Universidade do Minho
Escola de Engenharia

Message Authentication Codes (MACs)

Tecnologia Criptográfica
Trabalho Prático IV

Trabalho realizado por:
Filipe Freitas (PG42828)
Maria Barbosa (PG42844)

Dezembro de 2020

Índice

Lista de Figuras

Listagens

Capítulo 1

Introdução

1.1 Contextualização

Na sequência da UC de Tecnologia Criptográfica foi proposto o presente trabalho prático cujo o enunciado se encontra dividido em duas partes. Na primeira, pretende-se implementar os três esquemas estudados nas aulas para aplicar ás primitivas que fornecem integridade com primitivas que fornecem confidencialidade. Esta implementação pode ser visualizada nos ficheiros *enc.py* e *dec.py* . Na segunda parte, pretende-se estudar duas falsificações do CBC-MAC, e implementar uma delas.

1.2 Estrutura do relatório

Este relatório divide-se em duas partes principais.

A primeira, correspondente a esta introdução, pretende contextualizar o trabalho e explicar a sua estrutura.

Na segunda parte apresentamos a descrição de como se pode produzir as falsificações pedidas, e apresentamos também a implementação de uma delas.

Capítulo 2

Falsificações do CBC-MAC

Vamos considerar o CBCMAC, utilizando a cifra AES, para mensagens de tamanho fixo, igual a dois blocos do AES. Consideremos a mensagem $m = m_1||m_2$, bem como a chave k.

Como descrito no enunciado, podemos enfraquecer este MAC de dois modos distintos:

1. Utilizando um IV aleatório, em vez de um valor fixo (tipicamente uma string de zeros);
2. Utilizando como *tag* todos os blocos do criptograma, em vez de apenas o último bloco.

Vamos descrever em seguida, como produzir uma falsificação para cada um deles.

2.1 Falsificação utilizando um IV aleatório

É possível, de forma muito simples, falsificar uma mensagem se o MAC utilizar um IV aleatório. Uma vez que, se o invasor for capaz de definir o IV a usar para verificação do MAC, este pode realizar modificações arbitrárias do primeiro bloco de dados sem invalidar o MAC.

Consideremos uma mensagem $m = m_1||m_2$, uma chave k, uma *tag* t correspondente ao último bloco de $e = E_k(m)$. Como m apenas tem dois blocos, então e_2 é esse último bloco.

Suponhamos agora que o atacante quer alterar o primeiro bloco m_1, substituindo-o por um bloco m'_1. É agora necessário falsificar um MAC válido para esta mensagem. Isto pode ser feito facilmente:

Seja $m' = m'_1||m_2$ e $e' = E_k(m')$. Queremos que $e' = e$; ou, mais especificamente, visto que m_2 não foi alterado, queremos que $e'_1 = e_1$.

Sabemos que $e'_1 = E_k(iv \oplus m'_1)$, e que o atacante tem controlo sobre o iv. Queremos encontrar um iv' tal que $e_1 = E_k(iv' \oplus m'_1)$. Isto acontece se, para cada bit alterado entre m_1 e m'_1, o bit de iv na mesma posição for também alterado. Temos, portanto, que:

$$iv' = (m'_1 \oplus m_1) \oplus iv$$

Demonstração. Queremos provar que $E_k(iv' \oplus m'_1) = E_k(iv \oplus m_1) = e_1$. Então:

$$
\begin{aligned}
& E_k(iv' \oplus m'_1) \\
&= E_k(((m'_1 \oplus m_1) \oplus iv) \oplus m'_1) && \text{(substituir } iv' \text{ pela definição acima dada)} \\
&= E_k(m'_1 \oplus m_1 \oplus iv \oplus m'_1) && \text{(associatividade de } \oplus) \\
&= E_k(m'_1 \oplus m'_1 \oplus iv \oplus m_1) && \text{(comutatividade de } \oplus) \\
&= E_k(0 \oplus iv \oplus m_1) && (x \oplus x = 0, \text{ para todo o } x) \\
&= E_k(iv \oplus m_1) && (\textit{pois } 0 \oplus x = x, \textit{ para todo o } x) \\
&= e_1 && \text{(pela definição de } e_1 \text{ acima)} \\
& && \text{c.q.d.}
\end{aligned}
$$

□

Assim, a mesma *tag* mantém-se válida apesar de uma mensagem diferente ter sido transmitida. O esquema deste ataque é, portanto, o seguinte:

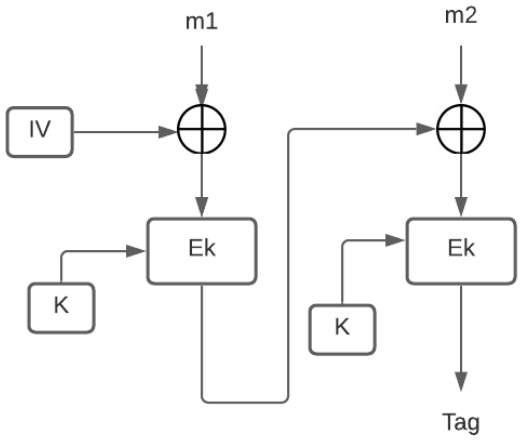

Figura 2.1: Esquema de falsificação CBC-MAC usando um IV aleatório

A função que faz a falsificação é, portanto, implementada do seguinte modo:

Ficheiro *forgery_stub.py*

```
56  def produce_forgery(msg, tag):
57
58      new_m1 = os.urandom(16) # Geramos um novo bloco inicial
59
60      iv = tag[1] # IV original
61      tag_original = tag[0] # Tag original
62
63      m1 = msg[:16] # Primeiro bloco original
```

```
64      other_blocks = msg[16:] # Restantes blocos da mensagem
65
66      # Gerar uma nova tag compatível
67      # Para tal, é preciso gerar um IV' = (new_m1 xor m1) xor iv, pois,
68      # para cada bit modificado em new_m1 em relação a m1, temos de fazer flip do
69      # mesmo bit no iv original
70      # (new_m1 xor m1) dá 1 em todas as posições modificadas, e portanto, ao fazer xor
71      # disso com o iv original, temos um novo iv' que corresponde ao flip dos bits nas
72      # mesmas posições onde new_m1 é diferente de m1
73      new_iv = byte_xor(byte_xor(new_m1, m1), iv)
74
75      # A nova mensagem é igual a new_m1 || m2 || m3 || ...
76      new_msg = new_m1 + other_blocks
77      new_tag = (tag_original, new_iv) # A tag é (tag_original, new_iv)
78
79      return (new_msg, new_tag)
```

2.2 Utilizando como *tag* todos os blocos do criptograma

A imagem 2.2 apresenta o mecanismo de falsificação do MAC que utiliza como *tag* todos os blocos do criptograma.

Vamos assumir que a Alice pretende enviar uma mensagem $m = m_1||m_2$, cuja *tag* é $t = t_1||t_2$, ao Bob. No entanto, uma entidade maliciosa, Eve, pode colocar-se à escuta e intersetar o envio de m e de t. Então, Eve pode construir a mensagem $m' = (t_1 \oplus m_2) \;||\; (t_2 \oplus m_1)$ e a *tag* $t' = t_2||t_1$. Isto é possível porque $t_1 = E_k(m_1)$ e $t_2 = E_k(t_1 \oplus m_1)$.

Apesar de $m \neq m'$ o verificador vai aceitar m' e t' tal como aceita m e t.

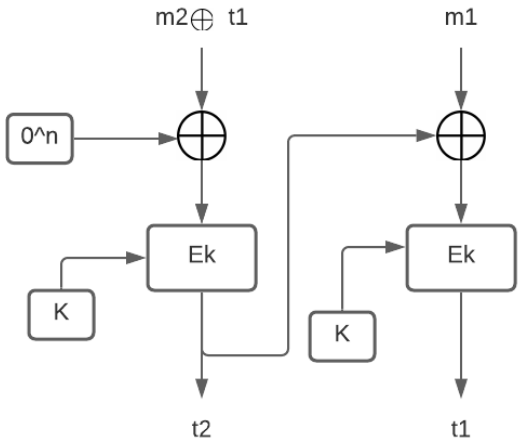

Figura 2.2: Esquema falsificação CBC MAC usando como *tag* todos os blocos do criptograma