Criptografía y Seguridad
Tarea 5 y 6

Ailyn Rebollar Pérez Ángela Janín Ángeles Martínez

Ejercicios

1. Acabas de intervenir la comunicación de un sistema que usa RSA.

   (a) Si detectas que se envió el mensaje cifrado $c = 10$ al usuario que tiene clave pública $e = 5, N = 35$, ¿cuál es el mensaje claro?

   i. Como $N = 35$, entonces tomemos un par de primos $p = 5$ y $q = 7$
   ii. Calculemos la función phi:
   $$phi = (7-1)(5-1) = 24$$
   iii. Con la llave privada y el criptotexto $c = 10$, aplicamos:
   $$mensaje = pow\ (10,\ privateKey)\ \%\ 35$$
   iv. El mensaje en claro es $mensaje = 5$

   (b) Si la clave pública de un usuario es $e = 31, N = 3599$, ¿cuál es la clave privada correspondiente?

   i. Como $N = 3599$, entonces tomemos un par de primos $p = 59$ y $q = 61$
   ii. Calculemos la función phi:
   $$phi = (59-1)(61-1) = 3480$$
   iii. Obtenemos la llave privada por medio de la fórmula:
   $$privateKey = modInverse(31, 3480)$$
   iv. Entonces obtenemos $privateKey = 3031$

2. Considera el esquema de ElGamal sobre $\mathbb{Z}_{71}^*$, con elemento generador $g = 7$.

   (a) Si Bartolo tiene clave pública $Y_B = 3$ y Alicia escoge el entero $k = 2$, ¿cuál es el mensaje cifrado de $M = 30$?

   Los valores a considerar son:
   $M = 30$ (mensaje a cifrar)
   $q = 71$
   $g = 7$
   $h = 3$ (la llave pública de Bartolo)

   Para cifrar, Alicia eligió el número al azar
   $k = 2$

   La tupla de cifrado se calculó mediante:
   $y1 = pow(g, k)\ \%\ q$
   $y2 = (pow(h, k) * msg)\ \%\ q$

   Por lo tanto, tenemos que: $C_k(M, k) = (49, 57)$

(b) Si ahora Alicia escoge otro valor para k de manera que el cifrado de $M = 30$ es la pareja $(59, C_2)$, ¿cuál es el valor de C_2?

Como tenemos la información dada en el punto anterior, ahora sabemos que para la primera entrada de la tupla tenemos: $C_k(M, k) = (59, C_2)$
Generamos un script de tal forma que obtuviera la k elegida por Alicia:

```
            for x in range(q):
                if (int(pow(g,x) % q) == 59):
                   k = x
                   print("Esta es la k", k)
                   break
```

Como resultado obtenemos $k = 3$

Realizamos la operación necesaria para obtener la segunda tupla:
$y2 = (pow(h, k) * msg) \% q$

Por lo tanto, tenemos: $C_k(M, k) = (59, 29)$

3. Cómo verificar si un entero positivo N es una potencia de un número. Sea $n = \lfloor \log N \rfloor + 1 =$ longitud en bits de N.

(a) Muestra que si $N = m^e$, para algunos enteros $m, e > 1$, entonces $e < n$.

Para mostrar esto vamos a tomarnos $N = m^e$.

$log(N) = log(m^e)$	aplicamos logaritmo a ambos lados de la igualdad.
$log(N) = elog(m)$	por propiedad de logaritmo
$\frac{log(N)}{log(m)} = e$	despejamos a e

Ahora consideremos las siguientes desigualdades:

$\frac{log(N)}{log(m)} < log(N)$ ... (1)
$\frac{log(N)}{log(m)} < \frac{log(N)}{log(m)} + 1$ ... (2)
$\frac{log(N)}{log(m)} + 1 < log(N) + 1$ ... (3)

Juntando las desigualdades obtenemos:

$\frac{log(N)}{log(m)} < \frac{log(N)}{log(m)} + 1 < log(N) + 1$

Sustituyendo a $\frac{log(N)}{log(m)}$ por e y a $log(N) + 1$ por n tenemos:
$e < e + 1 < n$

Y por transitividad se cumple que:

$$e < n$$
$$\therefore \text{ se cumple que } e < n$$

(b) Dados N y e, donde $2 \leq e \leq n + 1$, muestra cómo determinar si existe m tal que $N = m^e$. *Hint:* Usa búsqueda binaria en el rango $[2, N]$.

Para mostrar si existe o no una m que cumpla con $N = m^e$ se dará un algoritmo que corre en tiempo polinomial y para ello haremos uso del hint indicado.

1. Definimos una lista con los números del 2 a N.
2. Nos posicionamos a la mitad de la lista y supongamos que ese elemento se llama m, verificamos si $N == m^e$, si se cumple, entonces ya encontramos a la m que lo cumple y regresamos True, pero si no entonces debemos ver cuál de las siguientes condiciones se cumple:
   - Si N $< m^e$, entonces hacemos una búsqueda recursiva en la sublista de 2 a $\frac{N}{2}$.
   - Si N $> m^e$, entonces buscamos de forma recursiva en la sublista de $\frac{N}{2} + 1$ a N.
3. En caso de que terminemos de recorrer la lista y no se haya encontrado a m, regresamos False.

Observemos que como se realiza una búsqueda binaria, entonces en cada paso vamos descartando la mitad de la lista, haciendo un número logaritmico de pasos. Por lo que toma O(log(N)) pero si queremos dejar la complejidad en términos de n, hay que despejar a n de la siguiente manera:

$n = log(N) + 1$
$2^n = 2^{log(N)+1}$
$2^n = 2^{log(N)}2^1$
$2^n = 2N$
$\frac{2^n}{2} = N$
$2^{n-1} = N$

Entonces sustituyendo tenemos $O(log(2^{n-1})) = O(n-1)$ donde entra en $O(n)$.

(c) Dado N, muestra cómo determinar si N es una potencia de un entero.

La complejidad de los algoritmos debe ser polinomial en n. Justifica en ambos casos.

Para mostrar que sí podemos saber que dado un número N es una potencia de un entero vamos a hacer uso de los incisos anteriores. Pero para ejecutar el algoritmo presentado en el inciso b) tenemos que encontrar a la n primero, así que el algoritmo completo será el siquiente:

1. Vamos a tener una lista de números de 2 a n+1 donde recordemos que $n = \lfloor \log N \rfloor + 1$ que serán los posibles exponentes para la potencia. Así que vamos a recorrer la lista, donde cada que lleguemos a un número con el algoritmo visto en el inciso (b) vamos a revisar si podemos obtener a la m para tener a N expresada como potencia, pero hay dos posibles casos que son:
   - El algoritmo de busqueda regresa True, si esto sucede podemos regresar True, pues ya encontramos que si es una potencia de un numero entero.
   - El algoritmo de busqueda binaria regresa False, si sucede esto, entonces nos pasamos al siguiente candidato de exponente.
2. Si llegamos al final de la lista y no encontramos ninguna exponente e ni m que lograrán dar a N, entonces regresamos False.

Ahora analicemos la complejidad, ésta cumple con ser $O(n^2)$ pues estamos recorriendo una lista de un número lineal que es de 2 a $n+1$ y cada que la recorremos ejecutamos un algoritmo que toma $O(n)$, entonces estamos haciendo un algoritmo con una cantidad lineal de pasos que toma cada uno tiempo lineal dando como resultado un algoritmo de $O(n^2)$

4. Intercambio de clave.

(a) Describe un ataque de hombre en el medio sobre el protocolo de Diffie-Hellman, donde un adversario comparte una clave k_a con Alice y una clave diferente k_b con Bob, y Alice y Bob no pueden detectar esta intervención.

Para describir un ataque de hombre en el medio sobre el protocolo de Diffie-Hellman vamos a suponer que el atacante es $\mathcal{A}$, por lo que $\mathcal{A}$ pudo interceptar ya sea el mensaje de Alice hacia Bob o viceversa. S.p.g supongamos que fue el mensaje que Alice le envío a Bob donde venía su clave pública y $\mathcal{A}$ se hace pasar por Bob contestándole con su clave pública que es k_a y con Bob se hace pasar por Alice enviando como clave pública k_b.

De esa manera, $\mathcal{A}$ tiene el control de los mensajes que se están enviando Alice y Bob y éste ataque se da si no hay una forma de comprobar que esas claves realmente pertenecen a Alice y Bob. Así, $\mathcal{A}$ puede leer, recibir y modificar los mensaje que se envían entre ellos.

(b) Se tiene el siguiente protocolo de intercambio de clave:

i. Alicia escoge $k, r \leftarrow \{0,1\}^n$ al azar y envía $s := k \oplus r$ a Bartolo.
ii. Bartolo escoge $t \leftarrow \{0,1\}^n$ al azar y manda $u := s \oplus t$ a Alicia.
iii. Alicia calcula $w := u \oplus r$ y manda w a Bartolo.
iv. Alicia devuelve k y Bartolo devuelve $w \oplus t$.

Verifica que Alicia y Bartolo devuelven la misma clave, luego muestra que si Eva puede ver los mensajes intercambiados, entonces puede recuperar la clave.

Para mostrar que Alicia y Bartolo devuelven la misma llave entonces debemos mostrar que $k = w \oplus t$ y para ello partiremos de $w \oplus t$.
Ejecutando el protocolo de intercambio de clave podemos ver que:
$w = ((k \oplus r) \oplus t\ ) \oplus r$
Entonces haciendo $\oplus$ de ambos lados obtenemos:
$w \oplus t = (((k \oplus r) \oplus t) \oplus r) \oplus$ t
Ahora recordemos dos propiedades importantes de $\oplus$ que nos serán de ayuda para la demostración que son las siguientes:

(1) A $\oplus$ A = 0
(2) (B $\oplus$ A) $\oplus$ A = B $\oplus$ 0 = B por (1)

Entonces podemos asociar de la siguiente manera para ir eliminando cadenas y poder obtener a k del lado derecho de la igualdad.
$w \oplus t = (((k \oplus r) \oplus t) \oplus r) \oplus$ t
$w \oplus t = (k \oplus r) \oplus ((t \oplus r) \oplus t)$
$w \oplus t = (k \oplus r) \oplus r$ aplicando la propiedad (2)
$w \oplus t = k$ aplicando nuevamente la propiedad (2). Y el regreso es análogo.

$$\therefore k = w \oplus t$$

Después supongamos que Eva ha estado viendo de alguna manera los mensajes que se han mandado Alicia y Bartolo, si ella pudo ver los mensajes desde el inicio y conoce el protocolo de intercambio de llave, podrá hacerlo sin problemas pues sólo necesita hacer unos despejes para obtener la llave.
S.p.g supongamos que Eva obtendrá la llave como Bartolo la regresa, es decir, como $w \oplus t$ y para ello basta con ver el penúltimo mensaje para recuperar a w ya que Alicia se lo mandó a Bartolo y finalmente como Eva vio el mensaje s y u puede obtener a t realizando $u \oplus s$.

5. El siguiente protocolo sirve para que Alice y Bob generen un bit aleatorio.

(a) Una entidad confiable T publica su llave pública pk.
(b) Alice escoge un bit aleatorio b_A, lo encripta usando pk, con lo que obtiene c_A y lo manda a Bob y a T.
(c) Luego de recibir c_A, Bob hace lo mismo que Alice y manda $c_B \neq c_A$.
(d) T descifra c_A y c_B y anuncia el resultado. Alice y Bob hacen un XOR del resultado y este es el bit de salida.

Preguntas.

(a) Supón que Bob actúa honestamente pero Alice no. ¿Por qué aun así el bit de salida es aleatorio?

El protocolo tiene el siguiente orden: T, una entidad confiable, publica su llave pública. Alice lee esa pk, escoge un bit aleatorio y lo cifra con pk. Luego de recibir C_A, Bob escoge un bit aleatorio y lo encripta usando pk. Luego C_A *y* C_B son enviados a la entidad segura T.
La elección de valores disponibles de Alice es 1 o 0, además de que Alice no recibe otra información

aidicional más que la de la entidad T, una llave pública. Es por ello, que aunque Alice haga manipule la elección de su bit, ella no conoce nada respecto del de Bob, y por lo tanto, no sería trivial encontrar la forma de influir en el XOR de la entidad a su favor.

(b) Supón que en el protocolo se usa el cifrado de El Gamal. Muestra de qué forma Bob puede manipular el bit de salida para que resulte lo que él quiera. (Un bit b se codifica como el elemento g^b.)

En el protocolo se nos indica que, esta vez, quien sí posee el conocimiento del bit elegido por el otro participante en el protocolo es Bob. De esta forma, Alice genera su bit aleatorio, lo cifra con ElGamal y la llave pública de T y posteriormente lo envía a Bob.
Sabemos que, $C_1 = g^k (mod\ q)$ y $C_2 = g^b * h^k (mod\ q)$ es la forma del bit después del cifrado de Alice. Además, sabemos que:

- $g^0 = 1 (mod\ q)$
- $g^1 = g (mod\ q)$
- $gq - 1 = 1 (mod\ q)$

En específico, sabemos que los mensajes cifrados de Alice pueden verse de la forma:

- $C_1 = g^k (mod\ q)$ y $C_2 = 1 * h^k (mod\ q)$
- $C_1 = g^k (mod\ q)$ y $C_2 = g * h^k (mod\ q)$

De tal forma que, si Bob actúa de forma no honesta, sabe que si Alice envía 1 en la primera coordenada de C_2, entonces su bit elegido fue 0. Si Alice envía g como primer elemento en la coordenada de C_2, entonces eligió 1.
Así, si Bob desea que el resultado del XOR sea 0, entonces enviará el mismo bit que Alice. Si Bob desea que el resultado del XOR sea 1, entonces enviará el bit distinto al de Alice.

6. ¿Para qué es el siguiente código? En particular, ¿cuál es el significado de la entrada y la salida (n y p respectivamente)? ¿Qué propósito tiene la línea 7?

```
1  def funcion(n=16):
2      r = open('/dev/urandom', 'rb')
3      p = 4
4      while not test_miller_rabin(p):
5          s1 = r.read(n)
6          s2 = int.from_bytes(s1, 'big')
7          p = s2 | (1<<(n*8-1))
8      return p
```

El propósito de la función es devolver un número primo muy grande a partir de la lectura de una cadena de bytes de tamaño n en el archivo urandom.

El significado de la entrada, n, es el número de bytes que se espera que tenga el primo generado p, a saber, se toman una cadena de n bytes del archivo urandom para, posteriormente, convertirlo a int con bigendian y crear p.

Para crear un ejemplo específico para el funcionamiento de la línea 7, supongamos que no modificamos el parámetro default de 16 bytes. Con la línea 7, básicamente, se asegura que el primo generado sea exactamente de 16 bytes (que son 128 bits). El bit más significativo se fija como 1 y se agregan 0's hasta que se satisfaga que (n*8-1) (aquí se considera la cantidad de bits en n bytes). Posteriormente, en el *or* de bitewise, es donde se asegura que la longitud de s2 empate con la de $(1 << (n * 8 - 1))$. Es así como se asegura la longitud en bits que tenga el primo p.

7. (2 puntos) Para el sistema de ElGamal se tienen los siguientes parámetros de una curva elíptica E: $y^2 =$

$x^3 + ax + b$ sobre $\mathbb{F}_p$ con generador G:

$$\begin{aligned} p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ &= \texttt{0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff} \\ a &= -3 \\ b &= \texttt{0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b} \\ G &= (\texttt{0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296}, \\ &\quad \texttt{0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5}) \end{aligned}$$

Usando la clave pública

$$\begin{aligned} Q = (&\texttt{0xbbb041abbc4b2e9973726520c0a47b9db7d7a8d4e534f5c75d58acd68bf413a1}, \\ &\texttt{0xc567543991c0d7aac45ac9c325f10b62c77189e3f0a34d977a448bde60590e2d}) \end{aligned}$$

se encriptó un mensaje $M \in E(\mathbb{F}_p)$ y como resultado se obtuvieron los puntos

$$\begin{aligned} C_1 = (&\texttt{0x26efcebd0ee9e34a669187e18b3a9122b2f733945b649cc9f9f921e9f9dad812}, \\ &\texttt{0x90238bde9cc7bb330d150c67704dd25ae7055205744b6f31bf4070745872d0e6}) \text{ y} \\ C_2 = (&\texttt{0x46c4b5cc528f15953943e0a775d3a6d08057b1fa30878473aaa28399198c4f8c}, \\ &\texttt{0x3edff2f73f7a89cfddc967ad9af2f36a06b690761a5d25810efe223f906ce63}). \end{aligned}$$

Encuentra M y describe el procedimiento. *Hint:* La llave secreta es pequeña. Usa Sage.

Sabemos que en la criptografía de curvas elípticas para ElGamal, el procedimiento es distinto al usual:

Alicia primero crea una **llave privada**, a, el cual es un valor escalar. También crea una **llave pública**, $A = aG$, donde G es el punto base de la curva. Cuando Bob cifra un mensaje, éste se ve de la sigiuente forma $C_1 = kG$ y $C_2 = kQ + M$, a saber, k un escalar elegido de forma aleatoria por Bob y M su mensaje a cifrar. Cuando Alice quiere decifrar un mensaje de Bob, entonces recibe C_1 y C_2. Calcula $S = aC_1$ y $M = C_2 - S$.

Aplicándolo en nuestro caso en particular, nosotros conocemos cómo definir la curva elíptica por medio de los puntos dados. Consideremos los valores enviados por Bob. Sabemos que $C_1 = kG$. De igual forma, iteramos hasta que encontremos k que satisfaga la ecuación. Obtenemos:

$$k = 12345$$

Ahora, recordemos que $C_2 = kQ + M$. Queremos encontrar M, sin embargo, ya tenemos los valores de k, Q y C_2. Por lo tanto, podemos despejar:

$$C_2 - kQ = M$$

Después de los cálculos en Sage, sabemos que el mensaje en claro es el punto en la curva:

$$\begin{aligned} &(52689254947963902595739054002869248928648273535722072553023360335275059379383 : \\ &437842047377718508479893046281130508185910640123055934500954364518468283051 72 : 1) \end{aligned}$$

8. Se tiene la siguiente clave pública RSA

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKzl5VggSXb/Jm2oqkPeRQwtpmGlLnJT
Nre4LKx3VUljtLzYWj4xoG+aHBouwJT7DyeibpasCH8Yderr4zIGTNUCAwEAAQ==
-----END PUBLIC KEY-----
```

```
F = GF(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
a = -3
b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
E = EllipticCurve(F, (a,b))
p = 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff
G = E(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296,0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
Q = E(0xbbb041abbc4b2e9973726520c0a47b9db7d7a8d4e534f5c75d58acd68bf413a1,0xc567543991c0d7aac45ac9c325f10b62c77189e3f0a34d977a448bde60590e2d)

C1 = E(0x26efcebd0ee9e34a669187e18b3a9122b2f733945b649cc9f9f921e9f9dad812,0x90238bde9cc7bb330d150c67704dd25ae7055205744b6f31bf4070745872d0e6)
C2 = E(0x46c4b5cc528f15953943e0a775d3a6d08057b1fa30878473aaa28399198c4f8c,0x3edff2f73f7a89cfddc967ad9af2f36a06b690761a5d25810efe223f906ce63)
```

```
for i in range(1000000):
    if i*G == C1:
        k = i
        print(k)
        break

_________
M = C2 - (k*Q)
print("Este es el punto en el mensaje cifrado: ")
M
```

```
12345
Este es el punto en el mensaje cifrado:
(52689254947963902595739054002869248928648273535722072553023360335275059379383 : 437842047377718508479893046281130508185910640123055934500954364518468283051 72 : 1)
```

y un mensaje que fue cifrado con ella

```
0x41b4e1609390ff8fb5f225b010d1cc79253dcab1744d5f865daabad0e28d2591
41722382114d9a73106b4d429676dae60a1528a0eb3b73eab0e9d2165c72492f
```

Se usó un generador de números aleatorios deficiente, y enseguida de obtener los primos para la clave anterior se obtuvieron los primos para la clave siguiente

```
-----BEGIN PUBLIC KEY-----
MF0wDQYJKoZIhvcNAQEBBQADTAAwSQJCAPsrpwx56OTlKtGAWn24bo5HUg3xYtnz
nTj1X/8Hq7pLYNIVE57Yxoyr3zTOOBJufgTNzdKSORc5Ti4zZUkCkQvpAgMBAAE=
-----END PUBLIC KEY-----
```

Descifra el mensaje y explica cómo lo encontraste. *Hint:* $|\{p_1, q_1, p_2, q_2\}| = 3$.

Sabemos que en RSA trabaja con números primos grandes, cuya propiedad es que sea dificíl encontrar sus factores primos, a saber, p y q.
Tenemos dos llaves públicas. En RSA, las llaves públicas se ven de la forma

$$publicKey = (e, n)$$

Para este ejercicio se utilizó la biblioteca *pycryptodome* para obtener la información de las llaves públicas. En concreto, las tuplas que pertenecen al par de laves dadas son

- $publicKey_1$ = (65537, 90554046405003001094058011529356632671762183207853485415666639821721622657784451073200651874490623755250026320437227345665931854619992866252345280366051 41)
- $publicKey_2$= (65537, 33676460591388774425798209728318764120062799170978090822794128516931239559642825451455004973935795989548595347318904602291943723392150985067883750506984273 69)

Lo que sabemos, es que cada n en las tuplas de las llaves está compuesta de la forma

$$n_1 = p_1 * q_1$$
$$n_2 = p_2 * q_2$$

Por el hint ($|\{p_1, q_1, p_2, q_2\}| = 3$), sabemos que de los factores primos, dos son el mismo número.
Sin pérdida de generalidad, supongamos entonces que $p_1 = p_2$, esto quiere decir que n_1 y n_2 no son coprimos entre sí, pues tienen factores primos en común. Ahora, obtengamos su máximo común divisor para obtener el factor primo que tienen en común.

$$p_1, p_2 = gcd\,(n_1\ n_2) = 84131146998723013035726124824393467539823449570683991030677848137795334610229$$

La llave utilizada para cifrar el mensaje fue $publicKey_1$, por lo tanto ahora conocemos que

$$n_1 = 84131146998723013035726124824393467539823449570683991030677848137795334610229 * q_1$$

Conocemos el valor de n_1 y p_1, obtengamos el valor de q_1 despejando:

$$q_1 = n_1//p_1$$

$$q_1 = 107634389444824136554734231305548679460065919884322199796801660366105198986529$$

Ahora ya tenemos todos los valores necesarios para ejecutar el algoritmo RSA.
Como resultado, obtenemos el mensaje en claro:

195894762

9. En el protocolo TLS, el primer paso para establecer una conexión es hacer un *handshake*. Explica las diferencias que hay en el proceso de handshake entre las versiones 1.2 y 1.3 de TLS.T
Primero, es conveniente explicar de forma breve en qué consisten ambos handshakes.

Para TLS 1.2:

- El cliente le envía al servidor un mensaje de saludo. Éste contiene información criptográfica, por ejemplo, los protocolos y los CipherSuites con los que tiene compatibilidad, así como un valor o una cadena de bytes aleatorio.
- El servidor le envía un saludo devuelta al cliente. En éste incluye el CipherSuite que el servidor a elegido del listado de los compatibles con el cliente. También se envía el id de la sesión, el certificado del servidory se agrega otro valor aleatorio.
- El cliente recibe la información del certificado y la verifica. Una vez lista la verificación, le envía al servidor un "pre-secret master", que es una cadena de bytes aleatoria cifrada con la llave pública del certificado del servidor.
- Cuando el servidor recibe el "pre-secret master" del cliente, tanto el cliente como el servidor generan una clave maestra junto con las claves de sesión. Éstas últimas serán utilizadas para el cifrado simétrico de los datos.
- El cliente envía un "Change Cipher Spec" al servidor, donde le avisa que va cambiarse al cifrado simétrico con la ayuda de las claves de sesión. Además, envía un mensaje de despedida, un "Client Finished".
- El servidor recibe el "Change Cipher Spec" del cliente y cambia también al cifrado simétrico. Finalmente, le envía un mensaje de despedida al cliente, un "Server finished".

Para TLS 1.3:

- El cliente le envía un mensaje de saludo al servidor. En éste, envía los CipherSuites con los que tiene compatibilidad y adivina qué protocolo de acuerdo de llave es probable que seleccione el servidor. De igual forma, el cliente también envía su llave compartida para ese protocolo de acuerdo de llave que adivinó.
- El servidor responde el saludo del cliente con el protocolo de acuerdo de llave que ha elegido. En este mensaje, también incluye su certificado y su llave compartida. Todo esto conforma el mensaje "Server Finished" del servidor.
- El cliente recibe el mensaje del servidor y verifica el certificado, genera llaves puesto que ya tiene la llave compartida del servidor. y envía el mensaje "Client Finished". Es entonces que comienza el cifrado de los datos.

Las diferencias:

Resulta sencillo ver que el protocolo handshake TLS 1.3 mejora muchísimo la cantidad de pasos necesarios para comenzar el intercambio de datos cifrados entre el cliente y el servidor, pues el cliente desde el primer mensaje establece más cosas hacia el servidor. Esto ahorra y mejora mucho el tiempo de espera antes del comienzo de envío de información cifrada.
De igual forma, en TLS 1.3 se usa 0-RTT Resumption, lo cual hace que si el cliente ya se ha conectado al servidor antes, entonces ya no se ejecuten los pasos necesarios para el handshake, pues la información sensible ya ha sido generada y almacenada en sesiones previas.