CMAC: Cipher-based MAC

Tufa Alexandru Daniel 1628927

3 November 2017

1 Introduction

Cipher-based MAC, or CMAC, is a message authentication code that is based on a symmetric key block cipher. It is analogous to the CBC-MAC algorithm, but it addresses its vulnerability in regards to the length of the messages that must be fixed. This algorithm in reality is a variation of CBC-MAC that was proposed under the name XCBC. On this version two improvements were made,OMAC1[1] and OMAC2 and from OMAC1 we have derived CMAC that is the equivalent. This algorithm became a NIST recommendation in May 2005 while on the 6th of October 2016 was updated with the release of 800-38B.

2 Specification

2.1 Subkeys generation

Based on the paper published by Mihir Bellare, Joe Kilian and Philip Rogaway on September 8, 2000 called `The Security of the Cipher Block Chaining Message Authentication Code` we know that CBC-MAC is secure only if the algorithm uses fixed length messages. This issue is addressed in CMAC by generating 2 subkeys from the original secret key that will be used on the last message block. For this we will need our `b`-bit block cipher E, the secret key K and a constant C that depends on b,the number of bits in a block. More precisely C is the concatenation of the coefficients of the lexicographically first among all irreducible binary polynomial of degree b with the minimal number of nonzero terms. We calculate a value $K_0 = E_K(0)$ what will be used to generate the subkeys in the following way:

[1] OMAC stands for One-key MAC

- K_1: if the most significant bit of K_0 is 0 then $K_1 = K_0 \ll 1$, else $K_1 = (K_0 \ll 1) \oplus C$
- K_2: if the most significant bit of K_1 is 0 then $K_2 = K_1 \ll 1$, else $K_2 = (K_1 \ll 1) \oplus C$

2.2 Tag generation

The blocks are divided into b-bit blocks $m = m1 \parallel .. \parallel m_{n-1} \parallel m_n$ where the first n-1 blocks are complete. If the last block is complete then $m_n = K_1 \oplus m_n$ else $m_n = K_2 \oplus (m_n \oplus 10...0)$. We use as initialization vector all zeroes and we calculate each cipher block as the encryption of the XOR between the previous cipher block and the corresponding message block. The authentication tag will be the `t`, length requested for the MAC, most signficant bits of the last cipher block. Verification is identical to the CBC-MAC.

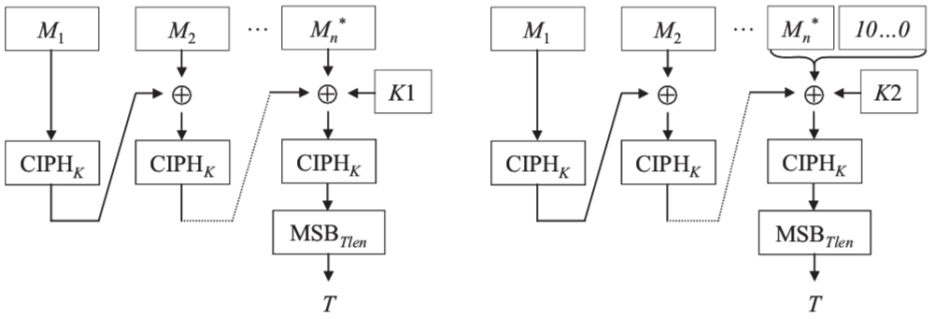

Figure 1: MAC generation

3 Attacks agains CMAC

As mentioned before this algorithm is not vulnerable when using messages with non fixed length because the last cipher block depends on one of the subkeys that depend on the secret key. If an attacker wants to do a brute force attack to find the authentican code the upper bound of the combinations to try is 2^t so larger values of t provide greater protection. A way to stop him from trying different MAC attemps is to limit the number of unsuccessful verification attemps for each key. Like other block cipher-based

MAC algorithms the attacker may want to find a collision between a new message and a valid authentication code and this collision is expected among a set of $2^{b/2}$ arbitray messages, property given by the `birthday paradox`, so if we use AES as the encryption algorithm we will have 2^{64} messages. A solution to this can be by limiting the message span for any CMAC key so in our case a default recommendation would be no more than 2^{48} messages for a key. An attacker may also try to resend the same message and authentication code pair at a later time and this can be solve easily by using some information that identifies the message, like a sequential message number or a timestamp, and adding it at the beginning of every message.

4 Analysis and comparison

CMAC is based on a symmetric key block cipher so it can be analyzed like a normal mode of operation. It's very similar to CBC so it will share most properties. Because in every encryption we keep track of the previous one, patterns will not be revealed. The process of generating the tag cannot be done in parallel or preprocessed, while the subkeys can be calculated in advance and stored with the secret key for repeated use. Even if it's more secure than CBC-MAC, it's still slow because it uses a block cipher while HMAC has the advantage of being fast thanks to the usage of a hashing function. This could not be the case if the platform of choice includes some hardware optimization for a specific block cipher, for example dedicated AES opcodes.

5 Library example

A useful library to use for generating a CMAC could be `The Bouncy Castle Crypto Package for Java` [1]. Given that we have the plain text and key as an array of byte, one can compute the authentication tag in the following way:

```
CipherParameters params = new KeyParameter(key);
AESFastEngine aesEngine = new AESFastEngine();
cipher = new CMac(aesEngine, macSizeInBits);
cipher.init(params);
for (byte b : plain) {
    cipher.update(b);
}
```

```
byte[] out = new byte[macSizeInBits / 8];
int i = cipher.doFinal(out, 0);
while (i < out.length) {
    i = cipher.doFinal(out, i);
}
```

`out` contains the desired authentication tag.

References

[1] The bouncy castle crypto package for java. https://github.com/bcgit/bc-java.