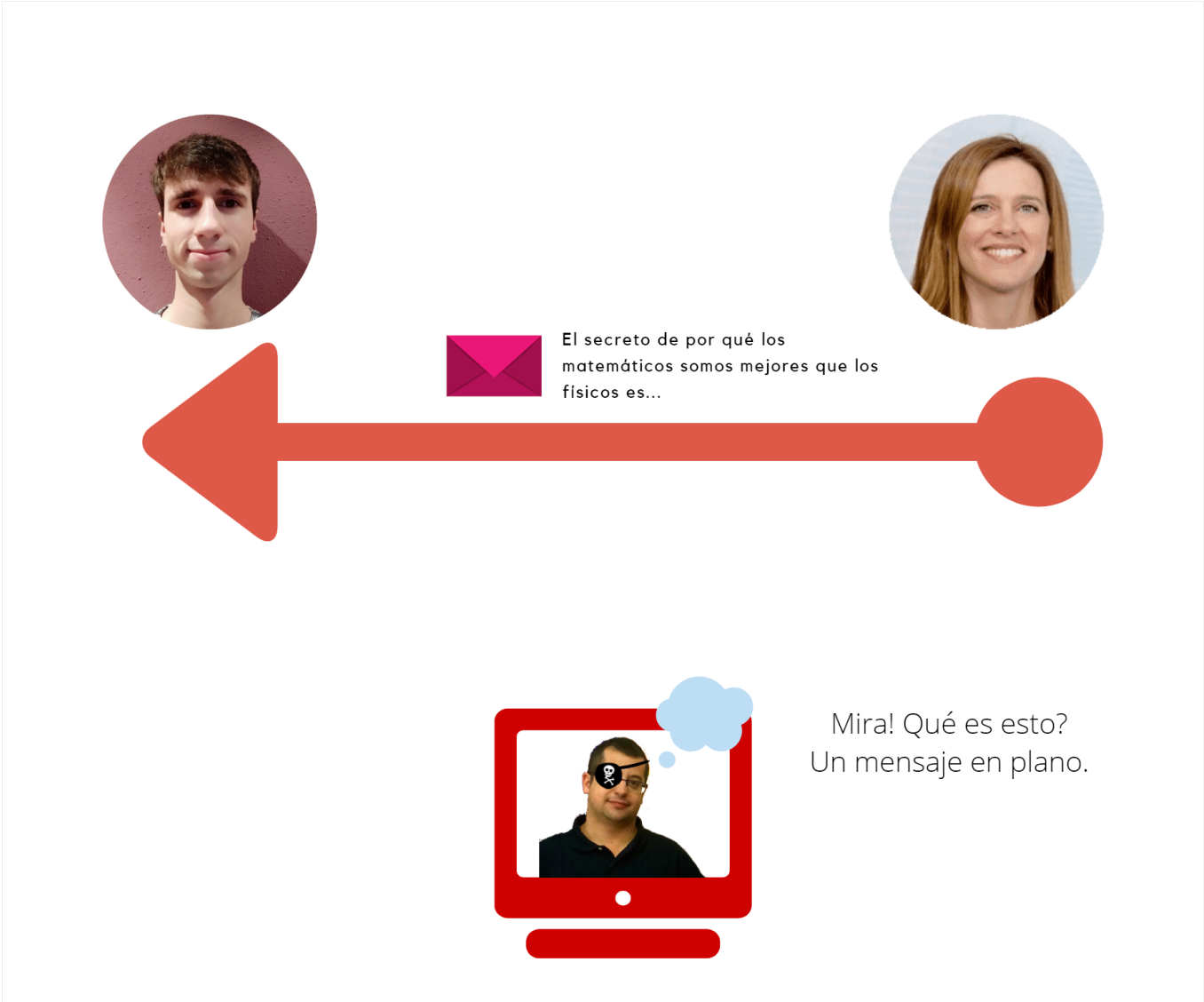
Diffie-Hellman

El intercambio de claves **Diffie – Hellman (DH)** es un método de intercambio seguro de claves criptográficas a través de un canal público y fue uno de los primeros protocolos de clave pública. Fue conceptualizado originalmente por Ralph Merkle (https://es.wikipedia.org/wiki/Ralph_Merkle) y el protocolo se llama así en honor a Whitfield Diffie (https://es.wikipedia.org/wiki/Whitfield_Diffie) y Martin Hellman (https://es.wikipedia.org/wiki/Martin_Hellman).

DH es uno de los primeros ejemplos prácticos de intercambio de claves públicas implementado en el campo de la criptografía. Hoy, DH se utiliza para todo tipo de aplicaciones como *Proton Mail* y *SSH*. Un software gratuito de cifrado de archivos que utiliza DH es *GPG*.

El problema en el cifrado de extremo a extremo

Pensemos en una situación súper simple. Imagina que Elena y yo decidimos intercambiar información. Ahora, digamos que un hacker llamado A. Corbi, está tratando de interceptar nuestro mensaje.



Una forma lógica de evitar que A. Corbi lea nuestro mensaje es mediante el cifrado. En el cifrado, se supone que incluso si se conoce el sistema de cifrado, el mensaje no se puede descifrar sin la clave de cifrado. Por lo tanto, siempre que Elena y yo usemos el mismo método de cifrado y tengamos la misma clave, ¡estamos listos para comenzar! Sin embargo, hay un problema ...

Para que yo pueda descifrar el mensaje cifrado de Elena, necesito que me envíe la clave a través de la red. El problema es que el Sr. Corbi está buscando la llave. Si obtiene acceso a la clave, ¡puede descifrar fácilmente todos nuestros mensajes! Este problema de intercambio de claves es abordado por el algoritmo Diffie-Hellman.

¿Cómo se resuelve el problema con Diffie-Hellman?

Diffie-Hellman funciona según el principio de no compartir completamente la clave de cifrado a través del medio. En cambio, cada parte consta de una clave pública (que todos pueden ver, incluido Mr. Corbi) y una clave privada (solo el usuario del punto final puede ver). Yo no tengo acceso a la clave privada de Elena. Elena tampoco tiene acceso a mi clave privada.

Si Elena y yo queremos intercambiar una clave de forma segura, primero tenemos que ponernos de acuerdo en un número **primo** `p = 761` y un `g = 6` que sea **raíz primitiva** respecto a p.

In [1]:

```
p: int = 761
g: int = 6
```

El protocolo es el siguiente

> Primero debo calcular un número entero grande x aleatoriamente (que no debo mostrar) y calcular:

$$(X = g^x) mod p$$

En criptología es importante poder calcular de manera eficiente b^n mod m, donde b, n y m son enteros grandes. No es práctico calcular primero b^n y posteriormente hallar el resto de dividirlo por m porque b^n puede ser un número excesivamente grande. En lugar de esto, podemos usar un algoritmo que emplea la expresión binaria del exponente n, es decir, $n = (a_{k-1}a_{k-2}...a_1a_0)$

Definimos el método **mod_exp** que nos devolvera el módulo

In [2]:

```
def mod_exp(b: int, e: int, p: int) -> int:
    x: int
    power: int
    x = 1
    power = b % p

    while e:

        if (e & 1):
            x = (x * power) % p

        e >>= 1
        power = (power * power) % p

    return x
```

Ahora podemos calcular el número X

In [3]:

```
import random
random.randint(1, 10**100)

# randomIker: int = random.randint(1, 10**100)
# randomElena: int = random.randint(1, 10**100)

# Entero aleatorio de Iker
# 997921920373160191192897584428170117330862397347306702632908401567426244950535502543593
2129284870203
randomIker: int = 997921920373160191192897584428170117330862397347306702632908401567426
244950535502543593212928487020 3
# Entero aleatorio de Elena
# 708838341297396596317233987453456904750597649468612469793577747095051899411958639990347
3171323486681
randomElena: int = 708838341297396596317233987453456904750597649468612469793577747095051
899411958639990347 3171323486681
```

In [4]:

```
xIker:int = mod_exp(g, randomIker, p)
```

$$(587 = 6^{997921920373160191192897584428170117330862397347306702632908401567426244950535502543593212928487020 3})m$$

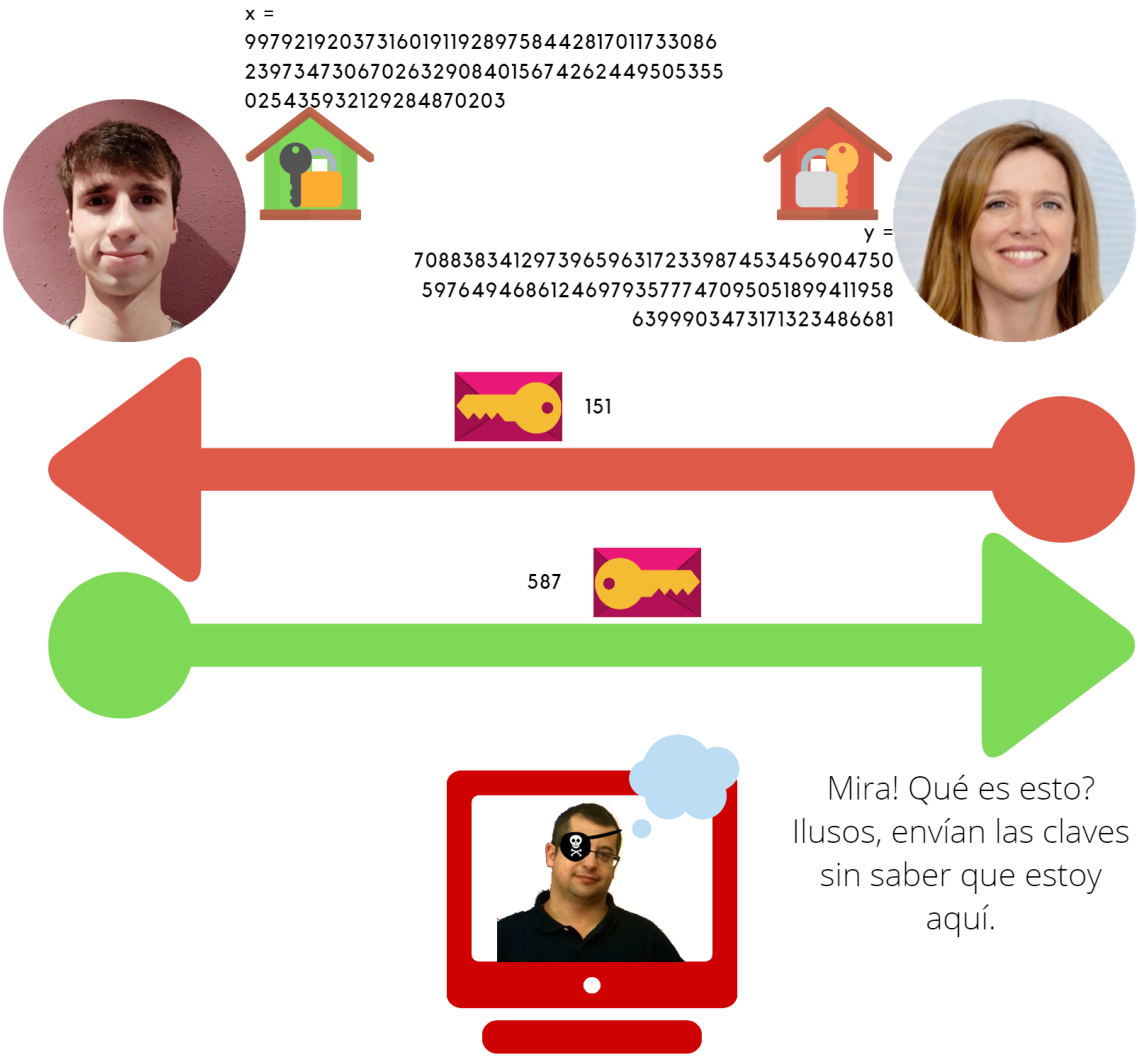
Repetimos el mismo paso para Elena:

In [5]:

```
yElena:int = mod_exp(g, randomElena, p)
```

$$(151 = 6^{708838341297396596317233987453456904750597649468612469793577747095051899411958639990347 3171323486681})m$$

Llega el momento de intercambiar las claves generadas. El Man In The Midle pensará que serán útiles para él.



Generar la llave completa

Es el momento de crear la llave que tendremos en común Elena y yo. Para Ello tenemos que calcular:

$$(K = Y^{x})modp \quad | \quad (K^{'} = X^{y})modp$$

In [6]:

```
kIker = mod_exp(yElena, randomIker, p)
kElena = mod_exp(xIker, randomElena, p)

if (kIker == kElena):
    print("Comparten la misma llave", kIker)
```

```
Comparten la misma llave 102
```

Resulta que se cumple

$$(g^{xy} \equiv X^y \equiv Y^x \equiv K \equiv K^{'})modp$$

Con lo que $(K \equiv K')modp =$ **102** es la clave de sesión que podemos usar Elena y yo para comunicarnos.

Como K nunca ha viajado por la red, A. Corbi no puede conocer la clave de sesión. Si quiere conocer K tiene que resolver el sistema de ecuaciones:

$$\begin{cases} (g^{xy}=X^y) \ mod \ p \\ (g^{xy}=Y^x) \ mod \ p \end{cases} \Longrightarrow \begin{cases} (\ xy \ log(g) = y \ log(X)\ ) \ mod \ p \\ (\ xy \ log(g) = x \ log(Y)\ ) \ mod \ p \end{cases}$$

Donde $x, y,$ son incógnitas y X, Y, g son los valores conocidos.

El problema que tiene aquí el Mr. Corbi es que calcular el logaritmo discreto en aritmética modular es muy complejo, y aunque $log(g)$ se puede conocer, $log(X)$ y $log(Y)$ dependen de las x, y que hemos elegido Elena y yo, que en este caso serian enteros aleatorios.

Aspectos para valorar

Para que el algoritmo sea seguro p debe ser un número muy grande. Sin embargo, la seguridad del algoritmo no depende del tamaño de g, con que g se elige como el número más pequeño que sea **raíz primitiva** respecto a p.

Tiempo de hablar

Una vez hemos establecido una conexión segura Elena puede comunicarme el importante mensaje que quería decirme.

In [7]:

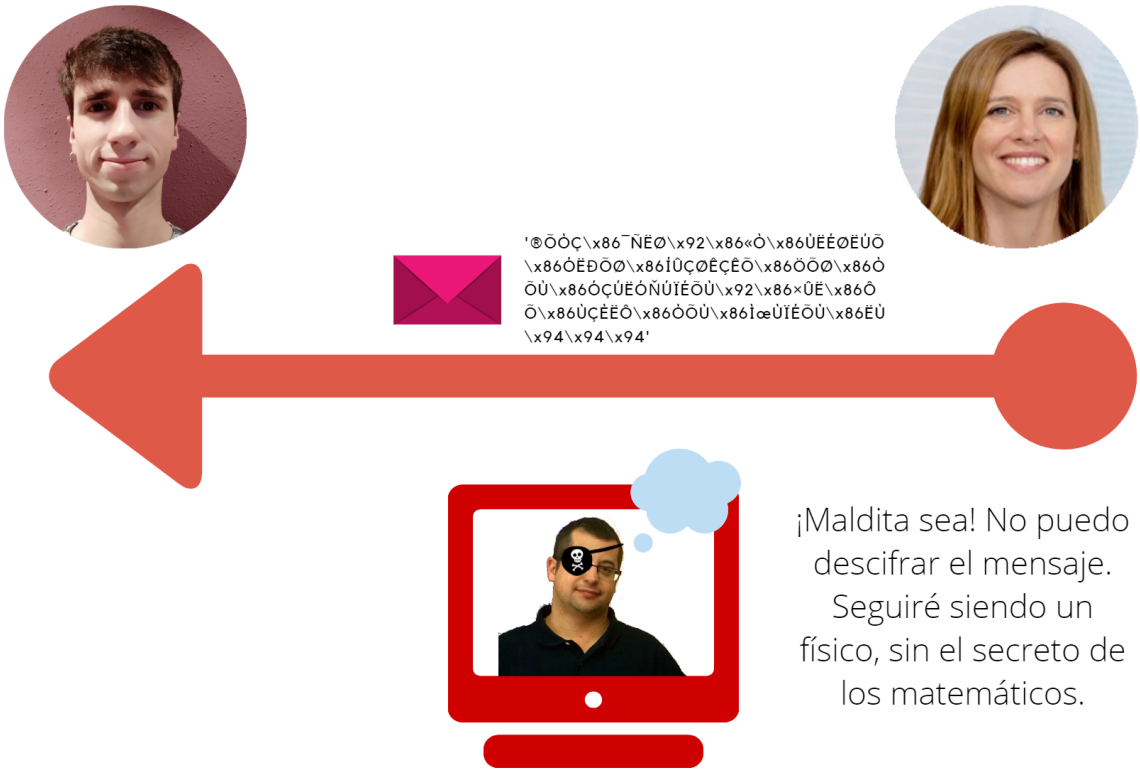
```
def cifrar_mensaje(mensaje: str, clave: int):
        mensaje_cifrado = ""
        key = clave
        for c in mensaje:
            mensaje_cifrado += chr(ord(c)+key)
        return mensaje_cifrado
```

In [8]:

```
cifrar_mensaje("Hola Iker, El secreto mejor guardado por los matemáticos, que no saben
 los físicos es...", 102)
```

Out[8]:

```
'®ÕÒÇ\x86¯ÑËØ\x92\x86«Ò\x86ÙËÉØËÚÕ\x86ÓËÐÕØ\x86ÍÛÇØÊÇÊÕ\x86ÖÕØ\x86ÒÕÙ\x86Ó
ÇÚËÓÑÚÏÉÕÙ\x92\x86×ÛË\x86ÔÕ\x86ÙÇÈËÔ\x86ÒÕÙ\x86ÌœÙÏÉÕÙ\x86ËÙ\x94\x94\x94'
```



Mientras que gracias a Diffie-Hellman yo sí tengo la llave maestra.

In [9]:

```
def descifrar_mensaje(mensaje: str, clave: int):
        mensaje_descifrado = ""
        key = clave
        for c in mensaje:
            mensaje_descifrado += chr(ord(c)-key)
        return mensaje_descifrado
```

In [10]:

```
descifrar_mensaje('®ÕÒÇ\x86¯ÑËØ\x92\x86«Ò\x86ÙËÉØËÚÕ\x86ÓËÐÕØ\x86ÍÛÇØÊÇÊÕ\x86ÖÕØ\x86ÒÕÙ
\x86ÓÇÚËÓÑÚÏÉÕÙ\x92\x86×ÛË\x86ÔÕ\x86ÙÇÈËÔ\x86ÒÕÙ\x86ÌœÙÏÉÕÙ\x86ËÙ\x94\x94\x94', 102)
```

Out[10]:

```
'Hola Iker, El secreto mejor guardado por los matemáticos, que no saben lo
s físicos es...'
```

Y así se consigue una comunicación segura.