Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Departamento de computación

Tópicos selectos de Criptografía

Tarea 3: Cifrado autenticado con OCB

Tarea

Autor:
André Fabián Castellanos Aldama
Matrícula:
211270011

2022-07-07

Índice

1. Resumen

Se presenta una implementación de cifrado autenticado usando OCB de datos asociados del artículo de Jha A., Mancillas-Lopez C., Nandi M., et al usando instrucciones intrinsics AES. La implementación se puede encontrar en link.

2. Cifrado autenticado usando OCB

De manera simple este modo de operación puede verse con las siguientes figuras tomadas del artículo:

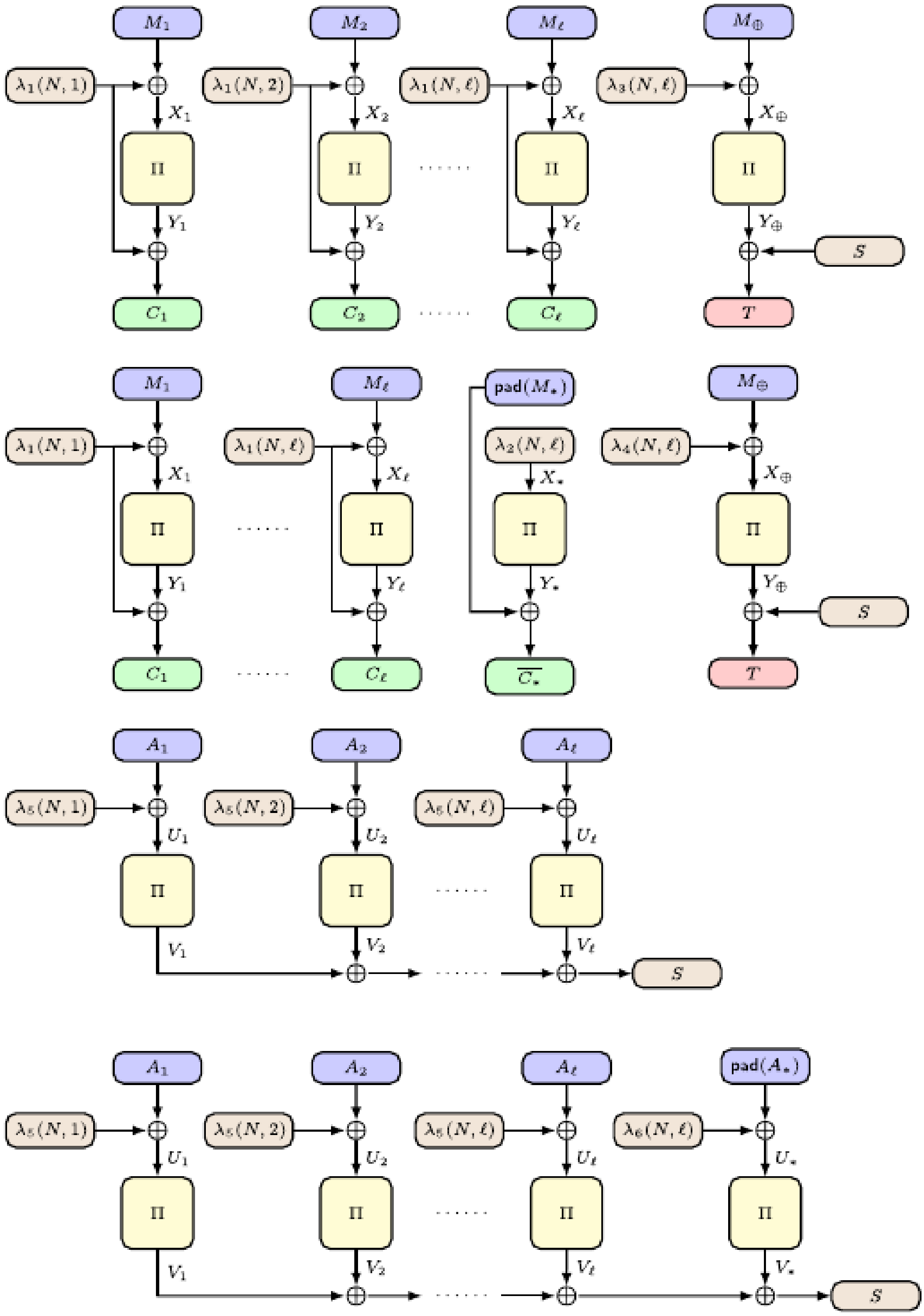

Figura 1: Diagrama del algoritmo.

3. Implementación

Por razones de tiempo se implementó en bloques completos sin usar padding.

3.1. Cifrado y creación de tag

En esta etapa inicial ciframos el **Nonce** para obtener **L** que será usado durante el cifrado y autenticado. En las líneas *14-45* realizamos la primera etapa que se reflejan en las líneas *6-9* en el algoritmo del artículo. Después de esto realizamos el cifrado del mensaje que corresponden a las líneas *49-85* que se reflejan en las líneas *14-18* del artículo. Por último, calculamos el tag en las líneas *88-112* que se reflejan en las líneas *30-33* del artículo. (En este caso nos referimos al algoritmo encrypt)

```
1  /* ------------------------------------ Initializing L ------------------------------------ */
2    // Initialize L with nonce
3    __m128i L = _mm_loadu_si128((__m128i *) NONCE);
4    // normal cipher nonce (10 rounds)
5    L = _mm_xor_si128(L, ((__m128i *) key1.KEY)[0]);
6    int j;
7    for (j = 1; j < 10; j++) {
8      L = _mm_aesenc_si128(L, ((__m128i *) key1.KEY)[j]);
9    }
10   L = _mm_aesenclast_si128(L, ((__m128i *) key1.KEY)[j]);
11
12   /* ------------------------------------ Hashing ------------------------------------ */
13   // Assuming complete blocks only
14   int complete_blocks = (length_associated_data_in_bytes - length_associated_data_in_bytes % 16) / 16;
15   int indexBlock;
16   int ctr = 1;
17   __m128i S = _mm_set_epi32(0, 0, 0, 0);
18   for (indexBlock = 0; indexBlock < complete_blocks; indexBlock++) {
19     // Load A_i 16 * 16 bits of A
20     __m128i A_i, U_i, V_i;
21     A_i = _mm_loadu_si128(&((__m128i *) ASSOCIATEDDATA)[indexBlock]);
22     // Create alpha and delta5 with key 2
23     __m128i alpha, delta5;
24     alpha = _mm_set_epi32(ctr, ctr, ctr, ctr);
25     alpha = _mm_xor_si128(alpha, L);
26     // cipher alpha (2 rounds)
27     alpha = _mm_xor_si128(alpha, ((__m128i *) key2.KEY)[0]);
28     for (j = 1; j <= 2; j++) {
29       alpha = _mm_aesenc_si128(alpha, ((__m128i *) key2.KEY)[j]);
30     }
31     delta5 = alpha;
32     ctr += 5;
33
34     // Calculate U_i
35     U_i = _mm_xor_si128(A_i, delta5);
36
37     // Cipher U_i with two rounds AES. Calculate V_i
38     V_i = _mm_xor_si128(U_i, ((__m128i *) key2.KEY)[0]);
39     for (j = 1; j <= 2; j++) {
40       V_i = _mm_aesenc_si128(V_i, ((__m128i *) key2.KEY)[j]);
41     }
42
43     // Calculate S
44     S = _mm_xor_si128(S, V_i);
45   }
46
```

```
/* ------------------------------------ Encryption
 ------------------------------------ */
// Assuming complete blocks only
complete_blocks = (length_plaintext_in_bytes -
 length_plaintext_in_bytes % 16) / 16;
ctr = 1;
__m128i M_xor = _mm_set_epi32(0, 0, 0, 0);
for (indexBlock = 0; indexBlock < complete_blocks; indexBlock++) {
  // Load M_i 16 * 16 bits of M
  __m128i M_i, X_i, Y_i, C_i;
  M_i = _mm_loadu_si128(&((__m128i *) PLAINTEXT)[indexBlock]);

  // Create alpha and delta1_i with key 2
  __m128i alpha, delta1;
  alpha = _mm_set_epi32(ctr, ctr, ctr, ctr);
  alpha = _mm_xor_si128(alpha, L);
  // cipher alpha (2 rounds)
  alpha = _mm_xor_si128(alpha, ((__m128i *) key2.KEY)[0]);
  for (j = 1; j <= 2; j++) {
    alpha = _mm_aesenc_si128(alpha, ((__m128i *) key2.KEY)[j]);
  }
  delta1 = alpha;
  ctr++;

  // Calculate M_xor
  M_xor = _mm_xor_si128(M_xor, M_i);

  // Calculate X_i
  X_i = _mm_xor_si128(M_i, delta1);

  // Cipher X_i to Y_i
  Y_i = _mm_xor_si128(X_i, ((__m128i *) key1.KEY)[0]);
  for (j = 1; j < 10; j++) {
    Y_i = _mm_aesenc_si128(Y_i, ((__m128i *) key1.KEY)[j]);
  }
  Y_i = _mm_aesenclast_si128(Y_i, ((__m128i *) key1.KEY)[j]);

  // Calculate C_i
  C_i = _mm_xor_si128(Y_i, delta1);
  _mm_storeu_si128(&((__m128i*)CIPHERTEXT)[indexBlock],C_i);
}

// Create alpha and delta3_l with key 2
__m128i alpha, delta3_l;
alpha = _mm_set_epi32(3*complete_blocks + 1, 3*complete_blocks + 1,
 3*complete_blocks + 1, 3*complete_blocks + 1);
alpha = _mm_xor_si128(alpha, L);
// cipher alpha (2 rounds)
alpha = _mm_xor_si128(alpha, ((__m128i *) key2.KEY)[0]);
for (j = 1; j <= 2; j++) {
  alpha = _mm_aesenc_si128(alpha, ((__m128i *) key2.KEY)[j]);
}
delta3_l = alpha;

// Calculate X_xor
__m128i X_xor = _mm_xor_si128(M_xor, delta3_l);

// Calculate Y_xor ciphering X_xor (two rounds)
__m128i Y_xor = _mm_xor_si128(X_xor, ((__m128i *) key2.KEY)[0]);
for (j = 1; j <= 2; j++) {
  Y_xor = _mm_aesenc_si128(Y_xor, ((__m128i *) key2.KEY)[j]);
```

```
  }

  // Calculate tag
  __m128i T;
  T = _mm_xor_si128(S, delta3_l);
  T = _mm_xor_si128(T, Y_xor);

  _mm_storeu_si128 (&((__m128i*)TAG)[0],T);
```

Listing 1: Implementación de cifrado y tag

3.2. Descifrado y recreación de tag

En esta etapa final en las líneas *4-34* realizamos la primera etapa que se reflejan en las líneas *6-9* en el algoritmo del artículo. Después de esto realizamos el descifrado del mensaje que corresponden a las líneas *38-73* que se reflejan en las líneas *14-18* del artículo. Por último, calculamos el tag en las líneas *76-88* que se reflejan en las líneas *29-32* del artículo. (En este caso nos referimos al algoritmo decrypt)

```
1  /* ------------------------------------ Decryption part ------------------------------------ */
2    /* ------------------------------------ Hashing ------------------------------------ */
3    // Assuming complete blocks only
4    complete_blocks = (length_associated_data_in_bytes - length_associated_data_in_bytes % 16) / 16;
5    ctr = 1;
6    S = _mm_set_epi32(0, 0, 0, 0);
7    for (indexBlock = 0; indexBlock < complete_blocks; indexBlock++) {
8      // Load A_i 16 * 16 bits of A
9      __m128i A_i, U_i, V_i;
10     A_i = _mm_loadu_si128(&((__m128i *) ASSOCIATEDDATA)[indexBlock]);
11     // Create alpha and delta5 with key 2
12     __m128i alpha, delta5;
13     alpha = _mm_set_epi32(ctr, ctr, ctr, ctr);
14     alpha = _mm_xor_si128(alpha, L);
15     // cipher alpha (2 rounds)
16     alpha = _mm_xor_si128(alpha, ((__m128i *) key2.KEY)[0]);
17     for (j = 1; j <= 2; j++) {
18       alpha = _mm_aesenc_si128(alpha, ((__m128i *) key2.KEY)[j]);
19     }
20     delta5 = alpha;
21     ctr += 5;
22
23     // Calculate U_i
24     U_i = _mm_xor_si128(A_i, delta5);
25
26     // Cipher U_i with two rounds AES. Calculate V_i
27     V_i = _mm_xor_si128(U_i, ((__m128i *) key2.KEY)[0]);
28     for (j = 1; j <= 2; j++) {
29       V_i = _mm_aesenc_si128(V_i, ((__m128i *) key2.KEY)[j]);
30     }
31
32     // Calculate S
33     S = _mm_xor_si128(S, V_i);
34   }
35
36   /* ------------------------------------ Decryption ------------------------------------ */
37   // Assuming complete blocks only
38   complete_blocks = (length_plaintext_in_bytes - length_plaintext_in_bytes % 16) / 16;
39   ctr = 1;
```

```
M_xor = _mm_set_epi32(0, 0, 0, 0);
for (indexBlock = 0; indexBlock < complete_blocks; indexBlock++) {
  // Load M_i 16 * 16 bits of M
  __m128i M_i, X_i, Y_i, C_i;
  C_i = _mm_loadu_si128(&((__m128i *) CIPHERTEXT)[indexBlock]);
  // Create alpha and delta1_i with key 2
  __m128i delta1;
  alpha = _mm_set_epi32(ctr, ctr, ctr, ctr);
  alpha = _mm_xor_si128(alpha, L);
  // cipher alpha (2 rounds)
  alpha = _mm_xor_si128(alpha, ((__m128i *) key2.KEY)[0]);
  for (j = 1; j <= 2; j++) {
    alpha = _mm_aesenc_si128(alpha, ((__m128i *) key2.KEY)[j]);
  }
  delta1 = alpha;
  ctr++;

  // Calculate Y_i
  Y_i = _mm_xor_si128(C_i, delta1);

  // Decipher Y_i to X_i
  X_i = _mm_xor_si128(Y_i, ((__m128i *) decrypt_key1.KEY)[0]);
  for (j = 1; j < 10; j++) {
    X_i = _mm_aesdec_si128(X_i, ((__m128i *) decrypt_key1.KEY)[j]);
  }
  X_i = _mm_aesdeclast_si128(X_i, ((__m128i *) decrypt_key1.KEY)[j])
;

  // Calculate M_i
  M_i = _mm_xor_si128(X_i, delta1);
  _mm_storeu_si128(&((__m128i*)DECRYPTEDTEXT)[indexBlock],M_i);

  // Calculate M_xor
  M_xor = _mm_xor_si128(M_xor, M_i);
}

// Calculate X_xor
X_xor = _mm_xor_si128(M_xor, delta3_l);

// Calculate Y_xor ciphering X_xor (two rounds)
Y_xor = _mm_xor_si128(X_xor, ((__m128i *) key2.KEY)[0]);
for (j = 1; j <= 2; j++) {
  Y_xor = _mm_aesenc_si128(Y_xor, ((__m128i *) key2.KEY)[j]);
}

// Calculate tag
T = _mm_xor_si128(S, delta3_l);
T = _mm_xor_si128(T, Y_xor);

_mm_storeu_si128 (&((__m128i*)TAG)[0],T);
```

Listing 2: Implementación de descifrado y recreación de tag

4. Pruebas

Se introducen ambas llaves, datos en claro, datos asociados, donde formamos un cifrado y un tag que después desciframos y verficamos el tag. A continuación se presentan los vectores de prueba y la salida donde se prueba el funcionamiento correcto del algoritmo implementado:

```
Vectores de prueba:
ALIGN16 uint8_t AES_128_TEST_KEY1[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
```

```
                                           0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
ALIGN16 uint8_t AES_128_TEST_KEY2[] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
                                       0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x34};
ALIGN16 uint8_t AES_128_TEST_NONCE[] = {0x2b, 0x71, 0x15, 0x15, 0x28, 0xae, 0xd2, 0xa6,
                                        0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x34};

ALIGN16 uint8_t GOCB_TEST_PLAINTEXT[] = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                         0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                         0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
                                         0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                         0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11,
                                         0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                         0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17,
                                         0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10};
ALIGN16 uint8_t GOCB_TEST_ASSOCIATED_DATA[] = {0x64, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                               0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                               0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
                                               0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
                                               0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11,
                                               0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
                                               0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17,
                                               0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x11};

Salida:
The Cipher Key 1:
                                        [0x2b7e151628aed2a6abf7158809cf4f3c]
The Cipher Key 2:
                                        [0x2b7e151628aed2a6abf7158809cf4f34]
The Key 1 Schedule:
                                        [0x2b7e151628aed2a6abf7158809cf4f3c]
                                        [0xa0fafe1788542cb123a339392a6c7605]
                                        [0xf2c295f27a96b9435935807a7359f67f]
                                        [0x3d80477d4716fe3e1e237e446d7a883b]
                                        [0xef44a541a8525b7fb671253bdb0bad00]
                                        [0xd4d1c6f87c839d87caf2b8bc11f915bc]
                                        [0x6d88a37a110b3efddbf98641ca0093fd]
                                        [0x4e54f70e5f5fc9f384a64fb24ea6dc4f]
                                        [0xead27321b58dbad2312bf5607f8d292f]
                                        [0xac7766f319fadc2128d12941575c006e]
The Key 2 Schedule:
                                        [0x2b7e151628aed2a6abf7158809cf4f34]
                                        [0xa0fa0d178854dfb123a3ca392a6c850d]
The decrypt Key 1 Schedule:
                                        [0xd014f9a8c9ee2589e13f0cc8b6630ca6]
                                        [0x0c7b5a631319eafeb0398890664cfbb4]
                                        [0xdf7d925a1f62b09da320626ed6757324]
                                        [0x12c07647c01f22c7bc42d2f37555114a]
                                        [0x6efcd876d2df54807c5df034c917c3b9]
                                        [0x6ea30afcbc238cf6ae82a4b4b54a338d]
                                        [0x90884413d280860a12a128421bc89739]
                                        [0x7c1f13f74208c219c021ae480969bf7b]
                                        [0xcc7505eb3e17d1ee82296c51c9481133]
                                        [0x2b3708a7f262d405bc3ebdbf4b617d62]
The PLAINTEXT:
                                        [0x6bc1bee22e409f96e93d7e117393172a]
                                        [0xae2d8a571e03ac9c9eb76fac45af8e51]
                                        [0x30c81c46a35ce411e5fbc1191a0a52ef]
                                        [0xf69f2445df4f9b17ad2b417be66c3710]
The ASSOCIATEDDATA:
                                        [0x64c1bee22e409f96e93d7e117393172a]
                                        [0xae2d8a571e03ac9c9eb76fac45af8e51]
```

```
                                        [0x30c81c46a35ce411e5fbc1191a0a52ef]
                                        [0xf69f2445df4f9b17ad2b417be66c3711]


The CIPHERTEXT:
                                        [0x0db5133aef86c48ec39886223cf975ce]
                                        [0xec5674fd96050f7e2e3a690055f6fe1f]
                                        [0xf1f1b7d7932695b48b3065db2e0c5fea]
                                        [0x8a9d26dcca8039834dad4e65be45f3ac]
TAG:
                                        [0x4f6fb04967144313f769b674af6e0eb8]


The DECIPHERTEXT:
                                        [0x6bc1bee22e409f96e93d7e117393172a]
                                        [0xae2d8a571e03ac9c9eb76fac45af8e51]
                                        [0x30c81c46a35ce411e5fbc1191a0a52ef]
                                        [0xf69f2445df4f9b17ad2b417be66c3710]
TAG:
                                        [0x4f6fb04967144313f769b674af6e0eb8]
```