Criptografía 2.0:

Continuación de 2022-08-10

Artículos a leer:

Juncotic criptografía y seguridad
Libro de criptografía y seguridad (Manuel J. Lucena López)
PBKDF2

Formas de romper la disponibilidad:

- ICMP Message flood.
- Haciendo que Eva envíe un reset a la conexión IP cuando Bob le quiere hablar a Alice.

¿Cómo lograr comunicación segura con cifrado simétrico y asimétrico?

Queremos lograr los 3 pilares: confidencialidad, autenticidad e integridad.

Criptografía simétrica:

Se necesita una clave tanto para cifrar como para descifrar. Y las hay de 2 tipos.

Criptografía simétrica de flujo/stream:

Se usa para cantidades de datos muy pequeñas de datos o cifras de bytes o words.
Ej: RC4 (antes se utilizaba en la web para lograr el https, pero es súper inseguro ahora).

Criptografía simétrica de bloques:

Este funciona sobre bloques de información de tamaño un fijo.
Tamaños: 128b, 56b, 384b, 512b.

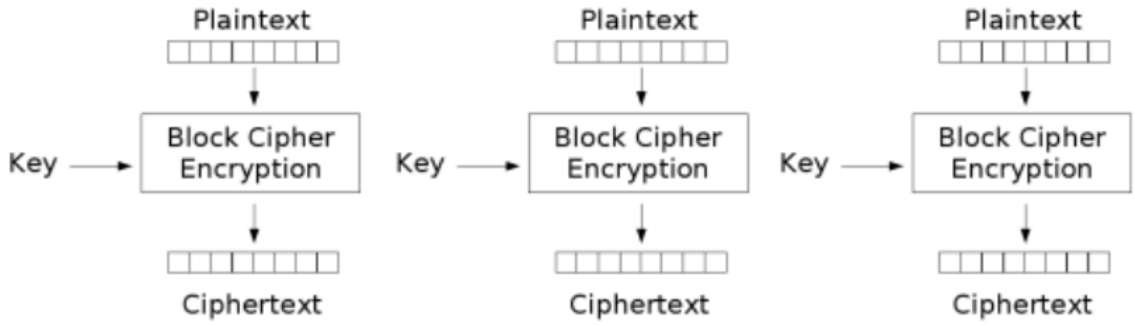
Ej: AES, 3DES.

Y tiene distintos **modos** en los que pueden funcionar.

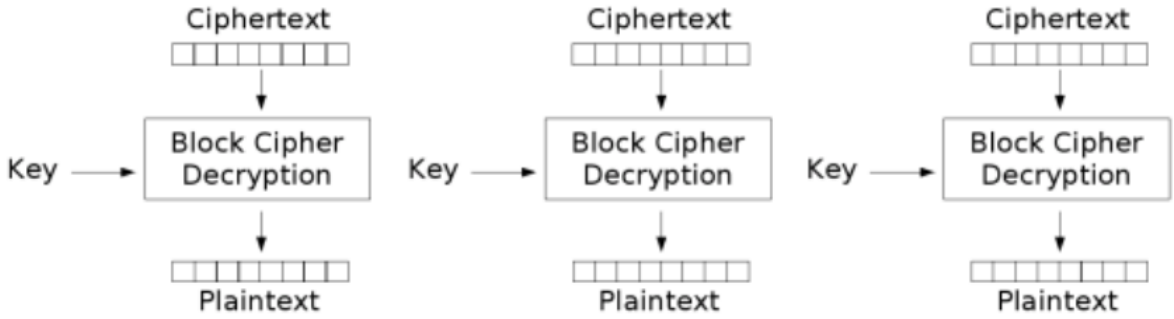
Modos:

Electronic code book (ECB):

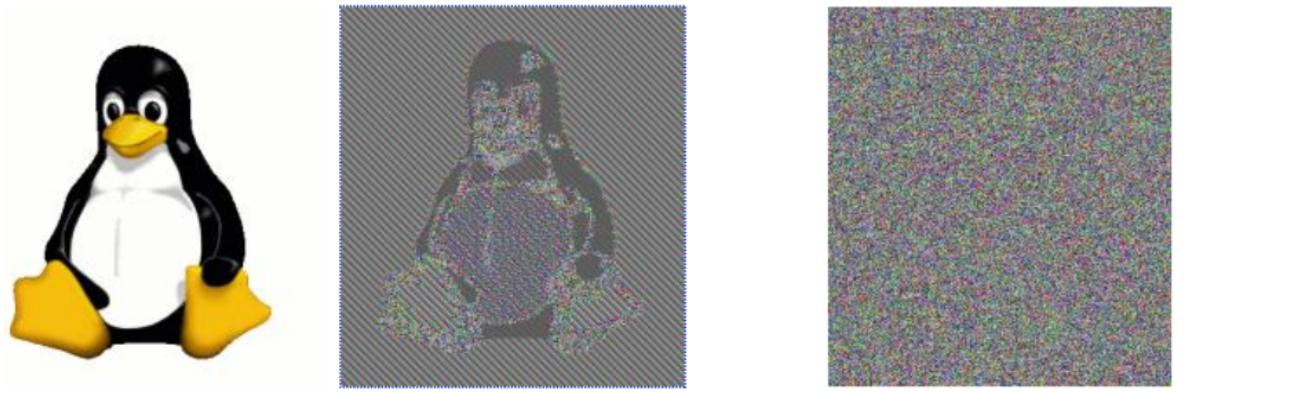
ECB - Electronic code book



Electronic Codebook (ECB) mode encryption



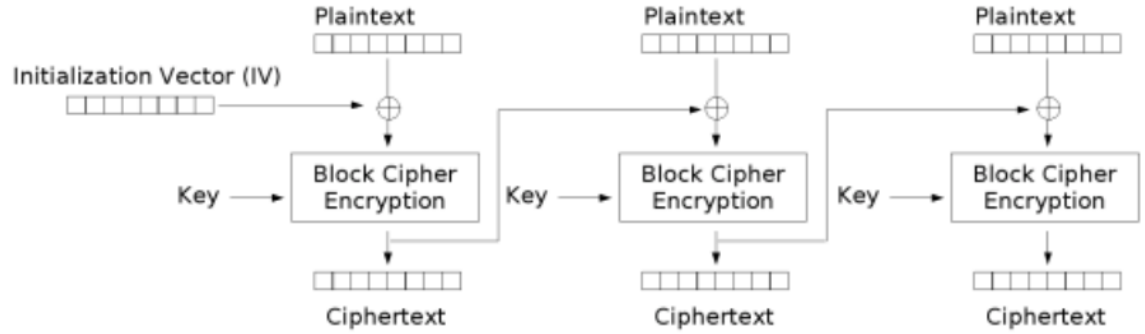
Lo bueno es que se puede paralelizar el cálculo porque los bloques son independientes entre sí, pero bloques iguales dan el mismo resultado, lo que puede llevar a dar información sobre el cifrado. Esto se vuelve evidente con las imágenes, pero funciona en todo archivo igual.



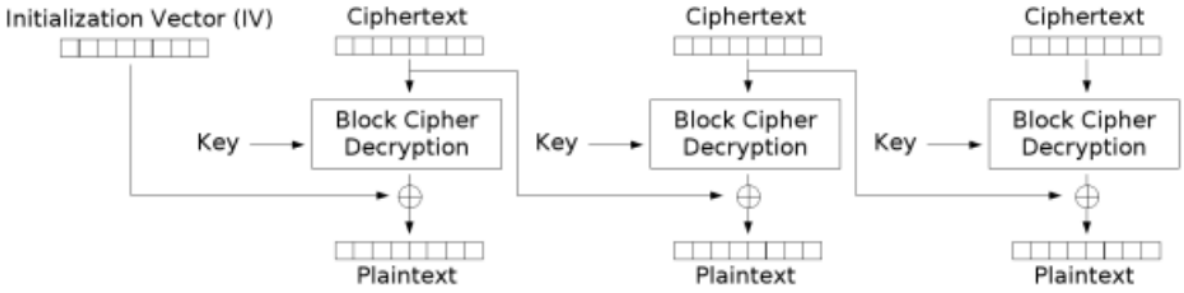
Original *Encrypted using ECB mode* *Other modes than ECB results in pseudo-randomness*

Cipher block chaining (CBC):

CBC - Cipher block chaining



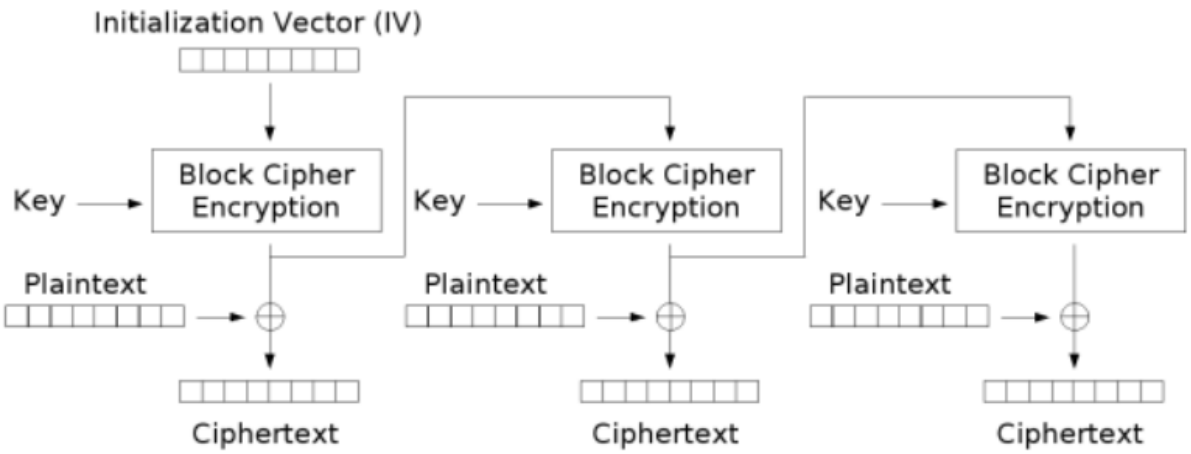
Cipher Block Chaining (CBC) mode encryption



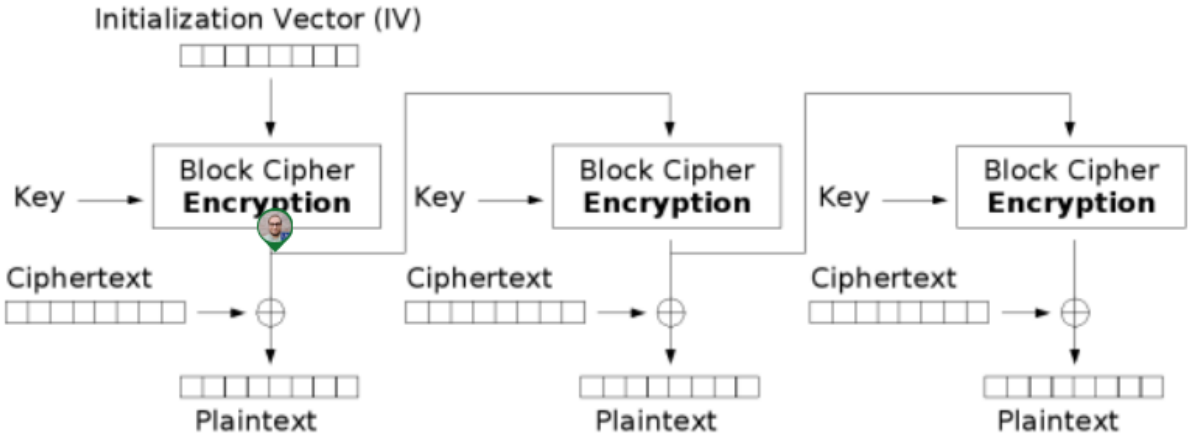
Cipher Block Chaining (CBC) mode decryption

Está piola, no genera algo que entregue información, pero es muy secuencial y no se puede paralelizar, por lo que resulta demasiado lento.

Output feedback:



Output Feedback (OFB) mode encryption

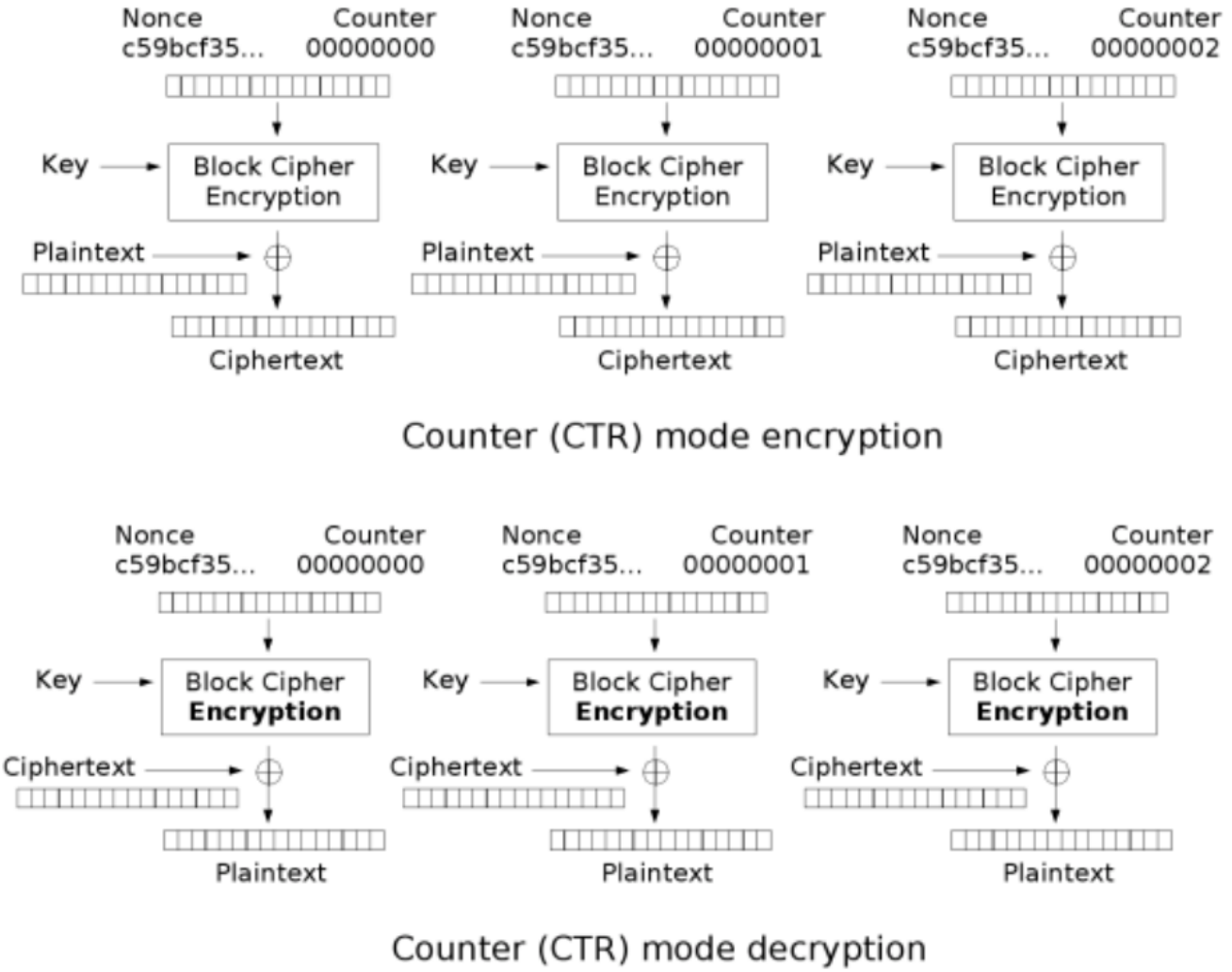


Output Feedback (OFB) mode decryption

Se pueden hacer 2 cosas que le da ventaja sobre el CBC:

- Se puede paralelizar el cifrado **i** y el cálculo del vector cifrado **i+1**.
- Se puede pre-calcular todos los bloques de cifrado y después solo ciframos o desciframos los textos (plano o cifrado). De esta forma es mucho más rápido, lo que está bueno y termina pareciéndose mucho a uno de stream.
  Tiene un poco de secuencialidad y una vez uno descifra 1 bloque, el resto se descifran al toque.

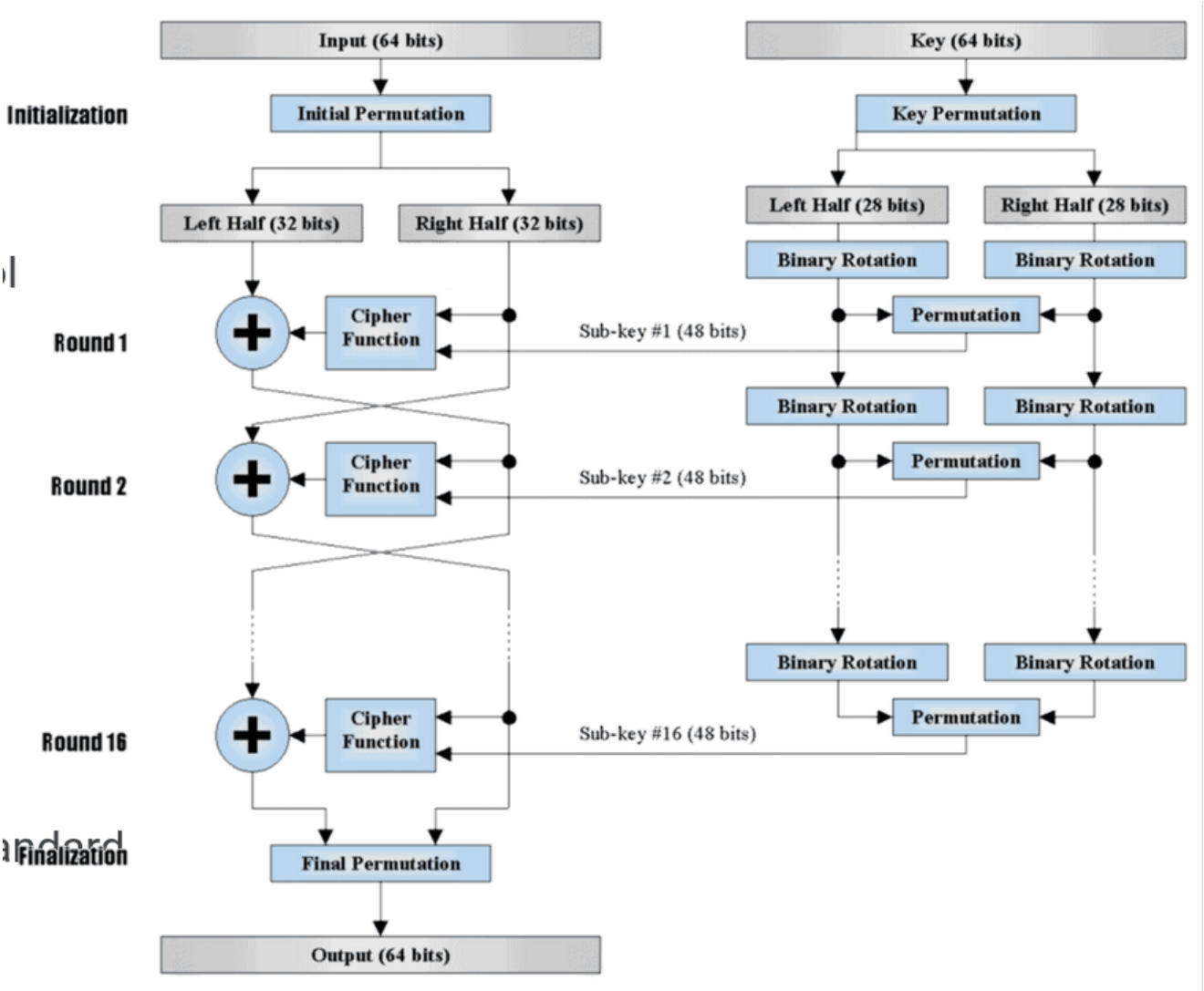
Counter mode encryption (CTR):



De estos es la opción más segura, varía el Vector, por lo que no importa si se descubre 1, se necesita saber que counter agregarle al vector para poder descifrar todos los demás.
Se puede paralelizar completamente.

DES > 3DES (Data Encryption Standard):

Utiliza una clave de 64b (56b de clave + 8b de Cyclic Redundancy Check (CRC)).
Es vulnerable a fuerza bruta, por lo que hoy en día se considera inseguro.



3DES:

3DES significa: DES en modo EDE (Encrypt - Decrypt - Encrypt).
Emplea 3 claves distintas, 1 para cada operación (E||D).

$$C = E_{k_3}(D_{k_2}(E_{k_1}(M)))$$

$$M = D_{k_1}(E_{k_2}(D_{k_3}(C)))$$

Se hace así para tener compatibilidad con los que no se bancan el 3DES, pero si DES. Se hace todo con la misma clave y es como si encriptaras 1 sola vez, solo que tardas un toque más.

Advanced Encryption Standard (AES):

Es la evolución más segura de la encriptación. Por la forma en que está hecha, con un largo decente (**256b** en adelante) es resistente a fuerza bruta icluso con computación cuántica.

Práctica:

Mans:

> man openssl
> opensl help
> man shadow
> man file
> man gpg

Confidencialidad:

Openssl:

> openssl enc -list

Prueba de encriptado:[1]

> openssl enc -aes-256-ctr -in Encrypt\ me.txt -out Decrypt\ me.enc

Más seguro porque genera a partir de la clave una encriptada *x* veces:

> openssl enc -aes-256-ctr -iter 10 -in Encrypt\ me.txt -out Decrypt\ me2.enc

Desencriptado:

> openssl enc[2] -d -aes-256-ctr -iter 10 -in Decrypt\ me2.enc -out Decrypted.txt

También debemos notar que los archivos encriptados están generados a partir de datos pseudo-random o *salted*. Utilizando un algoritmo como PBKDF2 para generar ese *vector* que vimos antes. Esto hace que el metodo sea aun mas seguro.

Test profe:

Para testear si aprendimos algo en la clase el profe nos hizo desencriptar un archivo `enigma.enc`, con el algoritmo `-chacha20`, 10 iteraciones y contraseña `compu2`.
Desencriptarlo fue facil, lo dificil fue darse cuenta de que el archivo era en realidad una imagen `.png`.

OpenPGP:

> gpg -a --symmetric Encrypt\ me.txt[3]

Como resultado da algo así:

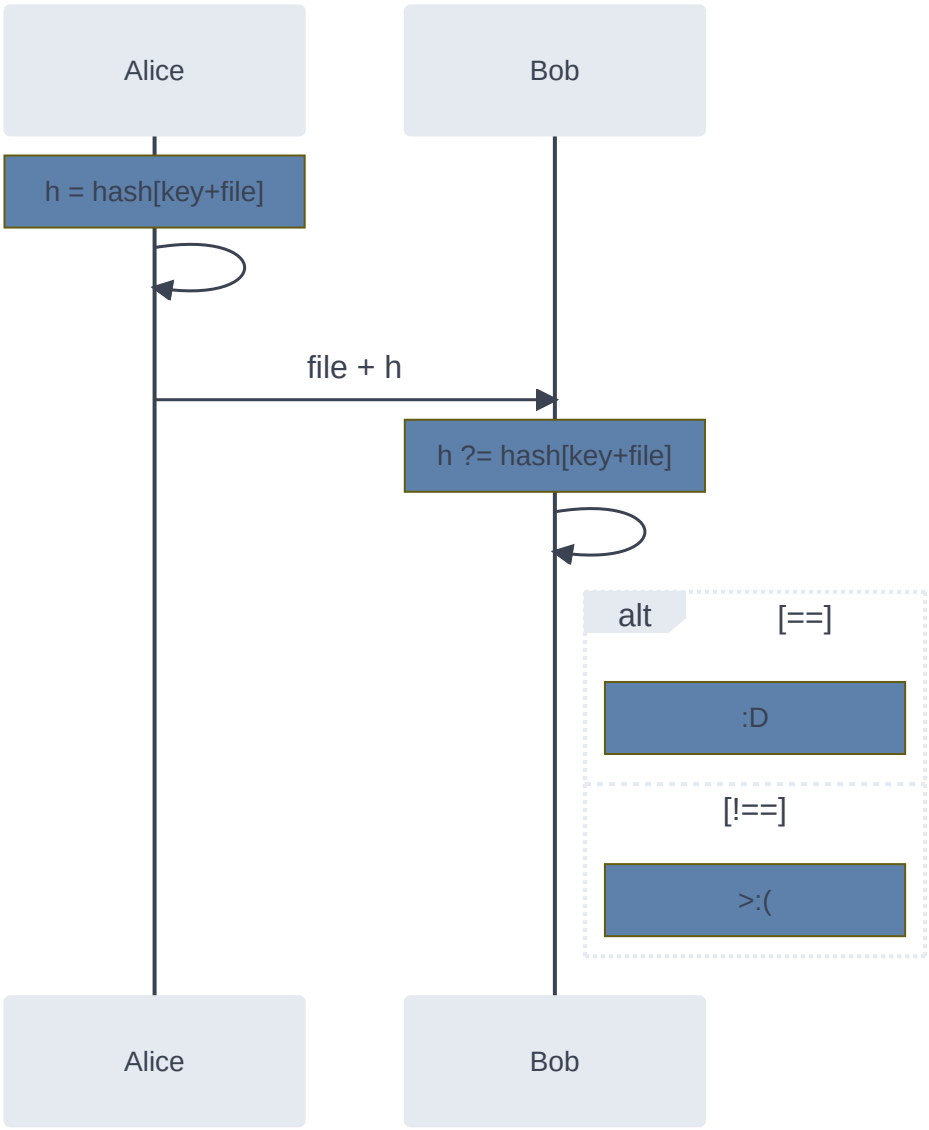
```
-----BEGIN PGP MESSAGE-----

jA0EBwMCb7qEDLFysrDx0l0BKsjRw63fvGHJaeF0QtDy4+Xobv3V2Ml6CjILzwk6
5QRgkNrVMz4v9j4K92LNYTsG5gea1Nw/V+ywbuTuxY98FXv7938MulDDh6hkSaoh
++rOfx70rScVYioZ8K8=
=j+8O
-----END PGP MESSAGE-----
```

Por dar un resultado en formato ASCII, está piola para pasar por mail, como se hace con el Base64...

- **La codificación** tiene función inversa y no necesita clave, solo necesitas saber como se hace. Ej: Base64 en correo para mandar binarios (como PDF) que no es carácter, porque el mail solo funciona con caracteres. Envío de mensajes, no secretos.

Integridad:



Con esto logramos integridad y autenticidad, porque solo Alice y Bob saben el password.
Esto se conoce como HMAC (Hash-based Message Authentication Code).

Todo junto (confidencialidad, integridad y autenticidad):

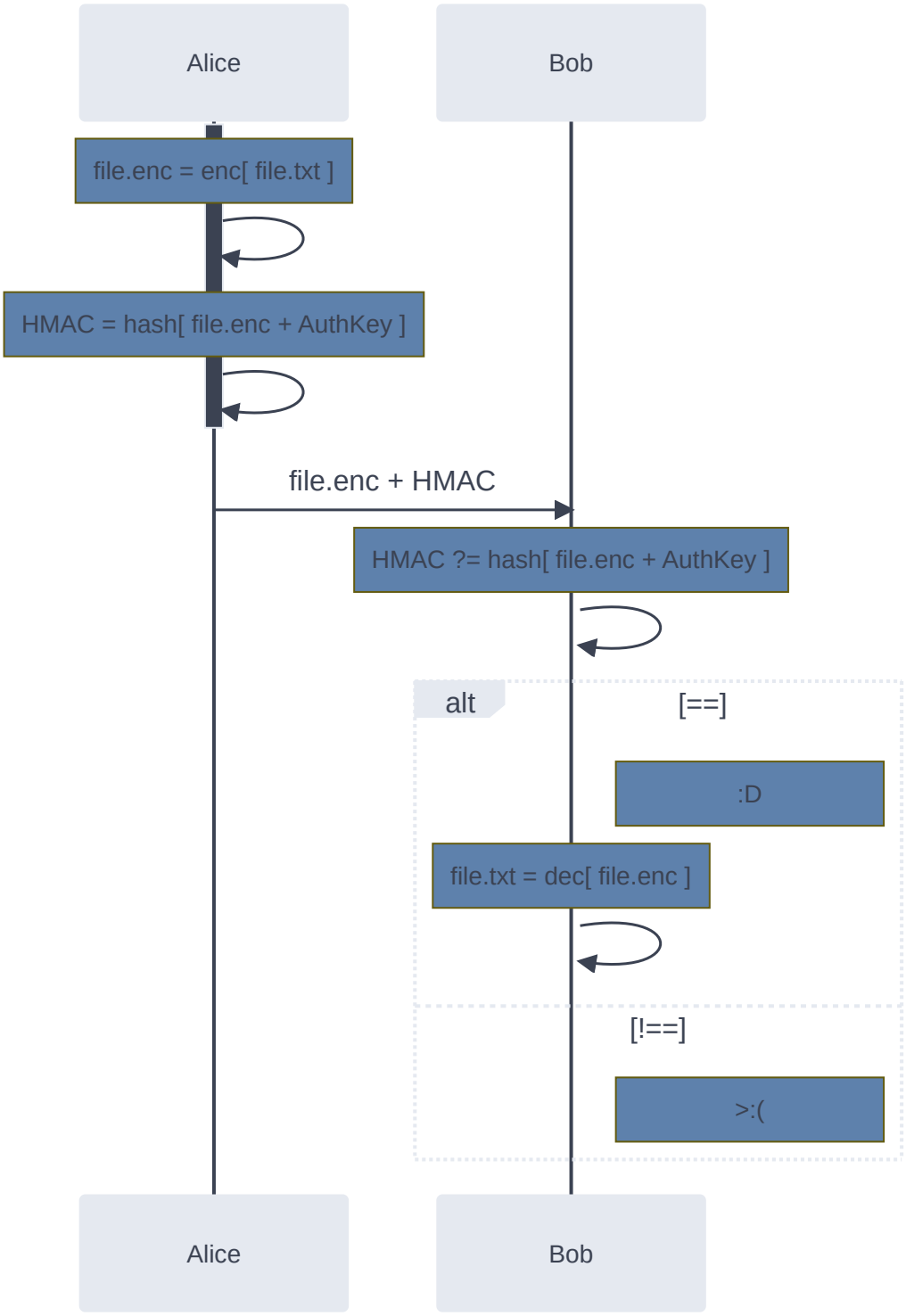
Confidencialidad -> Encriptar
Integridad y autenticidad -> HMAC

Necesitamos:

- File.

Necesitamos negociar:

- Authentication Key.
- Encryption Key.
- Encryption Algorithm.
- Hash Algorithm.



Próxima clase:

- Como se hace todo esto con asimétrico.
- Ver como funciona todo en un ejemplo de comunicaciones reales.
- Infraestructura PKI?

---

1. Contraseña: seguridad ↩
2. No hace falta usar el enc, podes solo mandarle el algoritmo y anda igual. ↩

3. La salida da un ASCII: 'Encrypt me.txt.asc'↩