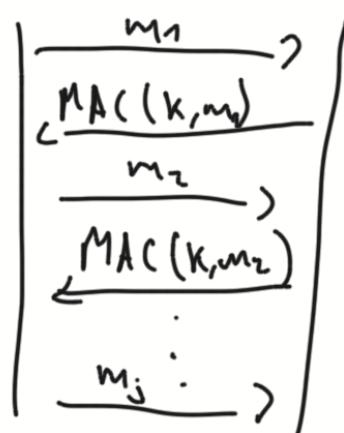


$$\text{MAC}(k, m) := \text{Enc}(k, m) = t$$

Adv      V



Adv gana si puede generar  
 $m' \notin \{m_1, \dots, m_j\}$  junto con  
 $\text{MAC}(k, m')$

Largo arbitrario? Sup. M es MAC de largo fijo.

$$\text{MAC}(k, m) := M(k, \bigoplus_{i=0}^{l-1} m_i) \text{ mala idea!!!}$$

$$\text{MAC}(k, m) := (M(k, m_0), \dots, M(k, m_{l-1}))$$

$$\text{MAC}(k, m) := (M(k, 0 || m_0), \dots, M(k, l-1 || m_{l-1}))$$

mala idea!!! Por qué?

Solución

Alice genera un r aleatorio y envía

$$\text{MAC}(k, m) = (r, M(k, r || l || i || m_i))_{i=0}^{l-1}$$

Esto funciona, pero no sabemos por qué.

Pero es horriblemente ineficiente.

Usaremos lo siguiente

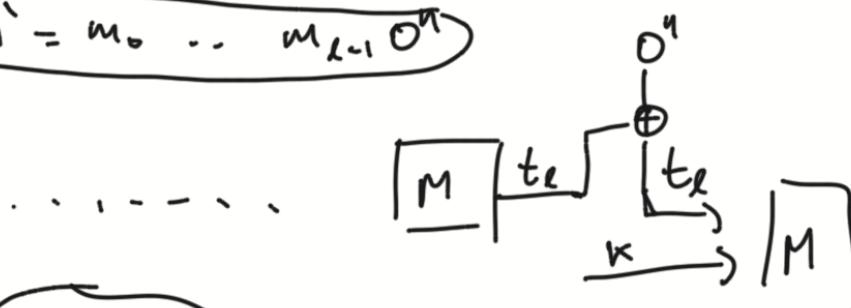


Esto viene de la construcción CBC  
(cipher block chaining)

Podemos atacar esta construcción

$$M = M_0 \dots M_{l-1}$$

$$m' = m_0 \dots m_{l-1} 0^n$$



$$m'' = t_l$$



¿Cómo gana el Adv.?

Envía  $m = m_0 \dots m_{l-1}$  y recibe  $t_m$

Envía  $m' = m_0 \dots m_{l-1} 0^n$  y recibe  $t_{m'}$

Genera  $m'' = \underline{t_m}$  junto con  $t_{m'}$

Mala idea!!!

¿Cómo lo arreglamos?

Por que usamos  $0^n$ ? Cambiarlo por un

rancor y compartimos  $t = (r, MAC)$

Cómo gana el Adv.:

Envía  $m = m_0 \dots m_{l-1}$  y recibe  $t_m = (r, t_e^m)$

Envía  $m' = m_0 \dots m_{l-1} 0^n$  y recibe  $t_{m'} = (r', t_{e+1}^{m'})$

Genera  $m'' =$  Mensaje arbitrario de 1 bloque

junto con  $t_{m''} = (m'' \oplus t_e^m, t_{e+1}^{m''})$



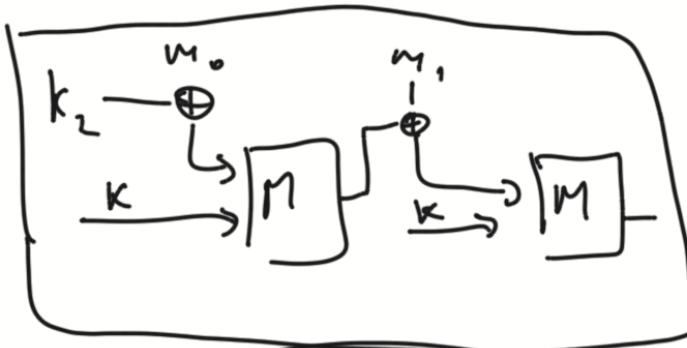
Pesar idea!!!

$$m'' \oplus r'' = t_e^m$$

Moraléja: Introducir más aleatorios es dar libertad al Adv.

- Y si: cambiamos  $K$  por  $\text{Enc}(k, l)$   
⇒ Distintas llaves para distintos largos.

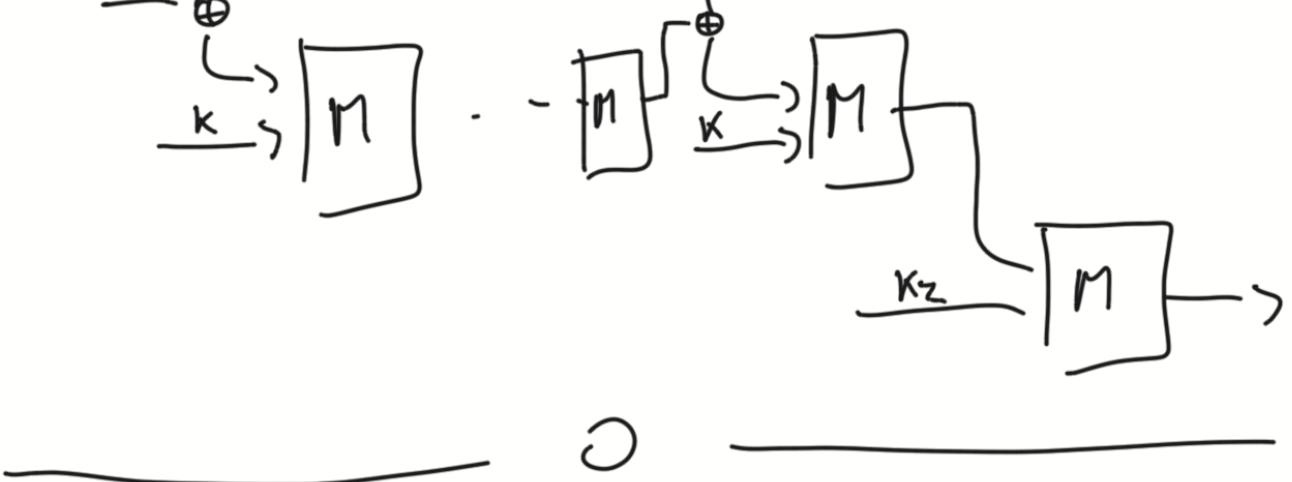
- Cambiar  $m$  por  $|l| || m$



Podría funcionar, no lo sabemos...  
para pensar...

- Compartir otra llave  $K_2$  y definir  
 $t = \text{Enc}(K_2, \text{MAC}(K, m))$

$0^n$   $m_1$   $m_{l-1}$



Y si usamos funciones de hash para construir MACs?

$$\text{MAC}(m) = h(m) \text{ pésimo}$$

$$\text{MAC}(x, m) = h(k || m) \\ h(m || k) \quad \text{para pensar...}$$



