Presentación

En esta práctica estudiaremos el algoritmo DSA y las funciones de *hash*. En primer lugar, implementaremos las funciones que nos permitirán firmar y validar firmas usando el estándar Digital Signature Standard (DSS), que implementa el algoritmo de clave pública DSA. El detalle de este sistema de firma digital lo podéis encontrar en el módulo 5: "Firmas digitales". Posteriormente, utilizaremos la función *hash* SHA-1 para ver la dificultad de obtener mensajes diferentes que tienen el mismo *hash*. Finalmente, realizaremos un ataque contra las firmas digitales DSA.

Objetivos

Los objetivos de esta práctica son:

1. Comprender como funciona el algoritmo de firma DSA.
2. Entender las propiedades de las funciones *hash* y los ataques que se pueden realizar sobre estas.
3. Entender el ataque planteado sobre las firmas DSA.

Descripción de la Práctica a realizar

1. El algoritmo de firma DSA (3,5 puntos)

En este ejercicio implementaremos el algoritmo de firma DSA. Para hacerlo podéis usar la librería `Sympy`, que os proporcionará funciones útiles como por ejemplo `mod_inverse()` o `randprime()`. Así mismo, tened en cuenta que la función `pow()` de Python, permite realizar operaciones modulares proporcionando como tercer parámetro el valor del módulo.

1.1. Función que implementa la generación de un par de claves DSA (1,5 puntos)

Esta función implementará el funcionamiento de la generación de claves DSA tal y como se explica en el módulo didáctico 5 de los materiales de la asignatura.

La función tomará como parámetros de entrada los valores `L` y `N` del algoritmo DSA y retornará una lista con el par de claves generado, en formato $[p, q, g, y]$ para la clave pública y $[p, q, g, x]$ para la clave privada.

- La variable `L` contendrá un número entero.
- La variable `N` contendrá un número entero.

- La función retornará la clave pública y la privada en formato $[[p, q, g, y], [p, q, g, x]]$.

1.2. Función que implementa la firma digital con DSA (1 punto)

Esta función implementará la firma DSA tal y como se explica en el módulo didáctico 5 de los materiales de la asignatura.

La función tomará como variables dos parámetros: `privkey` y `message`; y retornará el el mensaje firmado.

En el esqueleto de la práctica se os proporciona una clase llamada `UOCRandom`. La propia clase tiene un ejemplo de uso. Tened en cuenta que para que funcionen los tests de este ejercicio, tenéis que generar el valor k usando esta clase.

- La variable `privkey` contendrá la clave privada en formato $[p, q, g, x]$
- La variable `message` contendrá un número que representa el mensaje.
- La función retornará una lista con la firma en formato $[r, s]$.

1.3. Función que implementa la validación de la firma DSA (1 punto)

La función recibirá como argumentos una clave pública, el mensaje a validar (que igual que en la función anterior, será un valor numérico) y el valor de la firma.

- La variable `pubkey` contendrá la clave pública en formato $[p, q, g, y]$.
- La variable `message` contendrá un número que representa el mensaje del que se quiere validar la firma.
- La variable `signature` contendrá la firma a validar en formato $[r, s]$.
- La función retornará True o False, indicando si la firma se ha validado correctamente o no.

2. Análisis de las funciones *hash* (3,5 puntos)

En este ejercicio utilizaremos la función *hash* SHA-1 para analizar la importancia de la seguridad de las funciones *hash*. Para hacerlo, utilizaremos la implementación del algoritmo SHA-1 de la librería `hashlib` de Python . Además, para que la complejidad computacional de los cálculos a realizar no sea desproporcionada, usaremos una simplificación de la función SHA-1 que consistirá en utilizar únicamente algunos de sus bits de salida.

2.1. Función que implementa una función *hash* basada en SHA-1. (0,5 puntos)

La función recibirá como argumentos un mensaje y el número de bits. En el caso de que el mensaje sea un número entero, este se convertirá en un *string* antes de realizar el cálculo del *hash*.

- La variable `message` contendrá un número entero o un mensaje de texto en ASCII del que calcularemos el *hash*.
- La variable `num_bits` contendrá un número de bits (entre 1 y 160, y múltiplo de 4).
- La función retornará los `num_bits` menos significativos de la salida SHA-1 del mensaje, en formato hexadecimal.

2.2. Función que encuentra segundas preimágenes para la función *hash*. (1,5 puntos)

La función recibirá como argumentos un mensaje y el número de bits

- La variable `message` contendrá el mensaje de texto en ASCII del que queremos encontrar una preimagen del su *hash*.
- La variable `num_bits` contendrá un número de bits (entre 1 y 160, y múltiplo de 4).
- La función retornará un mensaje, diferente del proporcionado en la entrada, que tenga el mismo valor *hash* que el mensaje proporcionado.

2.3. Función que encuentra colisiones en la función *hash*. (1,5 puntos)

La función recibirá como argumento `num_bits`, el tamaño en bits (entre 1 y 160).

- La variable `num_bits` contendrá un número de bits (entre 1 y 160, y múltiple de 4).
- La función retornará una lista con dos mensajes diferentes que tengan el mismo valor *hash*.

3. Ataque a la firma DSA e implementación determinista (3 puntos)

Tal y como se explica en el Apartado 4.3.3 del módulo didáctico 5, existe un ataque a la firma digital de ElGamal que permite extraer la clave privada a partir de dos firmas que han usado el mismo k.

El valor de k se genera aleatoriamente y el rango suele ser suficientemente grande como para que la probabilidad de crear dos firmas con el mismo k sea lo suficientemente baja. Sin embargo,

en dispositivos donde es complicado disponer de una fuente de números aleatorios adecuada, la probabilidad de repetir k podría ser suficientemente alta como para suponer un riesgo.

Primero implementaremos este ataque, después veremos una posible solución mediante un mecanismo de firma determinista.

3.1. Función que implementa un ataque a DSA de recuperación de clave. (2 punto)

En el Apartado 4.3.3 del módulo didáctico 5 se explica un ataque a la firma de ElGamal, que permite recuperar la clave privada usando dos firmas que han repetido el parámetro h.

En este ejercicio tenéis que implementar una función que, dadas dos firmas digitales con DSA que han generado la misma k, se pueda extraer la clave privada que ha realizado la firma. Para hacerlo, se tendrán que modificar las ecuaciones que se muestran en el Apartado 4.4.4 del módulo didáctico 5 para el caso de ElGamal, para que se ajusten a la firma DSA. No es necesario que adjuntéis los cálculos con las modificaciones de estas ecuaciones, solo tenéis que realizar la implementación en el fichero de la práctica.

La función recibirá como argumentos la clave pública y dos firmas con sus correspondientes mensajes, y retornará la clave privada.

- La variable `pubkey` contendrá la clave pública en formato $[p, q, g, y]$.
- La variable `m1` contendrá un número que representa el primer mensaje.
- La variable `sig1` contendrá la firma del primer mensaje $[r, s]$.
- La variable `m2` contendrá un número que representa el segundo mensaje.
- La variable `sig2` contendrá la firma del segundo mensaje $[r, s]$.
- La función retornará la clave privada en formato $[p, q, g, x]$.

3.2. Función que implementa la firma DSA determinista (1 punto)

En el Apartado 4.3.3 del módulo didáctico 5 se indica la posibilidad de implementar una firma determinista, usando una función *hash* con la clave privada y el mensaje, elementos que servirán para inicializar un generador de números aleatorios.

En esta función tenéis que implementar la firma DSA determinista. Usad la clase `UOCRandom` proporcionada en el esqueleto de la práctica. Inicializad el generador de números aleatorios usando un número entero como semilla. La semilla será el número entero resultante de concatenar el valor hexadecimal del *hash* de la clave privada con el valor hexadecimal del *hash* del mensaje. Usad la función *hash* `uoc_sha1` que habéis implementado con 64 bits.

La función recibirá dos parámetros: `privkey` y `message`; y retornará el mensaje firmado.

- La variable `privkey` contendrá la clave privada en formato $[p, q, g, x]$
- La variable `message` contendrá un número que representa el mensaje.
- La función retornará una lista con la firma en formato $[r, s]$.

Criterios de valoración

La puntuación de cada ejercicio se encuentra detallada en el enunciado.

Por otro lado, es necesario tener en cuenta que el código que se envíe debe contener los comentarios necesarios para facilitar su seguimiento. En caso de no incluir comentarios, la corrección de la práctica se realizará únicamente de forma automática y no se proporcionará una corrección detallada. No incluir comentarios puede ser motivo de reducción de la nota.

Formato y fecha de entrega

La fecha máxima de envío es el **29/05/2020** (a las 24 horas).

Junto con el enunciado de la práctica encontrareis el esqueleto de la misma (fichero con extensión .py). Este archivo contiene las cabeceras de las funciones que hay que implementar para resolver la práctica. Este mismo archivo es el que se debe entregar una vez se codifiquen todas las funciones. No se tiene que enviar el fichero comprimido. No se aceptarán ficheros en otros foramtos que no sean .py.

Adicionalmente, también os proporcionaremos un fichero con tests unitarios para cada una de las funciones que hay que implementar. Podeis utilizar estos tests para comprobar que vuestra implementación gestiona correctamente los casos principales, así como para obtener más ejemplos concretos de lo que se espera que retornen las funciones (más allá de los que ya se proporcionan en este enunciado). Nótese, sin embargo, que los tests no son exhaustivos (no se prueban todas las entradas posibles de las funciones). Recordad que no se puede modificar ninguna parte del archivo de tests de la práctica.

La entrega de la práctica constará de un único fichero Python (extensión .py) donde se haya incluido la implementación.