Practical UMAC Algorithm on Hybrid Crypto-Code Constructions of McElise on Shortened Mec

Alla Havrylova
PRACTICAL UMAC ALGORITHM ON HYBRID CRYPTO-CODE CONSTRUCTIONS OF McELISE ON SHORTENED MEC

A. HAVRYLOVA*

*Simon Kuznets Kharkiv National University of Economics, Department of Cyber Security and Information Technology, UKRAINE

*Corresponding Author: alla.gavrylova@hneu.net

ABSTRACT

A study was carried out on the use of an improved UMAC algorithm in post-quantum cryptography based on the formation of a substrate on the third layer of the hash code generation by the McElise crypto-code system on elliptic codes. The paper considers a practical algorithm for generating a hash code based on an example implementation of a cascading UMAC hash algorithm with the McElise crypto-code construction on elliptic codes. Using a hybrid crypto-code design allows you to save the universality of the hash code at the output of the algorithm, which allows its use in large databases as an identifier. In addition, in the context of the implementation of a full-scale quantum computer, US NIST experts consider crypto-code systems as one of the effective post-quantum cryptography algorithms. This approach allows you to implement the UMAC modification on various modifications of hybrid crypto-code structures and to ensure the formation of authentication profiles of different strength and length.

Keywords: UMAC hashing algorithm, McElice hybrid crypto code constructions, elliptic codes.

1. INTRODUCTION

An important direction in the development of post-quantum cryptography today is crypto-code systems (constructions) (CCC). Their formation is based on the use of algebraic codes disguised as the so-called random code [1], [2]. CCC allow integrated to implement fast cryptographic data conversion and ensure the reliability of the transmitted data based on noise-resistant coding [3], [4]. Despite the advantages, their use in modern software and hardware is hampered by their practical implementation with the required level of cryptographic stability, and withstand the attack of V.M. Sidelnikov on the basis of linear-fractional transformations, allowing to open a private key (generating and / or verification matrix, depending on the crypto-code system of McElise or Niederreiter) [5]. At the same time, according to experts of NIST USA, these crypto-code designs can provide the required level of protection and are able to withstand modern threats. This is confirmed by the participation of the McElice crypto code construction in the NIST contest for post-quantum cryptography algorithms. It seems interesting to explore the possibilities of sharing the already known cryptographic coding systems for transmitting information.

2. LITERATURE REVIEW

The development of computing capabilities in recent years, and in the first place, the creation of full-scale quantum computers, has jeopardized the use of classical mechanisms of not only symmetric cryptography, public key cryptography (including algorithms using the theory of elliptic curves), but also algorithms for providing authenticity services based on MDC and MAC codes, specialized hash functions [1], [3], [6], [7]. In the face of modern threats and the use of cryptanalysis algorithms using full-scale quantum computers, the use of the SHA-3 algorithm and the winning algorithms of the NESSIE European cryptographic contest in authentication and digital signature algorithms is questioned because of the possibility of hacking. Under such conditions, an increase in the level of cryptographic stability can lead to an increase in the length of key sequences and a decrease in the speed of
cryptographic transformations. The use of the UMAC algorithm with the formation of the substrate of the third layer based on MASH-2 leads to an increase in the level of stability, collisions, but also to a decrease in the conversion speed [8], which is an indirect confirmation of the possibility of reducing the speed of cryptographic transformations in the conditions of post-quantum cryptography. An urgent task is to increase the speed of cryptocurrencies while ensuring the required level of cryptographic stability of this algorithm. In [3], [4], practical algorithms for crypto-code constructions are considered that provide their practical implementation by reducing the power of the alphabet. Their application in the UMAC algorithm will not only provide the required level of cryptographic stability of the generated hash code, but also preserve its versatility.

**Research problem** – investigation of the possibility of using hybrid McEliece crypto-code constructions with shortened flawed elliptic codes based on a practical example in the UMAC algorithm.

3. **CONSTRUCTION OF A MODIFIED UMAC ALGORITHM USING HCCC ON THE BASIS OF MKKS McELICE FOR A SHORTED MEC**

In works [9], [10], a mathematical model and a structural diagram of the hash code generation in the UMAC algorithm were considered using, as an algorithm, a substrate (pseudo-random sequence that ensures the hash code cryptographic stability) of the McElise crypto code design using elliptic codes (EC) (modified elliptical codes (MEC), flawed codes).

The use of various algebraic and multi-channel cryptography codes will allow the formation of various hash code lengths and provide the required level of its cryptographic strength. The basic steps of creating a hash code are considered in the work [10].

Consider the practical implementation of the modified UMAC algorithm using the McEliece HCCC in the EC using an example. The input to the calculations is:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_{L_{11}}$</td>
<td>universal hash value (UHASH-hash) of the first level of hashing</td>
</tr>
<tr>
<td>$Y_{L_{31}}$</td>
<td>hash value (Carter-Wegman-hash) of the third level of hashing</td>
</tr>
<tr>
<td>$T$</td>
<td>data block</td>
</tr>
<tr>
<td>Blocklen</td>
<td>data block length (bytes)</td>
</tr>
<tr>
<td>$K$</td>
<td>secret key</td>
</tr>
<tr>
<td>Keylen</td>
<td>secret key length (32 bytes)</td>
</tr>
<tr>
<td>$Tag$</td>
<td>integrity and authenticity control code</td>
</tr>
<tr>
<td>$K_{L_{11}}$</td>
<td>secret key of the first level of hashing, consisting of subkeys $K_1, K_2, ..., K_n$</td>
</tr>
<tr>
<td>$K_{L_{31}}$</td>
<td>second-level hash secret key consisting of keys $K_{L_{31}}$ (subkeys $K_1, K_2, ..., K_n$) and $K_{L_{32}}$ (subkeys $K_1, K_2, ..., K_n$)</td>
</tr>
<tr>
<td>$M$</td>
<td>length of the transmitted plaintext array $I$</td>
</tr>
<tr>
<td>$K'$</td>
<td>pseudo random key sequence</td>
</tr>
<tr>
<td>Numbyte</td>
<td>pseudo-random key sequence length (number of subkeys)</td>
</tr>
<tr>
<td>Index</td>
<td>subkey number</td>
</tr>
<tr>
<td>$I=11$</td>
<td>transmitted plaintext ($k$-bit information vector over $GF (q)$)</td>
</tr>
<tr>
<td>$Xor$</td>
<td>bitwise summation</td>
</tr>
<tr>
<td>$x^2+y^2+z^2=0$</td>
<td>algebraic curve over the field $GF (2^2)$</td>
</tr>
<tr>
<td>$e=000000200$</td>
<td>secret weight error vector $W_h(e) \leq t = \left\lfloor \frac{d-1}{2} \right\rfloor$</td>
</tr>
<tr>
<td>$X = \begin{bmatrix} 1 &amp; 2 \ 3 &amp; 0 \end{bmatrix}$</td>
<td>nondegenerate $k \times k$ matrix</td>
</tr>
</tbody>
</table>
| \( P = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix} \) | permutation matrix of size \( n \times n \)
| \( D = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \) | diagonal matrix equal 1
| \( G = \begin{bmatrix}
2 & 2 & 3 & 0 & 1 & 3 & 0 & 1 \\
3 & 3 & 2 & 1 & 0 & 2 & 1 & 0
\end{bmatrix} \) | generating matrix
| \( Taglen \) | the length of the integrity control code (authenticity) \( PadCx \) (4 bytes)
| \( Nonce \) | unique number for input message \( I \) (8 bytes)
| \( Numbyte \) | subkey length (equal to \( Keylen \))
| \( Index \) | subkey number (0)
| \( Cx=23023322 \) | cryptogram
| \( P^{-1} = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix} \) | matrix inverse to the permutation matrix (since its determinant is 1, then \( P^{-1} = P^T \))
| \( D^{-1} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \) | the inverse of the diagonal matrix \( D \) – is a unipotent matrix (a square matrix, all eigenvalues are 1), which preserves the Hamming weight of the vector \( e \)
| \( X^{-1} = \begin{bmatrix}
0 & 2 \\
3 & 1
\end{bmatrix} \) | matrix inverse of a non-degenerate matrix \( X \)
Algebraic curve points:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3.1. Hash code generation in the algorithm UMAC
The creation of a hash for an open message is carried out in parallel with the formation of the codogram, but we will describe the computational transformations according to these actions in sequence.

According to the block diagram of the iterative formation of Y, Pad, and Tag for an open message from the sender using the UMAC algorithm [9], [10], we distinguish the following calculation steps.

3.1.1. 1st layer formation
The value of the first level hash function \( UHASH = Hash_{L1} \) we will calculate by the formula:

\[
Y_{L1} = H_{L1} \left( K_{L1}, I \right)
\]

To form \( K_{L1} \) imagine it as a key sequence of four-byte subunits:

\[
K_{L1} = K_1 || K_2 || \ldots || K_n,
\]

where || is the concatenation (joining) of the strings corresponding to the subkeys.

The amount of subkey data depends on the values Numbyte and Blocklen:

\[
n = \left[ \frac{\text{Numbyte}}{\text{Blocklen}} \right] = \frac{1024 \times 16 \times 3}{32} = \frac{1072}{32} = 33.5 \approx 33 \Rightarrow i = 1, 2, \ldots, 33.
\]

Because the \( T_i = \text{Index} || i \), then for the first layer \( \text{Index} = 1 \), \( \Rightarrow T_i : \)

\[
T_1 = 1 || 1 = 00000001 \quad T_{17} = 1 || 17 = 00000001
00000001 \Rightarrow \text{K1}
00100001 \Rightarrow \text{K17}
T_2 = 1 || 2 = 00000001 \quad T_{18} = 1 || 18 = 00000001
00000000 \Rightarrow \text{K2}
00100010 \Rightarrow \text{K18}
T_3 = 1 || 3 = 00000001 \quad T_{19} = 1 || 19 = 00000001
00000001 \Rightarrow \text{K3}
00100011 \Rightarrow \text{K19}
T_4 = 1 || 4 = 00000001 \quad T_{20} = 1 || 20 = 00000001
00000010 \Rightarrow \text{K4}
00101000 \Rightarrow \text{K20}
T_5 = 1 || 5 = 00000001 \quad T_{21} = 1 || 21 = 00000001
00000011 \Rightarrow \text{K5}
00101010 \Rightarrow \text{K21}
T_6 = 1 || 6 = 00000001 \quad T_{22} = 1 || 22 = 00000001
00010110 \Rightarrow \text{K6}
T_7 = 1 || 7 = 00000001 \quad T_{23} = 1 || 23 = 00000001
00010011 \Rightarrow \text{K7}
T_8 = 1 || 8 = 00000001 \quad T_{24} = 1 || 24 = 00000001
00010100 \Rightarrow \text{K8}
T_9 = 1 || 9 = 00000001 \quad T_{25} = 1 || 25 = 00000001
00010101 \Rightarrow \text{K9}
T_{10} = 1 || 10 = 00000001 \quad T_{26} = 1 || 26 = 00000001
00110010 \Rightarrow \text{K10}
T_{11} = 1 || 11 = 00000001 \quad T_{27} = 1 || 27 = 00000001
00110011 \Rightarrow \text{K11}
T_{12} = 1 || 12 = 00000001 \quad T_{28} = 1 || 28 = 00000001
00111000 \Rightarrow \text{K12}
00111000 \Rightarrow \text{K12}
Based on the length $M$ of the input message ($M = 3$ bytes), the number of blocks is $T = 1$, therefore, the number of subkeys on this layer is the same. Wherein $K_{LI} = T_i = 0000000100000001$.

The hash values of this layer are calculated using the following formula:

$$Y_{Li} = (I + K_{Li}) \mod 32$$

$$Y_{Li} = (0100110+10000001)\mod32 = 111$$

### 3.1.2. 2nd layer formation.
Since the length of $M$ is less than 1024 bytes, this level of hashing will not be performed, and we will perform calculations using the hash code of the third level.

### 3.1.3. 3rd layer formation.
Number of subkeys for $K_{L31}$ and $K_{L32}$ also depends on the values $Numbyte$ and $Blocklen$.

Number of subkeys for $K_{L31}$:

$$n = \left\lfloor \frac{Numbyte}{Blocklen} \right\rfloor = 64 \times 4 \mod 32 = 8 \Rightarrow i = 1, 2, 3, 4, 5, 6, 7, 8$$

Therefore, to form $K_{L31}$ imagine it as a key sequence of eight four-byte subunits:

$$K_{L31} = K_{i1} || K_{i2} || K_{i3} || K_{i4} || K_{i5} || K_{i6} || K_{i7} || K_{i8}$$

For the third layer at Index $= 3$, $\Rightarrow T_i$:

$T_1 = 3 \parallel 1 = 00000001 \, 00000001 \Rightarrow K_{11}$

$T_2 = 3 \parallel 2 = 00000001 \, 00000010 \Rightarrow K_{21}$

$T_3 = 3 \parallel 3 = 00000001 \, 00000011 \Rightarrow K_{31}$

$T_4 = 3 \parallel 4 = 00000001 \, 00000100 \Rightarrow K_{41}$

$T_5 = 3 \parallel 5 = 00000011 \, 00000101 \Rightarrow K_{51}$

$T_6 = 3 \parallel 6 = 00000110 \, 00000110 \Rightarrow K_{61}$

$T_7 = 3 \parallel 7 = 00000111 \, 00000111 \Rightarrow K_{71}$

$T_8 = 3 \parallel 8 = 00000001 \, 00001000 \Rightarrow K_{81}$

Number of subkeys for $K_{L32}$:

$$n = \left\lfloor \frac{Numbyte}{Blocklen} \right\rfloor = 4 \times 4 \mod 32 = 0, \Rightarrow i = 1$$

To form $K_{L32}$ imagine it as a key sequence of 1 four-byte sub-block:

$$K_{L32} = K_{i1}$$

For the third layer at Index $= 4$, $\Rightarrow T_i$: 

$$T_1 = 4 \parallel 1 = 00000001 \, 00000001 \Rightarrow K_{11}$$

$$T_2 = 4 \parallel 2 = 00000001 \, 00000010 \Rightarrow K_{21}$$

$$T_3 = 4 \parallel 3 = 00000001 \, 00000011 \Rightarrow K_{31}$$

$$T_4 = 4 \parallel 4 = 00000001 \, 00000100 \Rightarrow K_{41}$$

$$T_5 = 4 \parallel 5 = 00000011 \, 00000101 \Rightarrow K_{51}$$

$$T_6 = 4 \parallel 6 = 00000110 \, 00000110 \Rightarrow K_{61}$$

$$T_7 = 4 \parallel 7 = 00000111 \, 00000111 \Rightarrow K_{71}$$

$$T_8 = 4 \parallel 8 = 00000001 \, 00001000 \Rightarrow K_{81}$$

$$T_9 = 4 \parallel 1 = 00000001 \, 00000001 \Rightarrow K_{91}$$
The hash value of the third layer is calculated using the following formula:

\[ Y_{L,3I} = ((I + K_I) \mod 32) \mod 2^{32} \times \text{xor} Y_{L,3I} = \]

\[ ((l_1 \mod (2^{36} - 5)) \mod 2^{32}) \times \text{xor} Y_{L,3I} \]

\[ Y_{L,3I} = ((11 \mod (2^{36} - 5)) \mod 2^{32}) \times \text{xor} 00000100 00000001 = 1000000010 \]

3.2.1. Pad Shaping

1) The recipient generates a public key, which in the McEliece cryptosystem is the matrix [3]:

\[ G^*_X = X \times G^{EC} \times P \times D \]

\[ G^*_X \times \begin{bmatrix} 1 & 2 & 3 \ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ 0 & 0 & 0 & 0 & 1 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 1 & 0 \ \end{bmatrix} \times \]

\[ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ \end{bmatrix} \]

2) The cryptogram (codogram) formed from the information message \( I \) is a vector of length \( n \), which is calculated by the following formula:

\[ C^*_X = I \times G^*_X \oplus e , \]

where is the vector \( I \times G^*_X \) is the codeword of the masked code, i.e. belongs to the \( (n, k, d) \)-code with the generating matrix \( G^*_X \); the vector \( e \) is a one-time session secret key.
3) We form the initialization vector $IV = 00100000$ for the recipient and sender. This vector shows the location of the code sequence reduction:

$$C_X^* = 11 \times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \oplus 0000200 = 2302322$$

$$C_X^* = 232322$$

4) Damage to the initial text based on the conversion Table 1.

<table>
<thead>
<tr>
<th>Word (shuffled)</th>
<th>Residue length</th>
<th>$C(x)$</th>
<th>$F(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>2</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>2</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>2</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>2</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

Initial text (word): $C_x^* = 232322_{10} = 010 011 000 010 011 011 010 010_2$.

5) Sending damage (flag) by the first channel to the recipient, sending the flawed code (balance) by the second channel to the recipient.

We get $C_x^* = 1011001011111010_2$

Convert to decimal notation: $545750_{10}$ — enters the first channel.

Flags received $F(x) = 111111111_2$.

When converted to decimal, we get: $777_{10}$ — enters the second channel.

3.2.2. The formation of a pseudo-random lining (substance) using the function PDF

To ensure the cryptographic stability of the UMAC algorithm at the level of stability of the used cryptographic algorithm, we form a $PadCx$ pseudo-random pad for $I$ using the function PDF:

$$Pad = PDF(K, Nonce, Taglen)$$

According to the pseudo-random lining formation procedure $Pad$ for $I$, it is necessary to form the following subkey, presented as a function KDF [8–10]:

$$K' = KDF(K, Index, Numbyte)$$
\[ K' = KDF(0106, 0, 4) \]

Pseudo-random lining \( Pad \) will have the form:

\[ Pad = PDF(0106, 8, 4) = 1101010 \]

As a result of the formation of the substrate, various parts of it can be used as an additional initialization vector.

4. HASH CODE VERIFICATION AT THE RECEPTION SIDE USING AN ALGORITHM UMAC

4.1. Generating a validity code for a received message

Generation of authentication codes of the received message is possible according to the formula [9, 10]:

\[
Tag = UMAC(K, I, Nonce, Taglen) = \text{Hash}(K, I, Taglen) \oplus \text{PDF}(K, Nonce, Taglen) = Y_{L3M} \oplus Pad
\]

\[ Tag = 1000000010 \oplus 1101010 \oplus 11011101100 = 10011101100 \]

To generate a summary code of the reliability of the transmitted text, we will use the found value of the hash code \( Y_{L3M} \) and code authentication code \( Tag \) plaintext sender:

\[ Y = Y_{L3M} \oplus Tag \]

\[ Y = 1000000010 \oplus 10011101100 = 1101110 = 110_{10} \]

4.2. Decoding a received message

1) Recover received text

The resulting values from two channels are translated into a binary number system:

\[ C^*_x = 545750_{10} = 1011001011111010_2 \]

\[ F(x) = 777_{10} = 11111111_2 \]

2) Loss recovery

Using Table 1 we get the code word:

\[ C^*_x = 010 011 000 010 011 011 010 010 = 2323322_{10} \]

3) To restore closed text, the recipient adds null information characters to the location indicated by the initialization vector \( IV \):

\[ C^*_t = 2323322 \rightarrow 23023322 \]

4) With recovered closed text \( C_t \) remove the action of secret permutation and diagonal matrices:

\[ C^*_s = C_s \times D^{-1} \times P^{-1} \]
5) We find the syndrome and polynomial of error locators:

\[
S = C^*_x \times H^{EC^T}
\]

\[
S = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 2 & 3 & 1 & 2 & 3 \\
0 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\
0 & 0 & 1 & 3 & 2 & 1 & 3 & 2 \\
0 & 0 & 2 & 3 & 1 & 3 & 1 & 2 \\
0 & 1 & 3 & 3 & 3 & 2 & 2 & 2
\end{bmatrix}
\]

We find a syndrome:

\[
S = (1,1,1,0,0,0);
\]

Find the polynomial of error locators \( \Lambda (x) = a_{00} + a_{10}x + y = 0 \)

\[
\begin{bmatrix}
S_{00} & S_{10} \\
S_{10} & S_{20}
\end{bmatrix} \times \begin{bmatrix} a_{00} \\ a_{01} \end{bmatrix} = \begin{bmatrix} S_{00} \\ S_{10} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}
\]

\[a_{00}=0; \ a_{10}=1;\]

\( \Lambda (xy) = x+y= 0 \) \( \) – error locator polynomial

6) We find error locators according to Chen’s procedure:

\[
P_1(0,0,1) \ \Lambda (x,y) = 0+0=0 \) \( \) – error
\[
P_2(0,1,1) \ \Lambda (x,y) = 0+1=1
\]
\[
P_3(1,2,1) \ \Lambda (x,y) = 1+2=3
\]
\[
P_4(2,2,1) \ \Lambda (x,y) = 2+2=0 \) \( \) – error
\[
P_5(3,2,1) \ \Lambda (x,y) = 3+2=1
\]
\[
P_6(1,3,1) \ \Lambda (x,y) = 1+3=2
\]
\[
P_7(2,3,1) \ \Lambda (x,y) = 2+3=1
\]
\[
P_8(3,3,1) \ \Lambda (x,y) = 3+3=0 \) \( \) – error

\[e^* = e_700e_400e_8\]
We find: \( e^* \times H^{EC^T} = S \), solving the system of equations, we obtain: \( e_1 = 0, e_4 = 2, e_8 = 3 \)

\[ e^* = 00020003 \]

We find \( i^* = e^* + C^*_x \)

\[ i^* = 00020003 \oplus 22102221 = 22; \]

7) Find plain text:

\[ i = i^* \times X^i, i = 22 \times \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} = 11. \]

4.3. Hash verification

The authorized user (recipient) generates in accordance with paragraph 3.1 - 4.1 hash code. Verification is carried out by comparison, received from the sender and generated by the recipient of the hash codes. If they coincide, a decision is made that the plaintext received through the open channel is not modified.

5. CONCLUSION

As a result of the research, practical algorithms for generating a hash code and its verification based on the UMAC algorithm using the McEliece hybrid crypto-code constructions on the MEC were developed. This mechanism of message authenticity can be used not only on defective shortened codes, but also on elongated ones. This approach can significantly increase the relative data transfer rate, which will positively affect the practical implementation of a fast hashing algorithm with a given level of strength in post-quantum cryptography.

REFERENCES


