CORE Working Group Working Group   P. Urien
Internet Draft   Telecom Paris
Intended status: Experimental

June 20 2023
Expires: December 2023

Bijective MAC for Constraint Nodes
draft-urien-core-bmac-12.txt

Abstract

In this draft context, things are powered by micro controllers units (MCU) comprising a set of memories such as static RAM (SRAM), FLASH and EEPROM. The total memory size, ranges from 10KB to a few megabytes. In this context code and data integrity are major security issues, for the deployment of Internet of Things infrastructure. The goal of the bijective MAC (bMAC) is to compute an integrity value, which cannot be guessed by malicious software.
In classical keyed MACs, MAC is computing according to a fixed order.
In the bijective MAC, the content of N addresses is hashed according to a permutation P (i.e. bijective application).
The bijective MAC key is the permutation P.
The number of permutations for N addresses is N!. So the computation of the bMAC requires the knowledge of the whole space memory; this is trivial for genuine software, but could very difficult for corrupted software, especially for time stamped bMAC.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Table of Contents

1 Overview

In this draft context, things are powered by micro controllers units (MCU) comprising a set of memories such as static RAM (SRAM), FLASH and EEPROM. The total memory size ranges from 10KB to a few megabytes.
In this context code and data integrity is a major security issue for the deployment of Internet of Things infrastructure.
The goal of the bijective MAC (bMAC) is to compute an integrity value, which cannot be guessed by malicious software.
In classical keyed MACs, MAC is computing according to a fixed order.
In the bijective MAC, the content of N addresses (A[0]...A[N-1]) is hashed according to a hash function H and a permutation P (i.e. bijective application in [0,N-1])so that :

```
bMAC(A, P) = H( A[P(0)] || A[P(1)] ... || A[P(N-1)] )
```

The bijective MAC key is the permutation P. The number of permutations for N addresses is N!, as an illustration 35! is greater than 2**128. So the bMAC computation requires the knowledge of the whole space memory. This is trivial for genuine software, but could very difficult for corrupted software, especially for time stamped bMAC.

2 Bijective MAC

2.1 Memory space

The memory space is represented by an application A, working with N addresses, whose content is a byte value.

```
  | [0,N-1] -> [0,255]
A |
  | x       -> A[x]
```

Non volatile memories (FLASH, EEPROM) MUST be included in the memory space. A subset of SRAM is included in the memory, whose structure relies on operational constraints (heap size, stack size,...).

2.2 Permutation

For practical reasons, permutation MAY use a range of M values, greater than the size N of the memory space (M>=N).

```
  | [0,M-1] -> [0,M-1]
P |
  | x       -> P(x)
```

For example, given a N memory space, and q a prime number so that q>N, and g a generator for the group Z/qZ, the P permutation (with M= q-1) can computed as:

```
      | [0,q-2] -> [0,q-2]
    P |
      | x       -> (g**(1+x) mod q)-1
```

2.3 bMAC computation

We consider a one way hash function H (such as SHA2 or SHA3) with three procedures, H.reset, H.update, and H.final.

Given a space memory N, a permutation P with M values, the bMAC, according to C like notation, is computed as:

```
H.reset() ;
for (i=0; i< M; i++)
{ if (P(i) < N)
       H.update(A[P[i]);
}
bMAC= H.final();
```

2.4 Unused memory

Unused memory MAY be filled by pseudo random values, before performing the bMAC computation.

2.5 Permutation entropy

A family of Pk permutations is a subset of M! permutations of M elements, which is computed according to dedicated algorithms.

We note #Pk the number of elements of a Pk family.

The entropy is the integer e, such as 2**e is closed to #Pk:

2**e <= #Pk < 2**(e+1)

The entropy of a family may be increased by the composition of Pk functions so that :

P(k1,k2,...,kn) = Pkn o ... o Pk2 o Pk1

2.6 Time-stamped bMAC

2.6.1 Rational

The main idea is to detect corrupted software that uses a code compression algorithm.

```
+-------------------------+    +-------------------------------+
|                         |    | +-+  Genuine Code Compressed  |
|                         |    +-|-|---------------------------+
|                         |    | | | Code Compression Algo.    |
|       Genuine Code      |    +-|-|---------------------------+
|                         |    | V +     Malicious bMAC + ^    |
|                         |    +-----------------------|-|----+
|                         |    |         Genuine Code   +-+    |
+-------------------------+    +-------------------------------+
|          bMAC           |    |             MALWARE           |
+-------------------------+    +-------------------------------+
```

The basic principle of the time stamped bMAC is that the code compression algorithm modifies the time needed for the bMAC computing. Furthermore we assume that the time required by the bMAC computing is dependent on the permutation.

Below is an illustration of C code that returns the content of a corrupted address:

```
if ((Adr >= Adr-Min) && (Adr <= Adr-Max))
v =decompress(Adr);
else
v= read(Adr);
```

Many computing cycles are added to the genuine code (read(Adr)) due to Program Counter jumps and execution of the decompression procedure.

2.6.2 Canonical time

We assume that the bMAC computing time (T) ranges between the values Tmin and Tmax

Tmin <= T <= Tmax

If the computing time is fixed (Tmin=Tmax) then the Canonical Time (cT) is the computing time T.

If Tmin#Tmax we define the following values:
Range = Tmax-Tmin+1
Delta = Tmin modulo Range

For a given computing time T, we define the canonical computing time cT as:
cT = (T-Delta)/Range

For every T value, cT has a fix value equal to the quotient of Tmin/Range.

The main interest of the canonical time is that it works as a secret value, deduced from the bMAC computing but not stored in the software memory image.

The time-stamped bMAC is computed from an exor operation between the bMAC and the canonical time:

Time-Stamped bMAC = bMAC exor cT

3. The Pq permutation family

We consider a N memory space, and q a prime number so that q>N.

Z/qZ is a monogenous group with n=phi(q-1) generators (g), phi being the Euler number. Generators (g) in Z/qZ can be used to build a permutation family Pq= = {Pg1, Pg2,.., Pgn}, so that:

```
                      | [1,q-1] -> [1,q-1]
                Pg(x) |
                      | x       -> g**x  mod q
```

Given a P permutation working in the [1,q-1] range (such as Pg), we use the P*(P) permutation in order to enforce compatibility with the memory space A(x) starting at the zero address :

```
                      | [0,q-2] -> [0,q-2]
                   P* |
                      | x       -> P*(x) = P(1+x)-1
```

3.1 How to find a generator

3.1.1 Method 1

Given x in [2, q-1],
If x**k mod q # 1 for all k in [1, q-2], then g is a generator.

3.1.2 Method 2

Factorize q-1 into primes: q-1 = q1**k1...qi**ki...qn**kn
Find n integers ai (a1...an) of order qi**ki, in Z/qZ (phi(qi**ki) elements)
The product of the n elements a1 x...x an, is a generator.

3.1.3 Method 3

q being a safe prime, q = 2*p+1 with p prime (p is the Sophie Germain prime),and q = 7 mod 8.

```
phi(q-1) = phi(2p) = p-1
1 generator of order 2, i.e. q-1
p-1 generators of order p, i.e. 2**k mod q with k in [1,p-1]
p-1 generators gk of order q-1.
```

The generators gk are the product of (q-1).2**k mod q, for k in [1,p-1]. In other words the generators gk are equal to q-(2**k mod q), for k in [1,p-1]

3.2 How to compute generators

Find a generator g.

There are phi(q-1) generators g**k, with k prime with q-1.

GCD(k,q-1)=1, GCD being the Greatest Common Divisor of two integers.

3.2.1 Example 1

```
q=11, phi(10)= 4
10= 2x5, phi(2)=1, phi(5)=4
prime numbers with 10= {1,3,7,9}

k     1 2 3 4  5 6 7 8 9 10
x**k  1
      2 4 8 5 10 9 7 3 6 1
      3 9 5 4 1
      4 5 9 3 1
      5 3 4 9 1
      6 3 7 9 10 5 8 4 2 1
      7 5 2 3 10 4 6 9 8 1
      8 9 6 4 10 3 2 5 7 1
      9 4 3 5 1
      10 1

10 has an order 2
3, 4, 5, 9 have order 5
10*3= 8, 4*10= 7, 5*10=6, 9*10=2 are generators

2 is a generator
2**3 = 8 is a generator
2**7 = 7 is a generator
2**9 = 6 is a generator
```

3.2.2 Example 2.

q= 23 = 2x11 + 1, p=11, q is a safe prime with q mod 8 =7

power of 2 mod 23 = {2**k, k in [1,10]}= {2,4,8,16,9,18,13,3,6,12}
10 generators gk of order 22 = {21,19,15,7,14,5,10,20,17,11}

3.2.3 Example 3.

Memory space N = 512B EEPROM + 8192B FLASH + 1024B SRAM = 9728B
Nearest prime number q = 9733
q-1 = 9732= 811 x 4 x 3
phi(9732) = 3240
2 is a generator
generators are numbers 2**k mod q, with k less than q-1, and k prime with 811, 4 and 3.

3.2.4 Example 4

Memory space N = 512B EEPROM + 8192B FLASH + 1024B SRAM = 9728B
Safe prime = 9887
4943 generators

3.2.5 Example 5

Memory space N = 4096B EEPROM + 262144B FLASH + 1024B SRAM= 274432
prime number q = 278543
q-1= 278542 = 2 x 11**2 x 1151
phi(278542) = 126500
5 is a generator
generators are numbers, 5**k mod q, with k less than q-1, prime with 2, 11, and 1151

3.2.6 Example 6

Memory space N = 4096B EEPROM + 262144B FLASH + 1024B SRAM= 274432
Safe prime = 275447
137723 generators

3.3 Shifted permutation

Given an integer s in the range [0, q-1], the shifted permutation P(g,s) is defined as

```
                     | [1,q-1] -> [1,q-1]
          P(g,s)(x) |
                     | x       -> s.g**x mod q
```

In other words P(g,s)(x) = s x Pg(x).

Because s can be written in the form s = g**d, s.g = g**(x+d), which leads to a right shift.
The number of shifted permutations is (q-1)*phi(q-1).
The benefit of shifted permutation is to increase, with a low cost

computation, the bMAC entropy.

3.4 Composition in Fq

Given a set of k ptuples {(g1,s1), (g2,s2),..., (gk,sk)} and associated shifted permutations P(gi,si), a permutation P(q,k) is computed according to the relation :

P(q,k) = P(gk,sk) o ... o P(g2,s2) o P(g1,s1)

3.5 Code example

The bMAC is computed with a permutation P= P(g2) o P(g1,s1)
The pseudo code is written in a C like way.
H is a SHA3-256 KECCAK hash function.

3.5.1 Example 1

In this example 32 bits integers are used.
The prime number q is 9733.
The address space is N= 9664.
For a 8 bits processor, 12MHz clock, the bMAC is computed in about 10s, i.e. 1ms per byte.

```
uint32-t x,y,bitn,v,gi[14];
uint32-t PRIME, g1=a-generator, s1=a-value, g2=a-generator;
bool tohash;

  PRIME =9733;
  H.reset();

  gi[0]= g2;
  for (int n=1;n<=13;n++)
  gi[n] = (gi[n-1] * gi[n-1]) % PRIME;

  x= s1;

  for(int i=1;i<PRIME;i++)
  { tohash = false
    x = (x*g1) % PRIME;
    bitn=x;
    y=1;
    for (int n=1;n<=14;n++)
    { if ( (bitn & 0x1) == 0x1)  y = (y*gi[n-1]) % PRIME;
      bitn = bitn >>1;
    }
    v = (y-1);
    // if address v exists, read the v address content A(v)
    // tohash=true ;
    if (tohash) H.update(A(v));
  }
```

```
H.dofinal();
```

3.5.2 Example 2

In this example 64 bits and 32 bits integers are used.
The prime number q is 278543.
The address space is N= 271360.
For a 8 bits processor, 16MHz clock, the bMAC is computed in about 320s, i.e. 1.1 ms per byte.

```
uint32-t bitn,v;
uint64-t x,y,gi[19];
uint32-t PRIME, g1=a-generator, s1=a-value, g2=a-generator;
bool tohash;

  PRIME = 278543;
  H.reset();

  gi[0]=(uint64-t)g2;
  for (n=1;n<=18;n++)
  { gi[n] =  gi[n-1] * gi[n-1];
    gi[n] =  gi[n] % PRIME;
  }

  x= s1;

  for(i=1;i<PRIME;i++)
  { tohash=false;
    x = x * (uint64-t)g1 ;
    x=  x  % PRIME ;
    bitn= (uint32-t) x;
    y=    (uint64-t) 1;

    for (n=1;n<=19;n++)
    { if ( (bitn & 0x1) == 0x1)
      { y = y * gi[n-1] ;
        y = y % PRIME;
      }
      bitn = bitn >>1;
    }

    v = (uint32-t)(y-1);
    // if address v exists, read the v address content A(v)
    // tohash=true ;
    if (tohash) H.update(A(v));
  }

  H.final();
```

4 bMAC protocol

A bMAC protocol involves a bMAC requester and a bMAC provider.

The requester sends to the bMAC provider the parameters needed for the P permutation.

The bMAC provider computes the bMAC according to the P permutation and returns the result.

If the bMAC provider has access to internet, the requester (typically a gateway) SHOULD control its internet access in order to avoid side channel attack.

5 IANA Considerations

TODO

6 Security Considerations

TODO

7 References

7.1 Normative References

7.2 Informative References

8 Authors' Addresses

Pascal Urien
Telecom Paris
19 Place Marguerite Perey
91120 Palaiseau
France

Phone: NA
Email: Pascal.Urien@telecom-paris.fr