform the following operations :

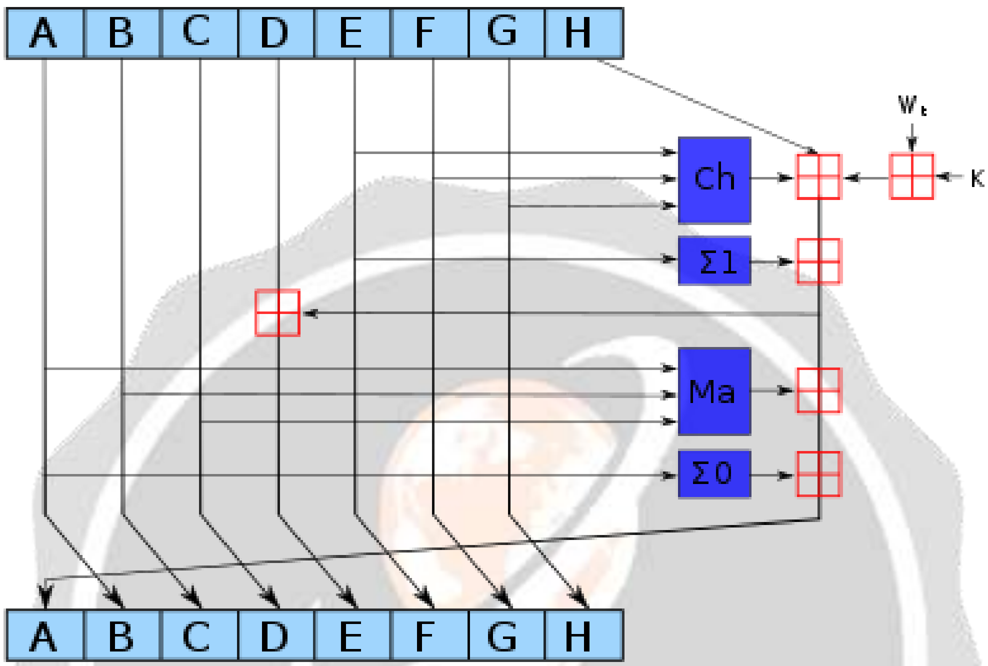

Fig -5 One SHA-256 operation schema bloc

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256. The red is addition ⊞ modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

$$
\begin{aligned}
Ch(E,F,G) &= (E \wedge F) \oplus (\neg E \wedge G) \\
Ma(A,B,C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\
\sum\nolimits_0 (A) &= (A \gg 2) \oplus (A \gg 13)(A \gg 22) \\
\sum\nolimits_1 (E) &= (E \gg 6) \oplus (E \gg 11)(E \gg 25)
\end{aligned} \tag{3}
$$

The named of parameters with SHA256 is similar to the MD5 like word W and constant K.

```
Note 1: All variables are 32 bit unsigned integers and addition is calculated
modulo 2^32
Note 2: For each round, there is one round constant k[i] and one entry in the
message schedule array w[i], 0 ≤ i ≤ 63
Note 3: The compression function uses 8 working variables, a through h
Note 4: Big-endian convention is used when expressing the constants in this
pseudocode,

    and when parsing message block data from bytes to words, for example,
    the first word of the input message "abc" after padding is 0x61626380
Initialize hash values:

(first 32 bits of the fractional parts of the square roots of the first 8
primes 2..19):
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19


Initialize array of round constants:
(first 32 bits of the fractional parts of the cube roots of the first 64
primes 2..311):
k[0..63] :=
   0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe,
0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa,
0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb,
0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624,
0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb,
0xbef9a3f7, 0xc67178f2


Pre-processing (Padding):
begin with the original message of length L bits
append a single '1' bit
append K '0' bits, where K is the minimum number >= 0 such that L + 1 + K +
64 is a multiple of 512
append L as a 64-bit big-endian integer, making the total post-processed
length a multiple of 512 bits


Process the message in successive 512-bit chunks:
break message into 512-bit chunks
```

```
for each chunk
    create a 64-entry message schedule array w[0..63] of 32-bit words
    (The initial values in w[0..63] don't matter, so many implementations
zero them here)
    copy chunk into first 16 words w[0..15] of the message schedule array

    Extend the first 16 words into the remaining 48 words w[16..63] of the
message schedule array:
    for i from 16 to 63
        s0 := (w[i-15] rightrotate  7) xor (w[i-15] rightrotate 18) xor (w[i-
15] rightshift  3)
        s1 := (w[i- 2] rightrotate 17) xor (w[i- 2] rightrotate 19) xor (w[i-
2] rightshift 10)
        w[i] := w[i-16] + s0 + w[i-7] + s1
    Initialize working variables to current hash value:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7
    Compression function main loop:
    for i from 0 to 63
        S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
        ch := (e and f) xor ((not e) and g)
        temp1 := h + S1 + ch + k[i] + w[i]
        S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
        maj := (a and b) xor (a and c) xor (b and c)
        temp2 := S0 + maj

        h := g
        g := f
        f := e
        e := d + temp1
        d := c
        c := b
        b := a
        a := temp1 + temp2

    Add the compressed chunk to the current hash value:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h

Produce the final hash value (big-endian):
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append
h6 append h7
```

2.8 PQ-C

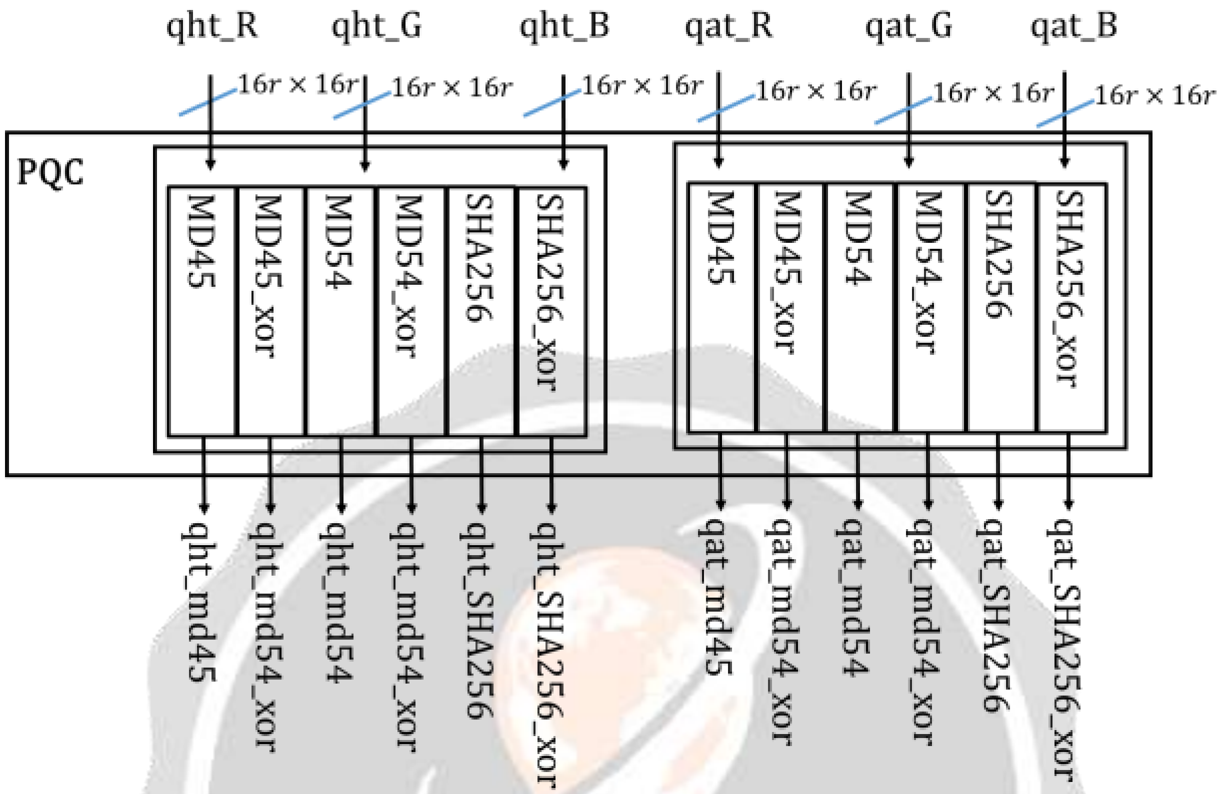

Fig -6 PQC algorithm

MD45: MD4 concatenated with MD5 on block matrices _R, _G, _B
MD54: M5 concatenated with MD4 on block matrices _R, _G, _B
SHA256: SHA 256 on block matrices _R, _G, _B
MD45: MD4 concatenated with MD5 on the separated matrices _R, _G, _B followed by xor between them
MD54: M5 concatenated with MD4 on the separated matrices _R, _G, _B followed by xor between them
SHA256: SHA 256 on the block matrices _R, _G, _B followed by xor between them

The PQC [5,6] used in this module use multiple hash function. The block matrix _R, _G, _B is a matrix of three dimensions of (16r × 16r × 3) byte by grouping the 3 parts qht_R, qht_G, qht_B resp. qat_R, qat_G, qat_B each size (16r × 16r). The 12 outputs are : qht_md45, qht_md45_xor, qht_md54, qht_md54_xor, qht_sha256, qht_sha256_xor, qat_md45, qat_md45_xor, qat_md54, qat_md54_xor, qat_sha256, qat_sha256_xor can also be simplified by a vector p formed by the elements p1 ... p12

3. Interpretation

The selector uses the effective probability to select the best option. The effective probability is derived from the probability of extremity, probability of proximity, probability of a bit changed, probability of disorder and probability of penalties. All the curves studied use interpolation by Hermite polynomials as known as PCHIP (Piecewise Cubic Hermite Interpolating Polynomial).

- Effective propability

The effective probability is obtained by the formula selector formula combined with an optimization based on one bit changed. The selector is defined by :

$$\begin{cases} prob_{eff4} = Max\left\{\dfrac{3*(prob_{extr\_prox}) + 2*prob_{change} + prob_H}{3+2+1} + Penalty(prob_{extr\_prox})\right\} \\ prob_{extr\_prox} = \dfrac{prob_{prox} + prob_{extr}}{2} \\ Penalty(prob_{extr\_prox}) = \begin{cases} 0 \ \ si\ min(x-ref, y-ref) \geq 0 \\ min(x-ref, y-ref) \qquad others \end{cases} \end{cases} \tag{4}$$

The algorithm needs optimization for having good performance. The schema bloc of the optimization is represented by the Figure 5. itself. The optimizer is used to increase the number of choices in the key by 256 times by changing only one bit in the key. Then, it possible to use a selector from option 4. The 12 optimized keys coming out of the PQC block will then be selected by the same selector of option 4. The Figure 24 shows the selector with optimization.

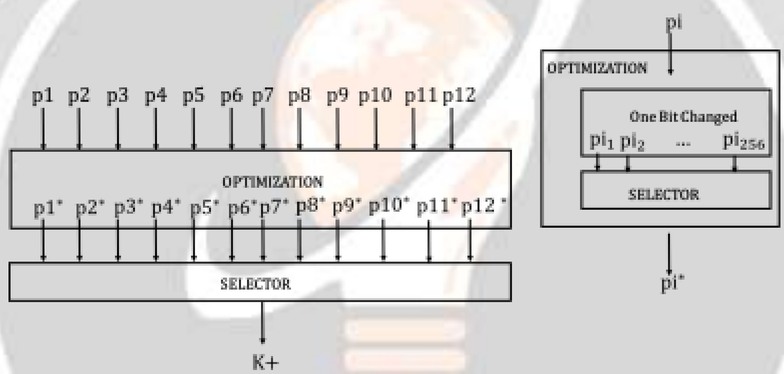

Fig -7 Optimization and Selector

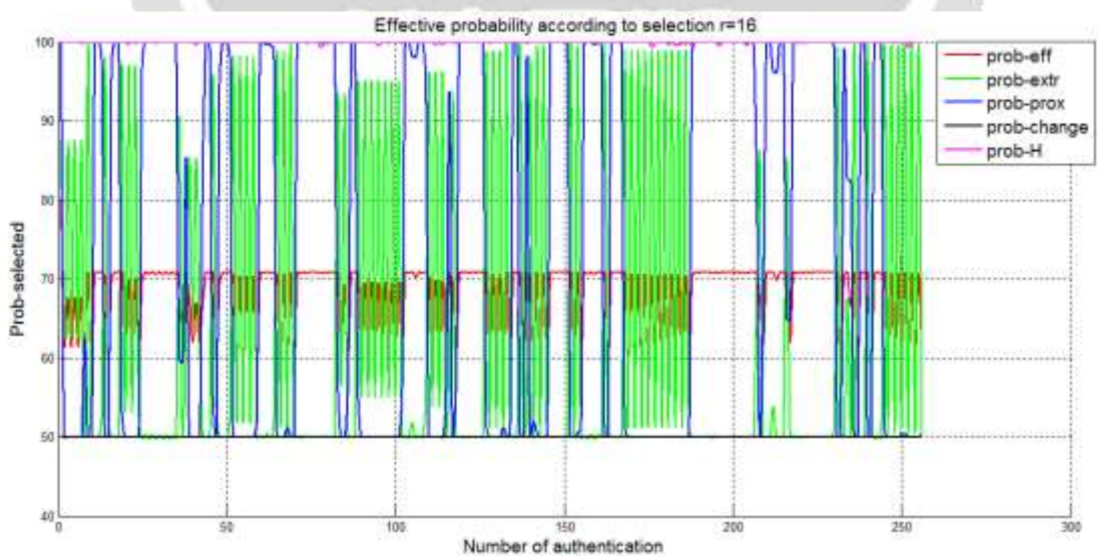

Fig 7a- Effective probability according to selection r = 16

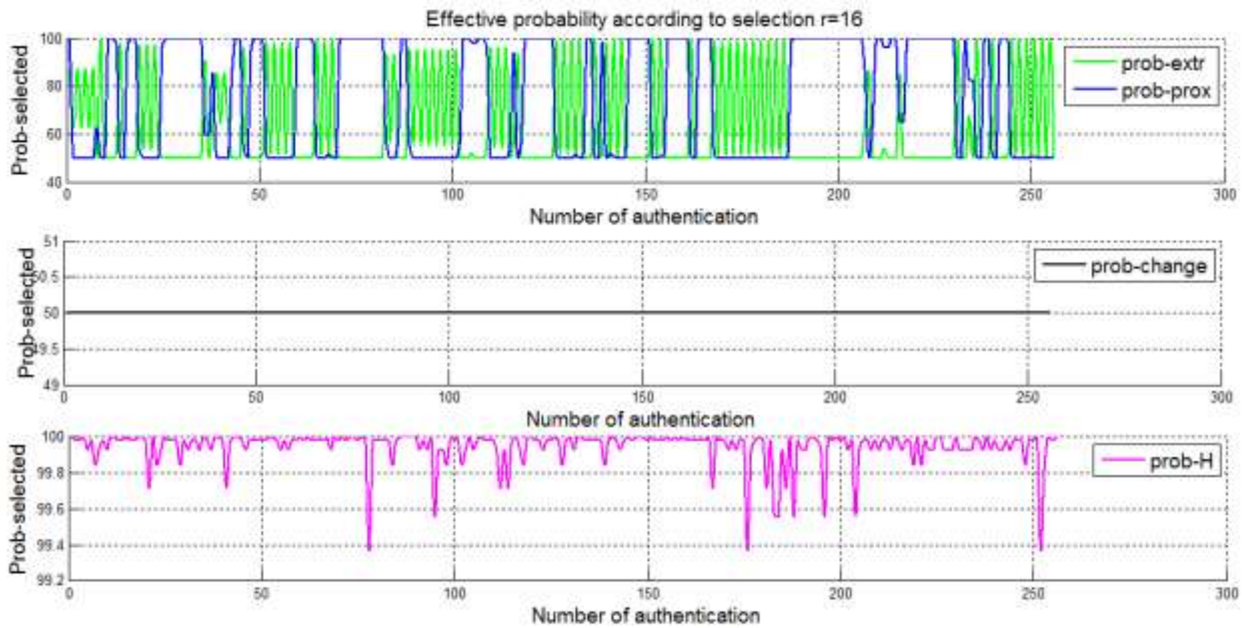

Fig -7b : effective probability according to selection r = 16

Interpretation :

By using r = 16, the effective probability optimized could achieve with probability by the extremity and probability of proximity more than 50%. This two parameter could be best at the same time. So the optimal value is near than 50%. The optimal probability of bit change is always at 50% and the probability of entropy in case of disorder in the number of bits zero and number of bits one is near 100%. The Figure 7b is obtained by separating some parameter of the Figure 7a. In this, the effective probability obtained by using all the probabilities cited are more than 80%. The new key generate will be very far the extremity of easy key to define, very far from the last key, a good probability of a bit changed and a good propriety of disorder.

- Probability by the extremity:

The brute force attack is to browse all the possibilities in a random way is not profitably compared to the orderly way. According to the logic as well, an opponent wanting to test all possible keys using the brute force algorithm always starts with 00...000 up to 11...111 using increases or starting with 11...111 up to 00 ...000 using decreases.

$$\underset{\text{Increases}}{\begin{cases} 00000 \dots\dots\dots 000 \\ \dots\dots\dots\dots\dots\dots\dots \\ 11111 \dots\dots\dots .111 \end{cases}} \quad \underset{\text{Decreases}}{\begin{cases} 11111 \dots\dots\dots .111 \\ \dots\dots\dots\dots\dots\dots\dots \\ 00000 \dots\dots\dots 000 \end{cases}} \qquad (5)$$

The closer the key is to 00 ... .000 or closer to 11 ... .111, the lower the probability of not detecting the key.

If the key is close to 0, the high-order one is difficult to detect. The probability of not detecting defined by:

$$p = \frac{\sum_{i=0}^{n-1} [k(i) == 1].\, 2^i}{\sum_{i=0}^{n-1}.\, 2^i} \qquad (6)$$

If the key is close to 1, the high-order zero value bit is difficult to detect. The probability is defined by:

$$q = \frac{\sum_{i=0}^{n-1} [k(i) == 0].\, 2^i}{\sum_{i=0}^{n-1}.\, 2^i} \qquad (7)$$

Using both approaches, the probability that the key is close to 00 ... 000 and 11 ... 111 is formed by the appearance of one of two formulas (2) and (3):

$$prob_{extr} = \begin{cases} p = \dfrac{\sum_{i=0}^{n-1}[k[i] == 1].2^i}{\sum_{i=0}^{n-1}.2^i} & if\ near(k, 0000 \ldots 000) = 1 \\ q = \dfrac{\sum_{i=0}^{n-1}[k[i] == 0].2^i}{\sum_{i=0}^{n-1}.2^i} & if\ near(k, 1111 \ldots 111) = 1 \\ \begin{cases} near(k, 0000 \ldots 000) = (k[n] == 0) \\ near(k, 1111 \ldots 111) = (k[n] == 1) \end{cases} \end{cases} \tag{8}$$

Where n is the size of the key.
prob_extr is the probability that key k will be close to the extremity 00 ... 000 or 11... 111
The near function is defined as follows the formula (4)

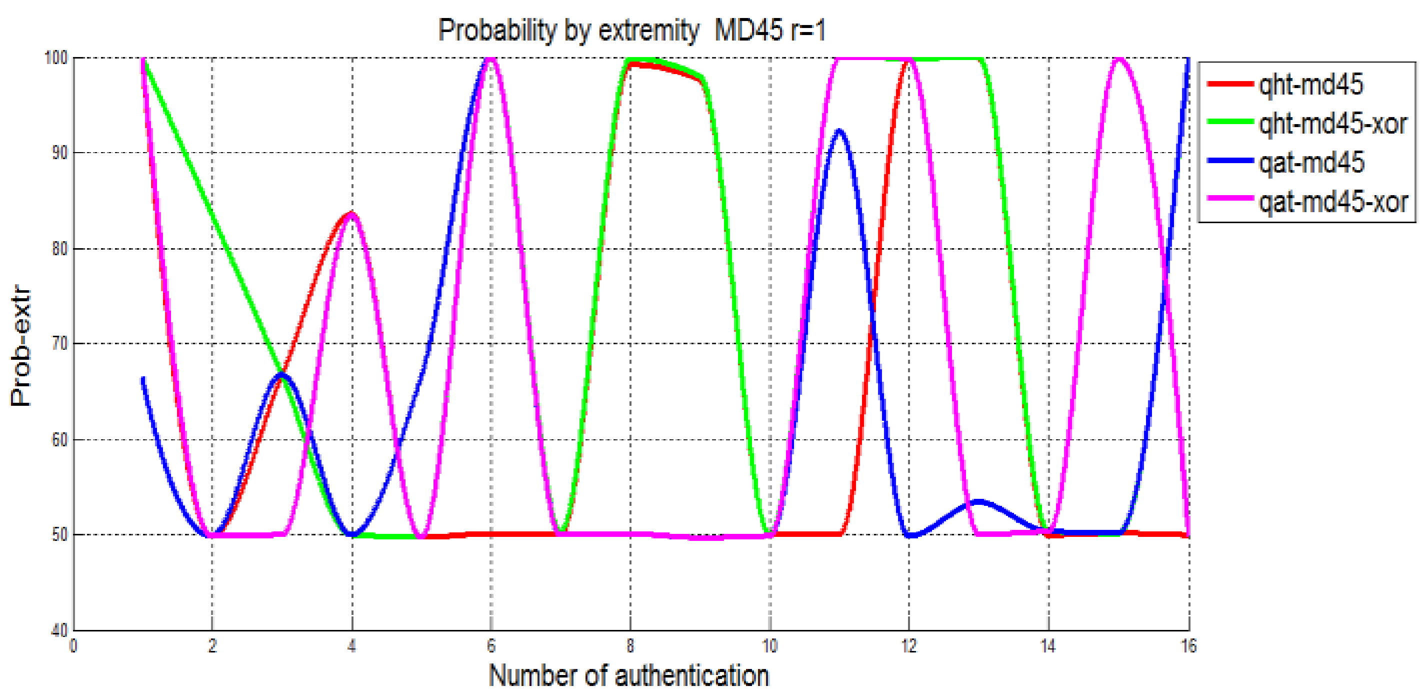

Fig -8 Extract of probability by the extremity using MD45

Interpretation :

At the optimal, the probability of signifies that it is too difficult to search for the key if the attacker use brute forcing to test to begin until the end of the possible key In the Figure, the probably vary between 50% to 100. So, all key is possible to be selected by the QPQ-CD. For the 12 options of Post Quantum Cryptography, the curve obtained is highly identic.

- Probability of proximity:

The probability of proximity is summed up by the fact that the two keys: current key and next keys are all closer to one another. By imagining two specific keys to compare:

$$(k_1, k_2) = (0010, 0100)$$

The distance between the two bits is the subtraction between the two keys:

$$xor(k_1, k_2) = 0110$$

To go from $k_1 \to k_2$ resp. $k_2 \to k_1$ is as going from $0000 \to xor(k_1, k_2)$ resp. $1111 \to xor(k_1, k_2)$

$$prob\_prox = prob\_extr(xor(k_1, k_2)) \quad (9)$$

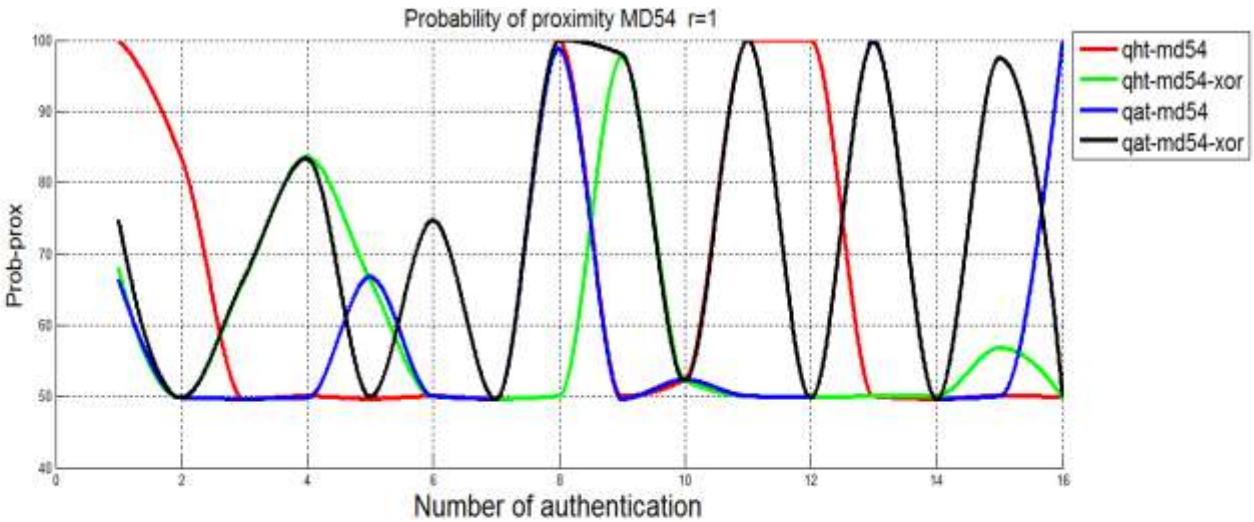

Fig -9 Extract of probability of proximity using MD45

After the optimization, the probability of proximity also varies to 50% until 100%. Note that for one example of the 12 selection, the choice will be the family of MD54 algorithm of Post-Quantum Cryptography. The selector will choose one option the probability of the extremity is near the 50% and the probability of proximity is near 100% or the other options : the inverse.

- bit probability changed :

Assuming two keys (k_1, k_2), the probability of bit change is not good if it's near to 256bits. So, it's defined by :

$$prob_{change} = \begin{cases} \frac{\sum_{i=0}^{n-1} xor(k_1, k_2)[i]}{n} & if\ \frac{\sum_{i=0}^{n-1} xor(k_1, k_2)[i]}{n} \leq 0.5 \\ \left|1 - \frac{\sum_{i=0}^{n-1} xor(k_1, k_2)[i]}{n}\right| & others \end{cases} \quad (10)$$

The xor operator can also check if two keys are not identical. But the key is not so good when the number of the bit change is near the 0 or near the 256 bits. The probability is maximum this probability is equal to 50%.

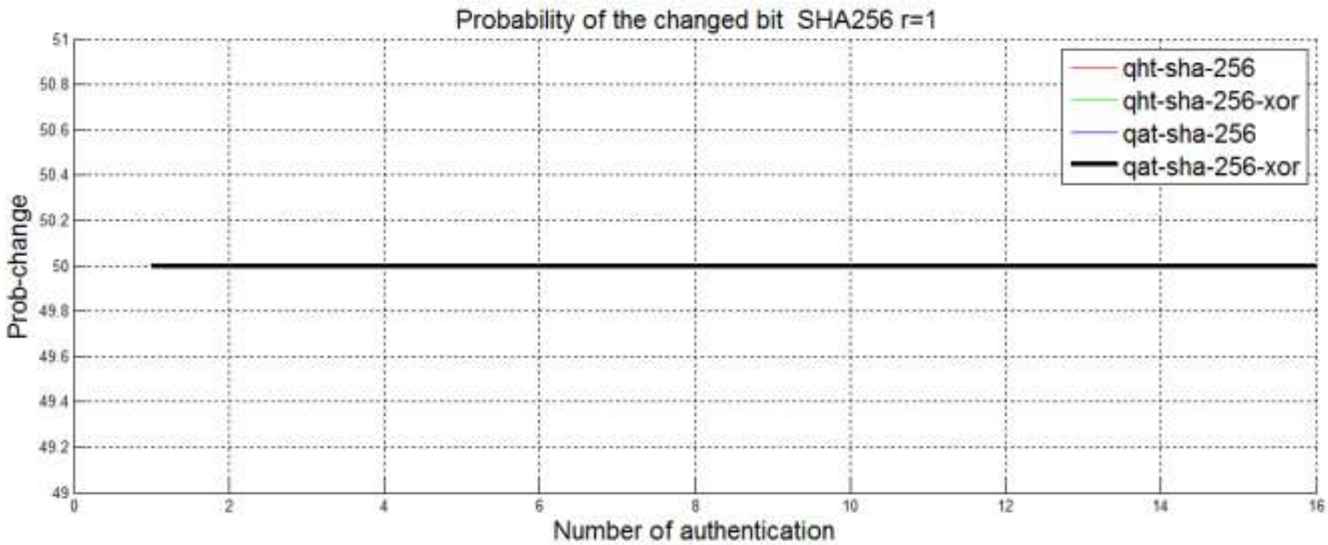

Fig- 10 probability of the changed bit using SHA 256

Interpretation :

The best parameter offered by the optimization is the probability of bit change. The number of bit change shouldn't near the 0 bit or 256 bits. If the number of bits changed is zero, that means that the key is static. If the number of bits changed is 256, that means the next key is only obtained by not operator of the previous key. If an attacker chooses an attack like to change one, two … n bit of keys or the logic inverse attack, concerning not to change only one,two … n bits. The key will be defined easily. That why for all options of the Post Quantum Cryptography, the probability of bit change is always to 50%.

- Binary Entropy: The entropy of the following key is defined by:

$$H = -p(0)log_2(p(0)) - p(1)log_2(p(1)) \tag{11}$$

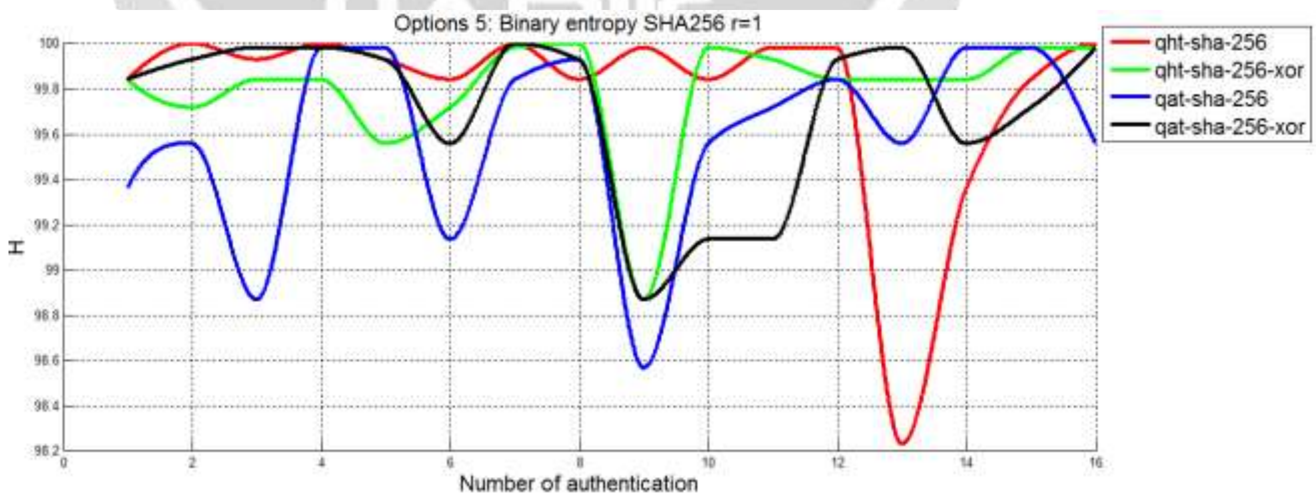

Fig -11 Extract of binary entropy for SHA256

Interpretation :

The entropy binary specify that the new key is totally on disorder or it's highly reparted. For having a best binary entropy signifies that the number of bits zero is the same of the number of bit ones. In the Post-Quantum cryptography used with QPQ-CD, the probability is nearly high the 100%.Chart of Selection

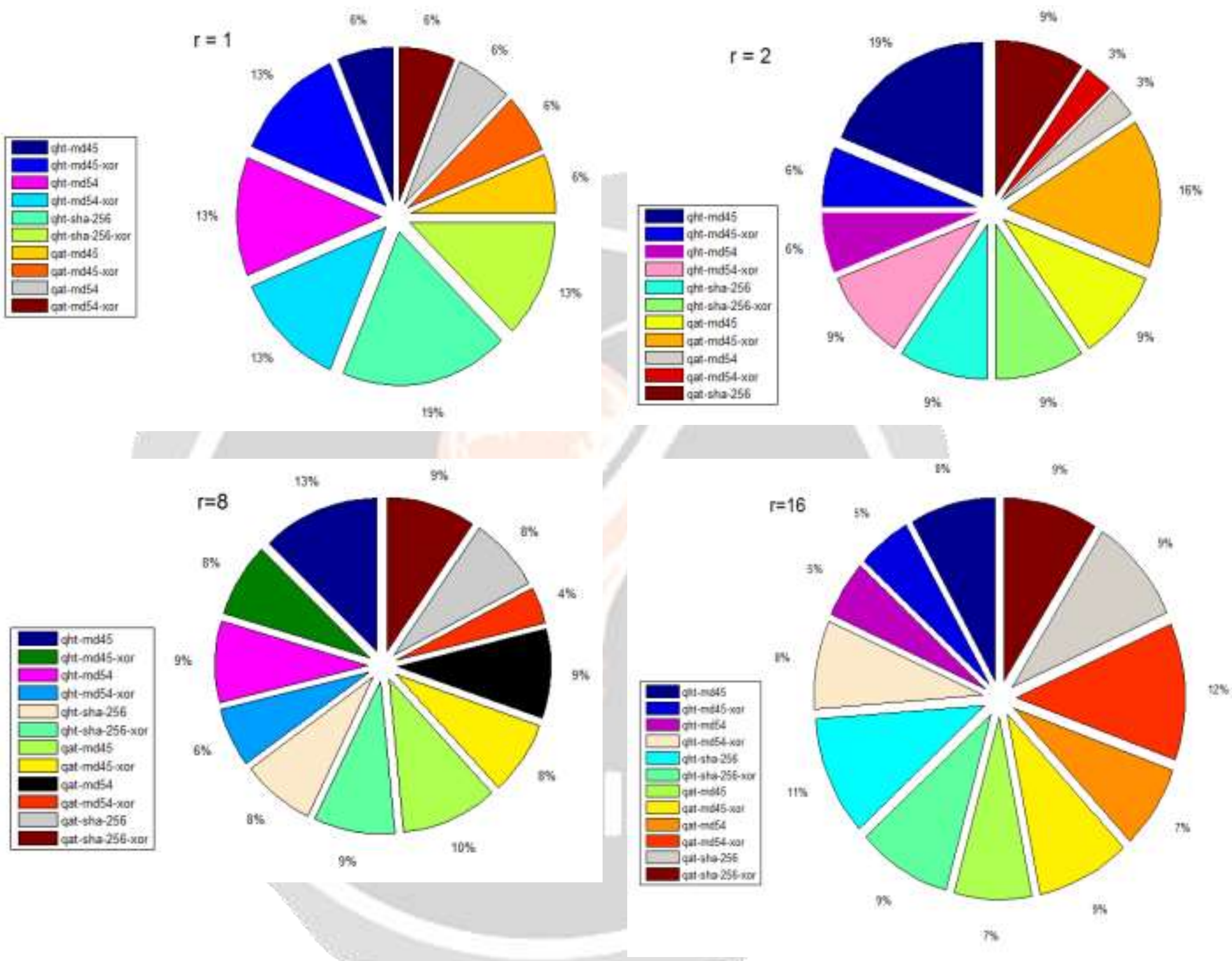

Fig 12- Extract of a chart of QPQ-CD

Interpretation :

The chart of selection specify how many times the type of algorithm of PostQuantum is used by the system varying the parameter r. If r is big, so the QPQ-CD will need more resources and more processing. By changing r, the Figure 12 conclude that all algorithm out of the PostQuantum Cryptography : qht_md45, qht_md45_xor, qht_md54, qht_md54_xor, qht_sha256, qht_sha256_xor, qat_md45, qat_md45_xor, qat_md54, qat_md54_xor, qat_sha256, qat_sha256_xor could be selected. So 12 algorithms with QPQ-CD must be implemented for having a chance to be selected by the selector.

4. Conclusion

The Post Quantum algorithm permit gives a multiple key of 256bits wich will be used for the selector of the next dynamic. The optimized PQC is obtained by changing one bit each other after the 12 outputs obtained PQC. An attacker could test possibilities and crack the key using the begin until the end resp. the end until the start and could know the next key. The probability of extremity evaluate this first attack. The second attack could be the same as the first but not compared to the begin or end but compared to the precedent key. This attack could also be evaluated with the probability of proximity. The two parameters couldn't be best at the same time. So, their value is between 50% and 100% for each other. A third attack could be the change resp. not change only n bits of the keys. The probability of bit change specified that the algorithm has a best security if it is 50. The optimization also permits this best condition. Like last parameters of evaluation, the entropy specify the disorder of the bit zeros and bit one of the key. The probability binary of entropy could be achieved near 100% for this method. A chart representative permit to conclude the importance of the 12 algorithms on the PQC. All of them have a chance to be selected by the last selector of the QPQ-CD.

5. Bibligraphies

[1] Y. Wu, H. Huang, C. Wang, Y. Pan, *« 5G Enabled Internet of Thing »*, CRC Press, 2019

[2] V. C. M. Leung, H. Zhang, X. Hu, Q. Liu, Z. Liu, *« 5G for Future Wireless Networks »*, ICST Institute for Computer Sciences, 2019

[3] W. Lei, Anthony C.K. Soong, L. Jianghua ,W. Yong , B. Classon, W. Xiao, D. Mazzarese, Z.Yang, T. Saboorian, *«5G System Design An End to End Perspective»*, Springer, 2020

[4] S. M. A. Kazmi, L. U. Khan, N. H. Tran, C. S. Hong, *« Network Slicing for 5G and Beyond Networks »*, Springer, 2019

[5] L. Song, Z. Xu, Z. Tian, J. Chen, R. Zhi, *« Research on 4G And 5G Authentication Signaling»*, International Journal Of Physics, 2019

[6]] R. Khan, P. Kumar, D. N. K. Jayakody, M. Liyanage, *« A Survey on Security and Privacy of 5G Technologies: Potential Solutions, Recent Advancements and Future Directions»*, Journal of IEEE, Juil. 2019

[7] R. Borgaonkar, L. Hirschi,, S. Park, and A. Shaik, *« New Privacy Threat on 3G, 4G, and Upcoming 5G AKA Protocols»*, Journal of Sciendo, 2019

[8] V. C. M. Leung, H. Zhang, X. Hu, Q. Liu, Z. Liu, *« 5G for Future Wireless Networks »*, ICST Institute for Computer Sciences, 2019

[9] M. Heigly, M. Schrammy, D. Fiala, *«A Lightweight Quantum-Safe Security Concept for Wireless Sensor Network Communication»*, Journal of IEEE 2019

[10] M S. Shoba, *«A Survey on Post Quantum Digital Signature Schemes for Blockchain»*, International Journal of Computer Science and Mobile Computing, June 2019