Criptografía 2.0:

Continuación de 2022-08-10
Índice:

Artículos relacionados:

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.
La seguridad en la comunicación se puede lograr con **funciones hash** combinadas con **criptografía simétrica**, **criptografía asimétrica**, o con ***ambas***.

Funciones hash:

Las funciones hash, o resumen, permiten realizar cálculos para mapear un determinado dato de entrada en una cadena fija de bytes de salida (independientemente de la longitud del dato de entrada).
Ejemplos: MD5, la serie SHA y la serie SHA-3, etc.

Criptografía asimétrica:

En este caso los algoritmos utilizan dos claves, una para cifrar, y otra para descifrar. A estos algoritmos también se los denomina de clave pública, ya que a una de las claves se la denomina pública (porque esta disponible para todos), y a la otra privada (solo disponible para su dueño).
Ejemplos: RSA, DSA o variantes como ECDSA, etc.

Criptografía simétrica:

Si volvemos al criptosistema, los algoritmos de cifrado usados serán simétricos si se utiliza la misma clave para encriptar el contenido y para desencriptarlo. Los hay de dos tipos: Criptografía simétrica de flujo/stream y Criptografía simétrica de bloques.
Ejemplos: AES, 3DES, Blowfish, Twofish, etc.

Criptografía simétrica de flujo/stream:

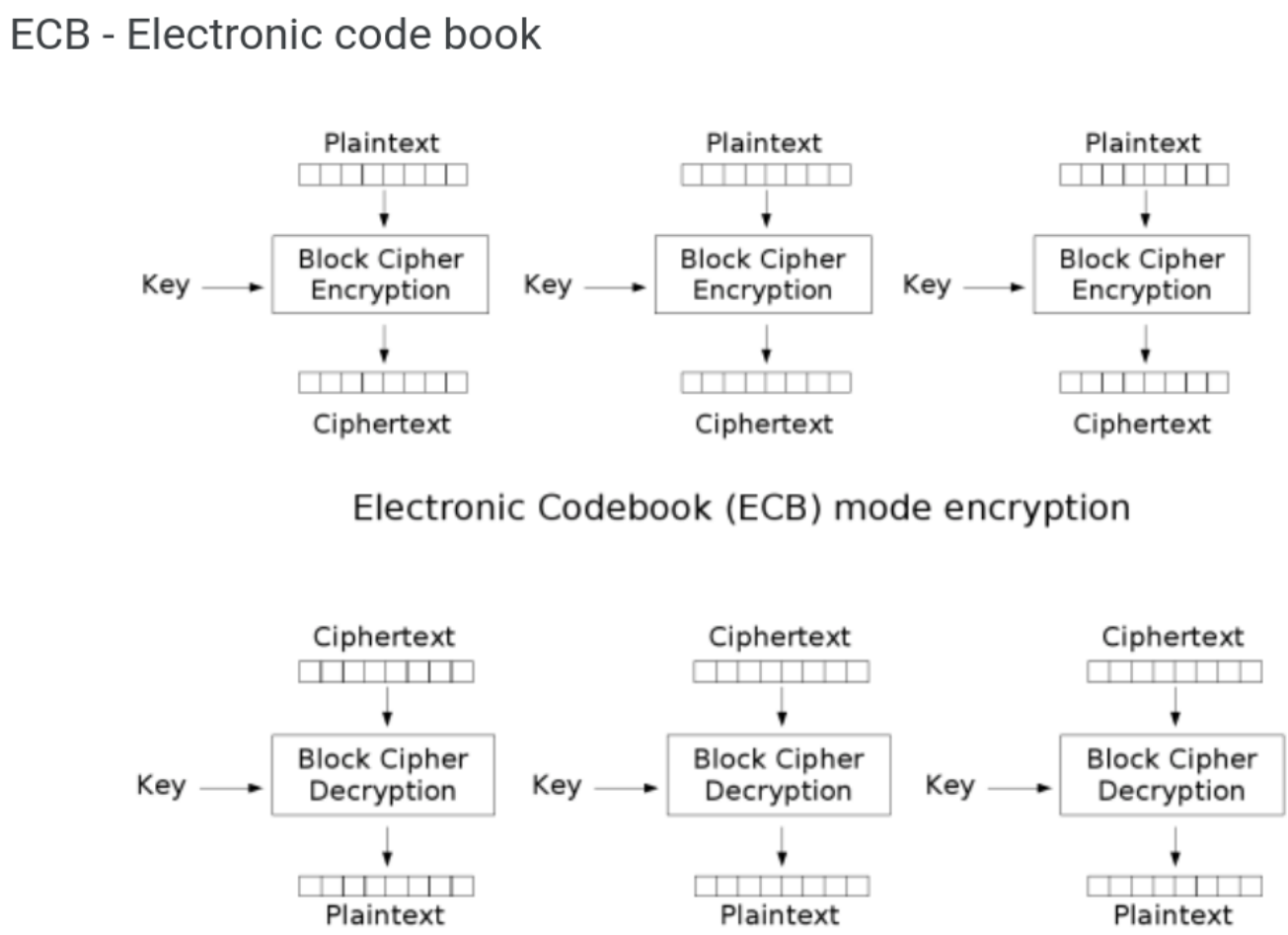
Se usa para cantidades de datos muy pequeñas de datos o cifras de bytes o words.
Ejemplo: RC4 (antes se utilizaba en la web para lograr el *https*, pero es superinseguro ahora).

Criptografía simétrica de bloques:

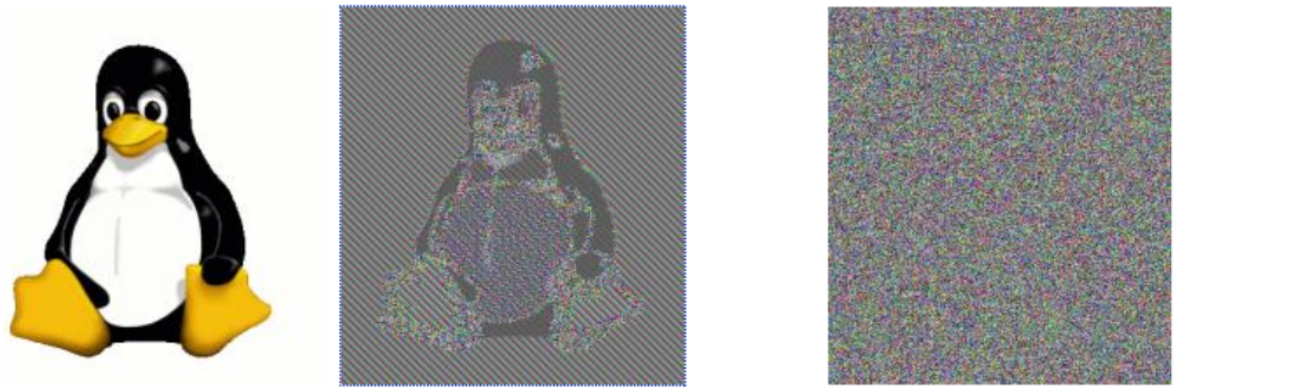
Este funciona sobre bloques de información de tamaño un fijo.
Tamaños: 128b, 56b, 384b, 512b.
Ejemplos: AES, 3DES.

Modos de funcionamiento:

Electronic code book (ECB):



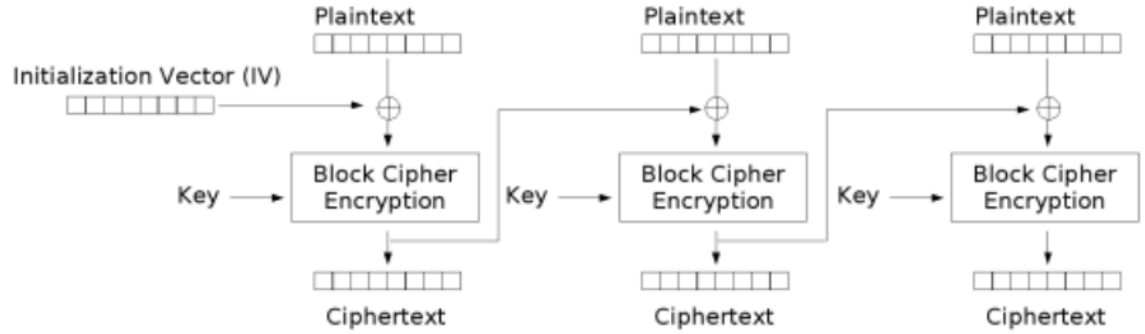
Lo bueno es que se puede paralelizar el cálculo porque los bloques son independientes entre sí, pero bloques iguales dan el mismo resultado, lo que puede llevar a dar información sobre el cifrado. Esto se vuelve evidente con las imágenes, pero funciona en todo archivo igual.



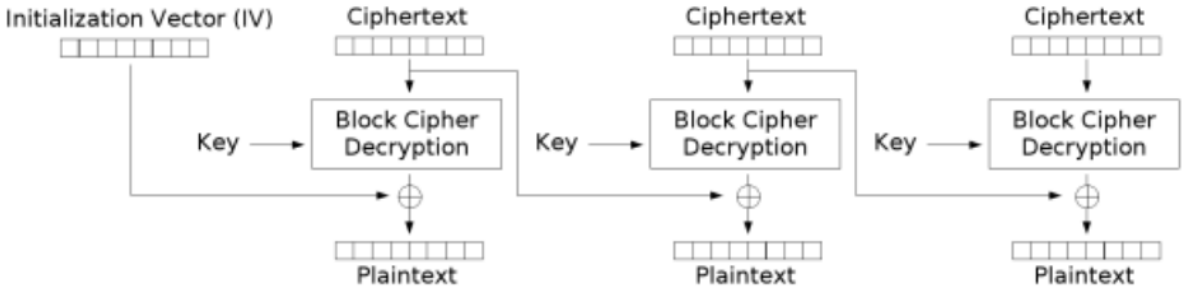
Original *Encrypted using ECB mode* *Other modes than ECB results in pseudo-randomness*

Cipher block chaining (CBC):

CBC - Cipher block chaining



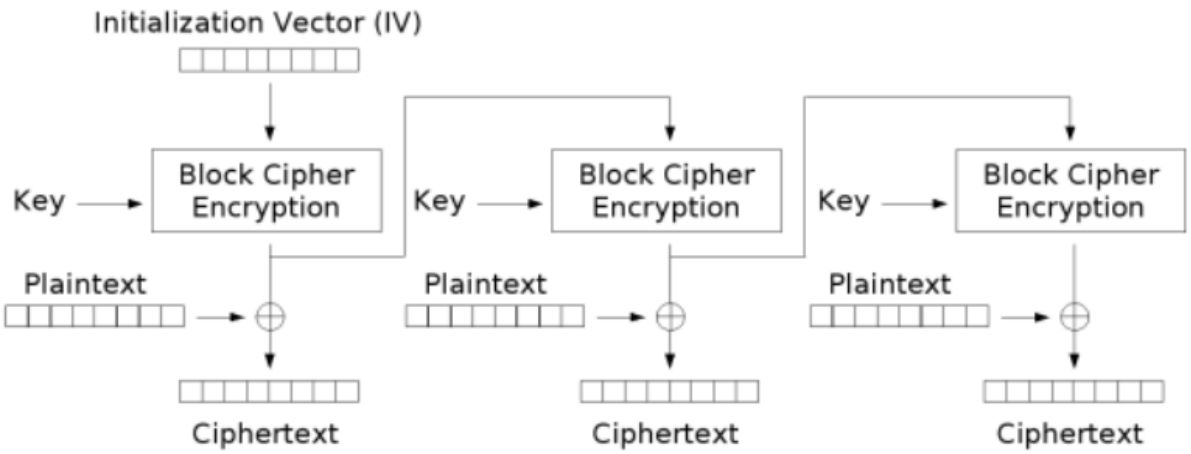
Cipher Block Chaining (CBC) mode encryption



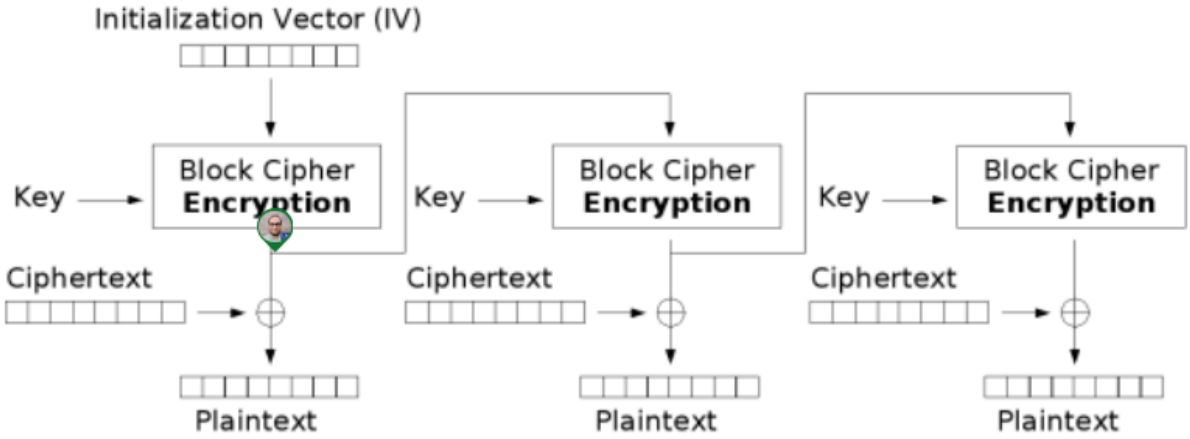
Cipher Block Chaining (CBC) mode decryption

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

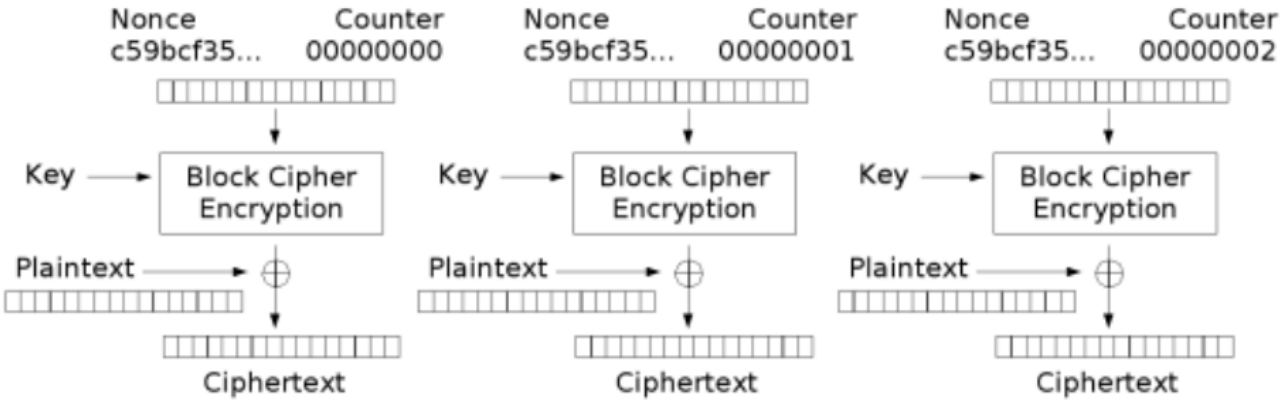


Output Feedback (OFB) mode decryption

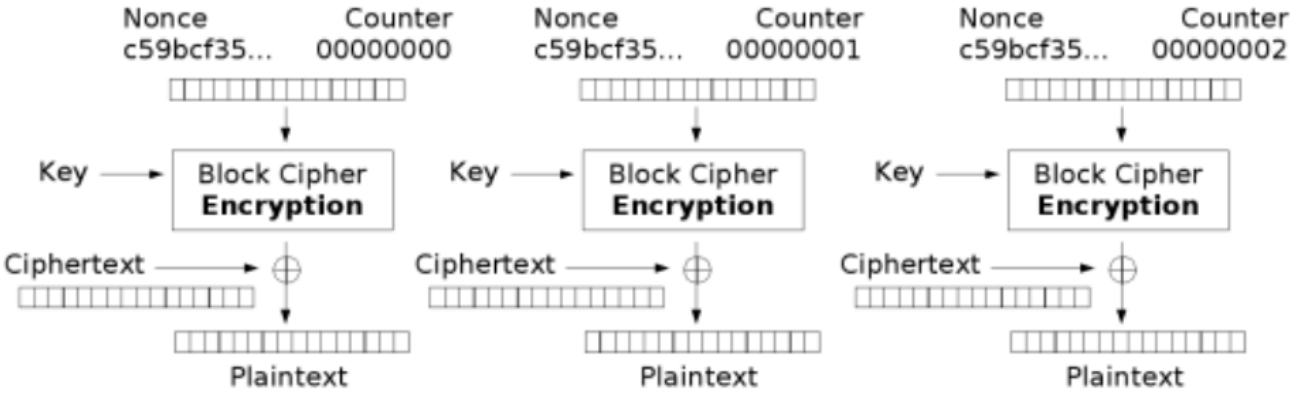
Se pueden hacer 2 cosas que le da ventaja sobre el CBC:

- 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):



Counter (CTR) mode encryption

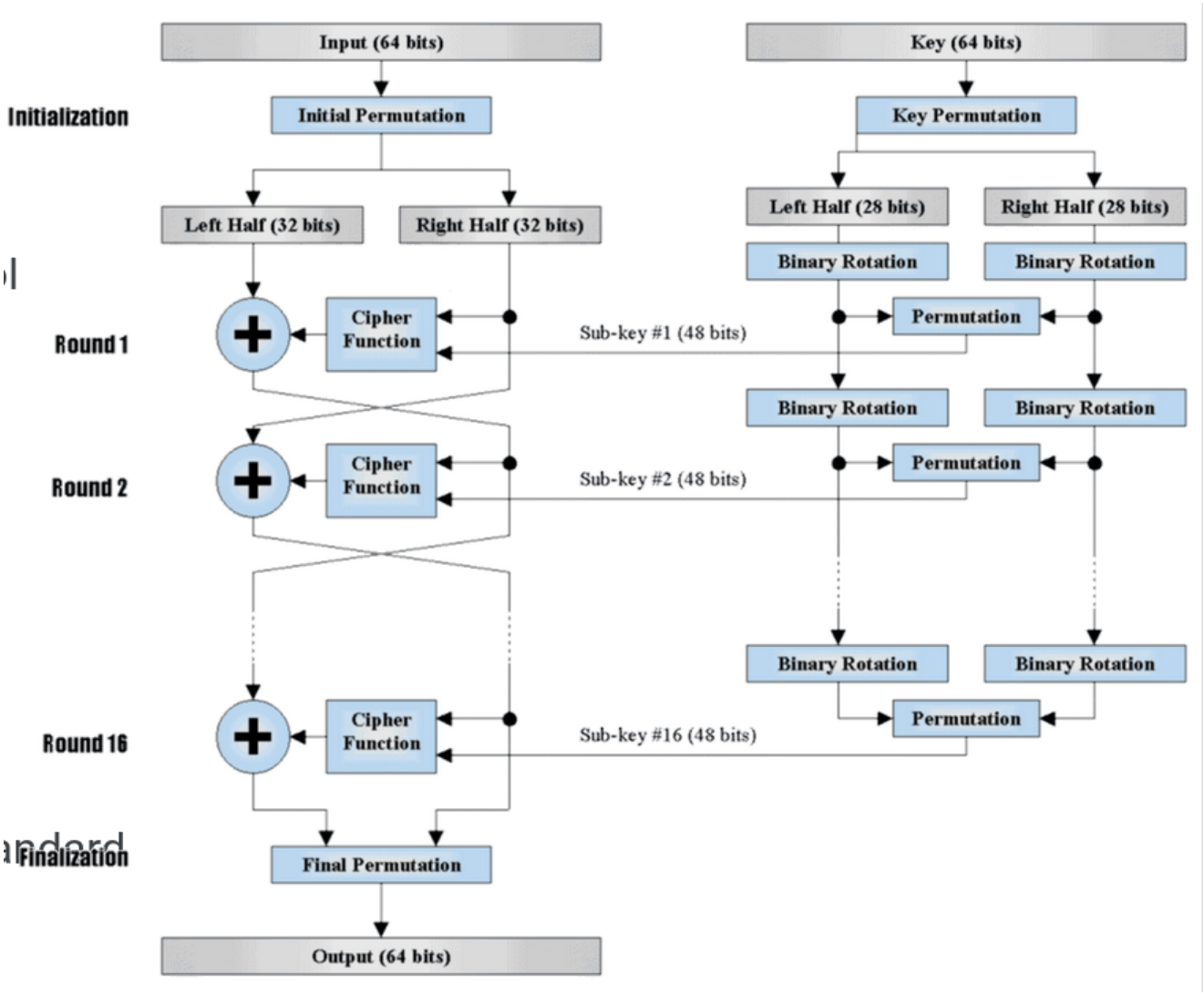


Counter (CTR) mode decryption

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 incluso con computación cuántica.

Criptografía hibrida y encapsulamiento de claves

Consideremos un cifrador asimétrico (Ej: RSA), este algoritmo permite cifrar mensajes de cierto tamaño, y es relativamente lento en comparación con cifradores simétricos

(Ej: AES). Entonces se debería usar los simétricos para mejorar el rendimiento de una comunicación. Por otro, RSA cifra mensajes de un tamaño igual al de la clave utilizada esto implica que si el mensaje es más pequeño que la clave, se deberá añadir un relleno, o padding, que a su vez añade algunos problemas de seguridad.
Una excelente forma de reducir estos problemas es emplear **criptografía híbrida**. Aquí se podría cifrar el mensaje utilizando una clave simétrica AES y luego cifrar dicha clave con RSA. (utilizar simétrico y asimétrico)
A esto se lo denomina mecanismo de encapsulación de clave, o **KEM (Key encapsulation mechanism)**, y es muy empleado, por ejemplo, en criptografía post-cuántica.

Criptografía post-cuántica

Ahora los esfuerzos se centran en lograr procesadores escalables, es decir, que pueda incrementarse la cantidad de bits cuánticos (*qubits*) para aumentar la capacidad de procesamiento. Para esto es necesario hacer una gestión eficiente del ruido, lo que actualmente es muy difícil de conseguir.
Igualmente, en teoría, estos procesadores, cuando alcancen una cantidad suficiente de *qubits* (miles), podrían romper **algoritmos asimétricos** de cifrado como RSA o DSA y sus variantes, e incluso protocolos de intercambio de claves como los basados en *Diffie-Hellman*.
Aquí surge un nuevo paradigma criptográfico: los **algoritmos post-cuánticos** (no confundir con los *cuánticos*). Estos algoritmos tienen fundamentos matemáticos no basados en principios de complejidad computacional, por lo que no pueden ser vulnerados (*en teoría*) por computadoras cuánticas.

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:[2]

```
openssl enc -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 método sea aún más 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 fácil, lo difícil fue darse cuenta de que el archivo era en realidad una imagen `.png`.

OpenPGP:

Encriptado usando gpg:[3]

```
gpg -a --symmetric Encrypt\ me.txt
```

Como resultado da algo así:

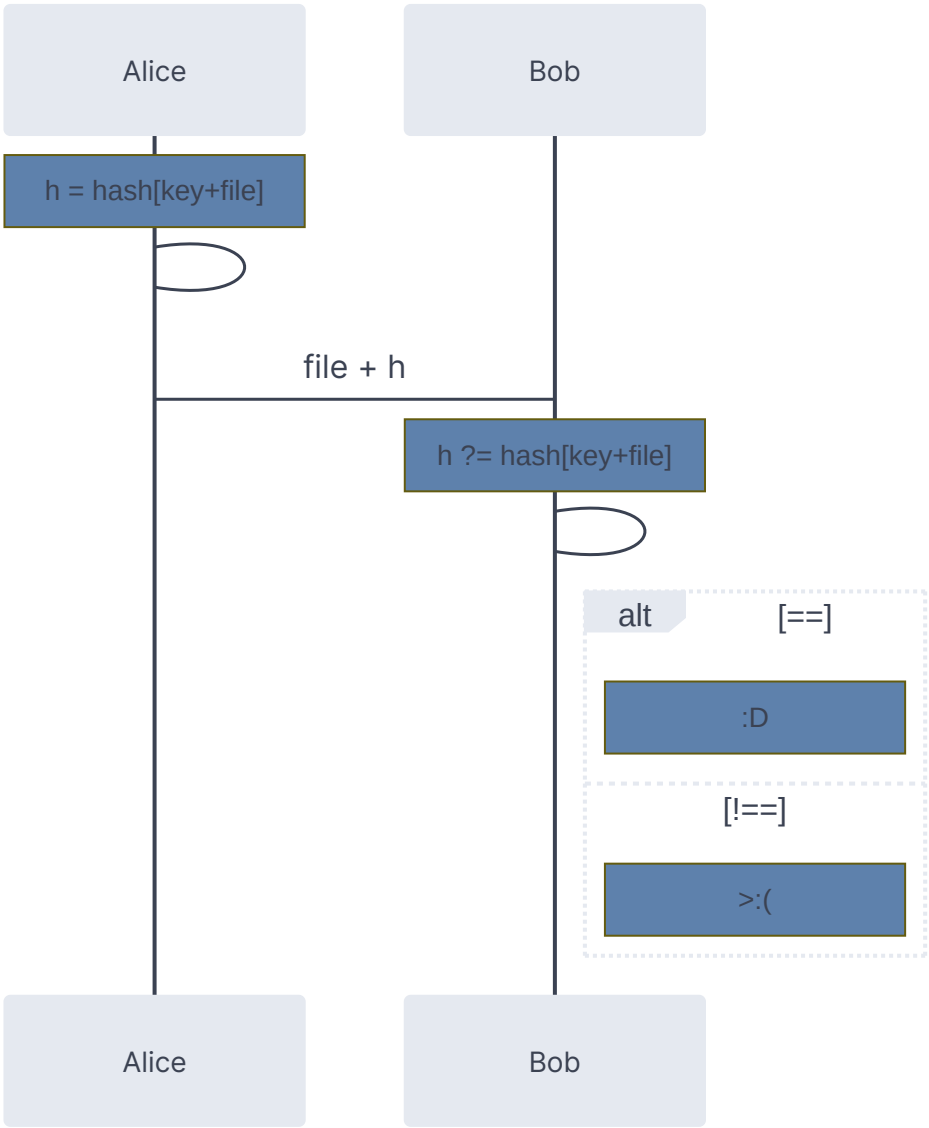
```
-----BEGIN PGP MESSAGE-----

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.

¿Cómo se logra la 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):

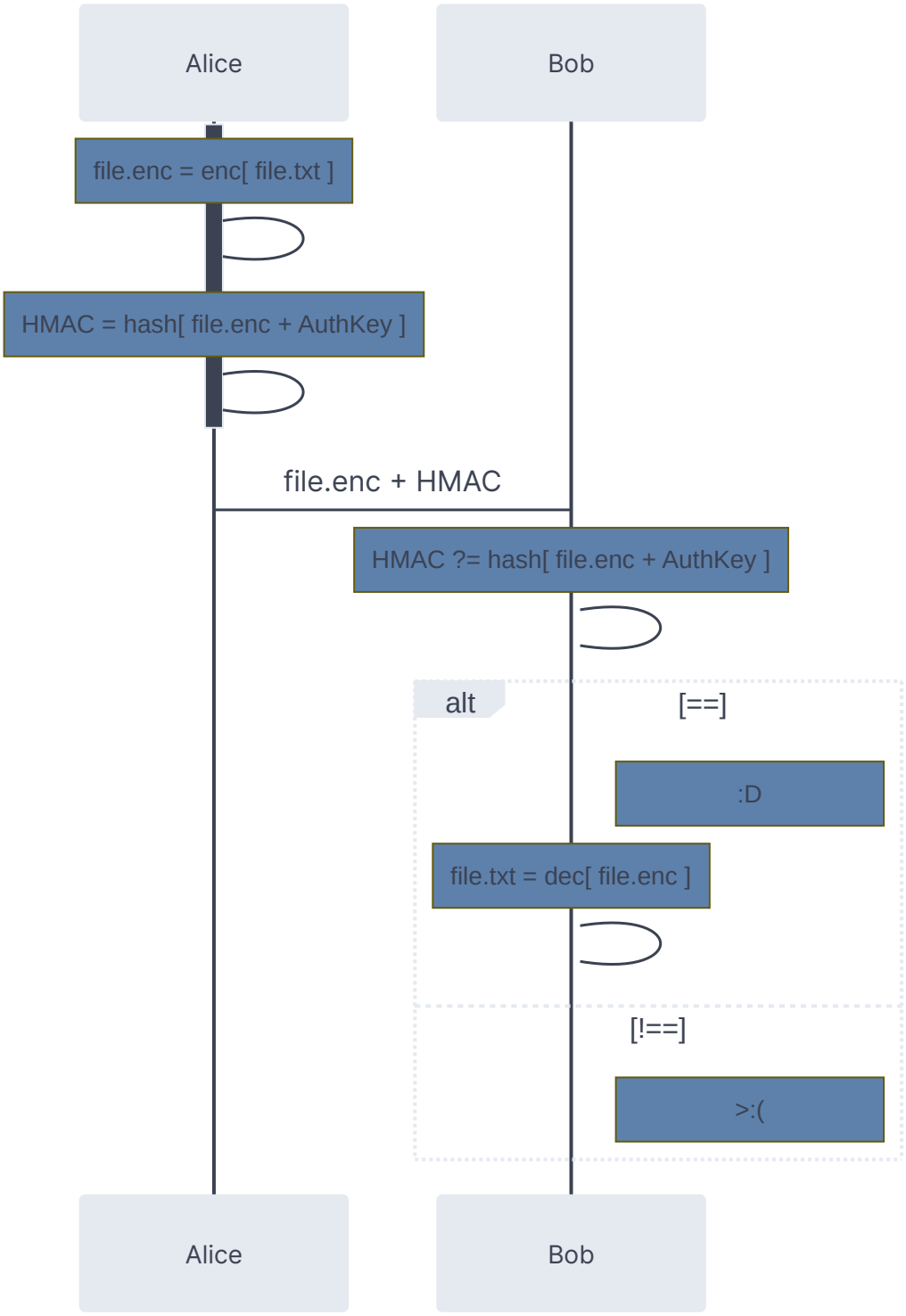
Confidencialidad → Encriptar
Integridad y autenticidad → HMAC

Necesitamos:

- File.

Necesitamos negociar:

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



Queda por ver:

- 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'↩