Laboratorio de cifrado de clave secreta

1. Información general

El objetivo de aprendizaje de esta práctica de laboratorio es que los estudiantes se familiaricen con los conceptos del cifrado de clave secreta y algunos ataques comunes al cifrado. A partir de este laboratorio, los estudiantes obtendrán una experiencia de primera mano sobre algoritmos de cifrado, modos de cifrado, rellenos y vector inicial (IV). Además, los estudiantes podrán usar herramientas y escribir programas para cifrar/descifrar mensajes.

Los desarrolladores han cometido muchos errores comunes al usar los algoritmos y modos de encriptación.
Estos errores debilitan la fuerza de la encriptación y, eventualmente, conducen a vulnerabilidades. Este laboratorio expone a los estudiantes a algunos de estos errores y les pide que lancen ataques para explotar esas vulnerabilidades.
Este laboratorio cubre los siguientes temas:

• Cifrado de clave secreta •
Cifrado de sustitución y análisis de frecuencia • Modos de
cifrado, IV y rellenos • Errores comunes en el uso de
algoritmos de cifrado • Programación usando la biblioteca criptográfica

Lecturas. La cobertura detallada del cifrado de clave secreta se puede encontrar en lo siguiente:

• Capítulo 21 del libro de SEED, Computer & Internet Security: A Hands-on Approach, 2nd Edition,
por Wenliang Du. Consulte los detalles en https://www.handsonsecurity.net.

Ambiente de laboratorio. Este laboratorio se probó en nuestra máquina virtual Ubuntu 20.04 preconstruida, que se puede descargar desde el sitio web de SEED.

2 entorno de laboratorio

En esta práctica de laboratorio, usamos un contenedor para ejecutar un oráculo de cifrado. El contenedor solo se necesita en la Tarea 6.3, por lo que no necesita iniciar el contenedor para otras tareas.

Configuración y comandos del contenedor. Descargue el archivo Labsetup.zip a su máquina virtual desde el sitio web del laboratorio, descomprímalo, ingrese a la carpeta Labsetup y use el archivo docker-compose.yml para configurar el entorno del laboratorio. Puede encontrar una explicación detallada del contenido de este archivo y todo el Dockerfile involucrado en el manual del usuario, que está vinculado al sitio web de este laboratorio. Si es la primera vez que configura un entorno de laboratorio de SEED utilizando contenedores, es muy importante que lea el manual del usuario.

A continuación, enumeramos algunos de los comandos de uso común relacionados con Docker y Compose. Dado que vamos a utilizar estos comandos con mucha frecuencia, hemos creado alias para ellos en el archivo .bashrc (en nuestra VM SEEDUbuntu 20.04 provista).

```
$ docker-compose build # Construye la imagen del contenedor $ docker-compose
up                                    # Iniciar el contenedor $ docker-
compose down # Cerrar el contenedor

// Alias para los comandos Compose anteriores $ dcbuild
                                # Alias para: docker-compose build $dcup
                                # Alias para: docker-compose up $ dcdown
                                # Alias para: docker-compose down
```

Todos los contenedores se ejecutarán en segundo plano. Para ejecutar comandos en un contenedor, a menudo necesitamos obtener un shell en ese contenedor. Primero debemos usar el comando "docker ps" para averiguar la ID del contenedor y luego usar "docker exec" para iniciar un shell en ese contenedor. Hemos creado alias para ellos en el archivo .bashrc.

```
$ dockps // Alias para: docker ps --format "{{.ID}} {{.Names}}" $ docksh <id> // Alias para: docker exec -it <id> /
bin/bash

// El siguiente ejemplo muestra cómo obtener un shell dentro de hostC $ dockps b1004832e275
hostA-10.9.0.5 0af4ea7a3e2e hostB-10.9.0.6 9652715c8e0a hostC-10.9.0.7




$ muelle 96
root@9652715c8e0a:/#

// Nota: si un comando de docker requiere un ID de contenedor, no es necesario que // escriba la cadena de ID
completa. Escribir los primeros caracteres // será suficiente, siempre que sean únicos entre todos los contenedores.
```

Si encuentra problemas al configurar el entorno de laboratorio, lea los "Problemas comunes"
sección del manual para posibles soluciones.

3 Tarea 1: Análisis de frecuencia

Es bien sabido que el cifrado de sustitución monoalfabético (también conocido como cifrado monoalfabético) no es seguro, porque puede estar sujeto a análisis de frecuencia. En esta práctica de laboratorio, se le proporciona un texto cifrado que se cifra mediante un cifrado monoalfabético; es decir, cada letra del texto original se reemplaza por otra letra, donde el reemplazo no varía (es decir, una letra siempre se reemplaza por la misma letra durante el cifrado). Tu trabajo es encontrar el texto original usando análisis de frecuencia. Se sabe que el texto original es un artículo en inglés.

A continuación, describimos cómo encriptamos el artículo original y qué simplificación hemos hecho.
Los profesores pueden usar el mismo método para encriptar un artículo de su elección, en lugar de pedirles a los estudiantes que usen el texto cifrado creado por nosotros.

- Paso 1: vamos a generar la clave de cifrado, es decir, la tabla de sustitución. Permutaremos el alfabeto de la a a la z usando Python, y usaremos el alfabeto permutado como clave. Ver el siguiente programa.

```
#!/bin/env python3

importar al azar
```

```
s = "abcdefghijklmnopqrstuvwxyz" list =
random.sample(s, len(s)) key = ''.join(list) print(key)
```

- Paso 2: hagamos una simplificación al artículo original. Convertimos todas las mayúsculas a minúsculas y luego eliminamos todos los signos de puntuación y los números. Mantenemos los espacios entre las palabras, por lo que aún puede ver los límites de las palabras en el texto cifrado. En el cifrado real con cifrado monoalfabético, se eliminarán los espacios. Mantenemos los espacios para simplificar la tarea. Esto lo hicimos usando el siguiente comando:

```
$ tr [:superior:] [:inferior:] < artículo.txt > minúsculas.txt $ tr -cd '[az][\n][:espacio:]' <
minúsculas.txt > texto sin formato.txt
```

- Paso 3: usamos el comando tr para hacer el cifrado. Solo encriptamos letras, dejando el espacio y devolver personajes solos.

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' \
        < texto simple.txt > texto cifrado.txt
```

Hemos creado un texto cifrado utilizando una clave de cifrado diferente (no la descrita anteriormente). Se incluye en el archivo Labsetup.zip, que puede descargarse del sitio web del laboratorio. Su trabajo consiste en utilizar el análisis de frecuencia para averiguar la clave de cifrado y el texto sin formato original.

También proporcionamos un programa de Python (freq.py) dentro de la carpeta Labsetup/Files. Lee el archivo ciphertext.txt y produce las estadísticas de n-gramas, incluidas las frecuencias de una sola letra, frecuencias de bigramas (secuencia de 2 letras) y frecuencias de trigramas (secuencia de 3 letras), etc.

```
$ ./frecuencia.py
----------------------------------
1 gramo (top 20): n: 488
y: 373 v: 348


 ...
----------------------------------
2 gramos (top 20): yt:
115 tn: 89

mu: 74
 ...
----------------------------------
3 gramos (top 20): ytn:
78 vup: 30 mur: 20


 ...
```

Pautas. Usando el análisis de frecuencia, puede encontrar el texto sin formato de algunos de los caracteres con bastante facilidad. Para esos caracteres, es posible que desee volver a cambiarlos a su texto sin formato, ya que puede obtener más pistas. Es mejor usar letras mayúsculas para el texto sin formato, por lo que para la misma letra, sabemos cuál es el texto sin formato

y cuál es el texto cifrado. Puede usar el comando tr para hacer esto. Por ejemplo, a continuación, reemplazamos las letras a, e y t en in.txt con las letras X, G, E, respectivamente; los resultados se guardan en out.txt.

```
$ tr 'aet' 'XGE' < entrada.txt > salida.txt
```

Hay muchos recursos en línea que puede utilizar. A continuación, enumeramos algunos enlaces útiles:

- https://en.wikipedia.org/wiki/Frequency_analysis: esta página de Wikipedia proporciona frecuencias para un texto sin formato típico en inglés.
- https://en.wikipedia.org/wiki/Bigram: frecuencia de Bigram.
- https://en.wikipedia.org/wiki/Trigram: frecuencia de trigramas.

4 Tarea 2: Cifrado utilizando diferentes cifrados y modos

En esta tarea, jugaremos con varios algoritmos y modos de encriptación. Puede usar el siguiente comando openssl enc para cifrar/descifrar un archivo. Para ver los manuales, puede escribir man openssl y man enc.

```
$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \
                -K 00112233445566778889aabbccddeeff\
                -iv 0102030405060708
```

Reemplace el tipo de cifrado con un tipo de cifrado específico, como -aes-128-cbc, -bf-cbc, -aes-128-cfb, etc. En esta tarea, debe probar al menos 3 cifrados diferentes. Puede encontrar el significado de las opciones de la línea de comandos y todos los tipos de cifrado admitidos escribiendo "man enc". Incluimos algunas opciones comunes para el comando openssl enc a continuación:

-en <archivo>	archivo de
-out <archivo>	entrada archivo
-mi	de salida cifrar
-d	descifrar clave/iv
-K/-iv -[pP]	en hexadecimal es el siguiente argumento imprimir la iv/clave (luego salir si -P)

5 Tarea 3: Modo de cifrado: ECB frente a CBC

El archivo pic original.bmp está incluido en el archivo Labsetup.zip y es una imagen simple. Nos gustaría encriptar esta imagen, para que las personas sin las claves de encriptación no puedan saber qué hay en la imagen.
Encripte el archivo utilizando los modos ECB (Libro de códigos electrónicos) y CBC (Encadenamiento de bloques cifrados), y luego haga lo siguiente:

1. Tratemos la imagen cifrada como una imagen y usemos un software de visualización de imágenes para mostrarla. Sin embargo, para el archivo .bmp, los primeros 54 bytes contienen la información del encabezado sobre la imagen, tenemos que configurarlo correctamente, para que el archivo cifrado pueda tratarse como un archivo .bmp legítimo. Reemplazaremos el encabezado de la imagen cifrada con el de la imagen original. Podemos usar la herramienta de edición hexadecimal bless (ya instalada en nuestra máquina virtual) para modificar directamente los archivos binarios. También podemos usar los siguientes comandos para obtener el encabezado de p1.bmp, los datos de p2.bmp (desde el desplazamiento 55 hasta el final del archivo), y luego combinar el encabezado y los datos en un nuevo archivo.

```
$ cabeza -c 54 p1.bmp > encabezado $ cola
-c +55 p2.bmp > cuerpo $ gato encabezado
cuerpo > nuevo.bmp
```

2. Muestre la imagen cifrada utilizando un programa de visualización de imágenes (hemos instalado un programa de visualización de imágenes llamado eog en nuestra máquina virtual). ¿Puede obtener alguna información útil sobre la imagen original a partir de la imagen cifrada? Por favor explique sus observaciones.

Seleccione una imagen de su elección, repita el experimento anterior e informe sus observaciones.

6 Tarea 4: Relleno

Para cifrados de bloque, cuando el tamaño de un texto sin formato no es un múltiplo del tamaño del bloque, es posible que se requiera relleno. El esquema de relleno PKCS#5 es ampliamente utilizado por muchos cifrados de bloque (consulte el Capítulo 21.4 del libro SEED para obtener más detalles). Realizaremos los siguientes experimentos para comprender cómo funciona este tipo de relleno:

1. Utilice los modos ECB, CBC, CFB y OFB para cifrar un archivo (puede elegir cualquier cifrado). Informe qué modos tienen rellenos y cuáles no. Para aquellos que no necesitan rellenos, explique por qué.

2. Vamos a crear tres archivos, que contienen 5 bytes, 10 bytes y 16 bytes, respectivamente. Podemos usar el siguiente comando "echo -n" para crear tales archivos. El siguiente ejemplo crea un archivo f1.txt con una longitud de 5 (sin la opción -n, la longitud será de 6, porque el eco agregará un carácter de nueva línea):

```
$ echo -n "12345" > f1.txt
```

Luego usamos "openssl enc -aes-128-cbc -e" para cifrar estos tres archivos usando AES de 128 bits con modo CBC. Describa el tamaño de los archivos cifrados.

Nos gustaría ver qué se agrega al relleno durante el cifrado. Para lograr este objetivo, descifraremos estos archivos usando "openssl enc -aes-128-cbc -d". Desafortunadamente, el descifrado predeterminado eliminará automáticamente el relleno, lo que nos impedirá verlo.

Sin embargo, el comando tiene una opción llamada "-nopad", que desactiva el relleno, es decir, durante el descifrado, el comando no eliminará los datos rellenos. Por lo tanto, al observar los datos descifrados, podemos ver qué datos se utilizan en el relleno. Utilice esta técnica para averiguar qué rellenos se agregan a los tres archivos.

Cabe señalar que es posible que los datos de relleno no se puedan imprimir, por lo que debe usar una herramienta hexadecimal para mostrar el contenido. El siguiente ejemplo muestra cómo mostrar un archivo en formato hexadecimal:

```
$ hexdump -C p1.txt 00000000
31 32 33 34 35 36 37 38 39 49 4a 4b 4c 0a |123456789IJKL.| $ xxd p1.txt 00000000: 3132 3334 3536 3738 3949
4a4b 4c0a
                                                    123456789IJKL.
```

7 Tarea 5: Propagación de errores: texto cifrado dañado

Para comprender la propiedad de propagación de errores de varios modos de cifrado, nos gustaría hacer el siguiente ejercicio:

1. Cree un archivo de texto que tenga al menos 1000 bytes de longitud.
2. Cifre el archivo con el cifrado AES-128.
3. Desafortunadamente, un solo bit del byte 55 en el archivo cifrado se corrompió. Puedes lograr esto corrupción usando el editor hexadecimal bless.
4. Descifre el archivo de texto cifrado corrupto usando la clave y el IV correctos.

Responda la siguiente pregunta: ¿Cuánta información puede recuperar descifrando el archivo dañado, si el modo de cifrado es ECB, CBC, CFB u OFB, respectivamente? Responda esta pregunta antes de realizar esta tarea y luego averigüe si su respuesta es correcta o incorrecta después de terminar esta tarea. Proporcione una justificación.

8 Tarea 6: Vector inicial (IV) y errores comunes

La mayoría de los modos de cifrado requieren un vector inicial (IV). Las propiedades de un IV dependen del esquema criptográfico utilizado. Si no tenemos cuidado al seleccionar los IV, es posible que los datos cifrados por nosotros no sean seguros en absoluto, aunque estemos utilizando un algoritmo y modo de cifrado seguro. El objetivo de esta tarea es ayudar a los estudiantes a comprender los problemas si no se selecciona correctamente un IV. Las pautas detalladas para esta tarea se proporcionan en el Capítulo 21.5 del libro SEED.

8.1 Tarea 6.1. Experimento IV

Un requisito básico para IV es la unicidad, lo que significa que ningún IV puede reutilizarse bajo la misma clave. Para comprender por qué, cifre el mismo texto sin formato usando (1) dos IV diferentes y (2) el mismo IV. Describa su observación, en base a la cual, explique por qué IV debe ser único.

8.2 Tarea 6.2. Error común: usar el mismo IV

Se puede argumentar que si el texto sin formato no se repite, usar el mismo IV es seguro. Veamos el modo de retroalimentación de salida (OFB). Supongamos que el atacante obtiene un texto sin formato (P1) y un texto cifrado (C1), ¿puede descifrar otros mensajes cifrados si el IV es siempre el mismo? Se le proporciona la siguiente información, intente averiguar el contenido real de P2 en función de C2, P1 y C1.

```
Texto sin formato (P1): ¡Este es un mensaje conocido!
Texto cifrado (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

Texto sin formato (P2): (desconocido para usted)
Texto cifrado (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
```

Si reemplazamos OFB en este experimento con CFB (Cipher Feedback), ¿cuánto de P2 se puede revelar? Solo necesita responder la pregunta; no hay necesidad de demostrar eso.

El ataque utilizado en este experimento se denomina ataque de texto sin formato conocido, que es un modelo de ataque para criptoanálisis en el que el atacante tiene acceso tanto al texto sin formato como a su versión cifrada (texto cifrado). Si esto puede dar lugar a la revelación de más información secreta, el esquema de cifrado no se considera seguro.

Código de muestra. Proporcionamos un programa de muestra llamado código de muestra.py, que se puede encontrar dentro de la carpeta Labsetup/Files. Le muestra cómo XOR cadenas (cadenas ascii y cadenas hexadecimales). El código se muestra a continuación:

```
#!/usr/bin/python3

# XOR dos bytearrays def
xor(primero, segundo):
    return bytearray(xˆy para x,y en zip(primero, segundo))

MSJ = "Un mensaje"
HEX_1 = "aabbccddeeff1122334455"
HEX_2 = "1122334455778800aabbdd"

# Convertir cadena ascii/hex a bytearray D1 = bytes(MSG, 'utf-8')

D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2) r2 =
xor(D2, D3) r3 = xor(D2,
D2) imprimir(r1.hex())
imprimir(r2.hex())
imprimir(r3.hex())
```

8.3 Tarea 6.3. Error común: usar un IV predecible

De las tareas anteriores, ahora sabemos que los IV no se pueden repetir. Otro requisito importante en IV es que los IV deben ser impredecibles para muchos esquemas, es decir, los IV deben generarse aleatoriamente. En esta tarea, veremos qué sucederá si los IV son predecibles.

Suponga que Bob acaba de enviar un mensaje cifrado y Eve sabe que su contenido es Sí o No; Eve puede ver el texto cifrado y el IV utilizado para cifrar el mensaje, pero dado que el algoritmo de cifrado AES es bastante fuerte, Eve no tiene idea de cuál es el contenido real. Sin embargo, dado que Bob usa IV predecibles, Eve sabe exactamente qué IV usará Bob a continuación.

Un buen cifrado no solo debe tolerar el ataque de texto sin formato conocido descrito anteriormente, sino que también debe tolerar el ataque de texto sin formato elegido, que es un modelo de ataque para el criptoanálisis donde el atacante puede obtener el texto cifrado para un texto sin formato arbitrario. Dado que AES es un cifrado fuerte que puede tolerar el ataque de texto sin formato elegido, a Bob no le importa cifrar cualquier texto sin formato proporcionado por Eve; él usa un IV diferente para cada texto sin formato, pero desafortunadamente, los IV que genera no son aleatorios y siempre pueden ser predecibles.

Tu trabajo es construir un mensaje y pedirle a Bob que lo cifre y te proporcione el texto cifrado. Su objetivo es aprovechar esta oportunidad para averiguar si el contenido real del mensaje secreto de Bob es Sí o No. Para esta tarea, se le proporciona un oráculo de encriptación que simula a Bob y encripta el mensaje con AES de 128 bits con modo CBC. Puede obtener acceso al oráculo ejecutando el siguiente comando:

```
$ nc 10.9.0.80 3000 El mensaje
secreto de Bob es "Sí" o "No", sin comillas.
Cifrado de Bob: 54601f27c6605da997865f62765117ce El IV utilizado:
d27d724f59a84d9b61c0f2883efa7bbc

Siguiente IV                    : d34c739f59a84d9b61c0f2883efa7bbc
```

```
Su texto sin formato: 11223344aabbccdd
Su texto cifrado: 05291d3169b2921f08fe34449ddc3611

Siguiente IV : cd9f1ee659a84d9b61c0f2883efa7bbc
Su texto sin formato: <su entrada>
```

Después de mostrarle el siguiente IV, el oráculo le pedirá que ingrese un mensaje de texto sin formato (como una cadena hexadecimal). El oráculo cifrará el mensaje con el siguiente IV y generará el nuevo texto cifrado. Puede probar diferentes textos sin formato, pero tenga en cuenta que cada vez, el IV cambiará, pero es predecible. Para simplificar su trabajo, dejamos que el oráculo imprima el siguiente IV. Para salir de la interacción, presione Ctrl+C.

8.4 Lecturas adicionales

Hay criptoanálisis más avanzados en IV que están más allá del alcance de este laboratorio. Los estudiantes pueden leer el artículo publicado en esta URL: https://defuse.ca/cbcmodeiv.htm. Debido a que los requisitos de IV realmente dependen de los esquemas criptográficos, es difícil recordar qué propiedades se deben mantener cuando seleccionamos un IV. Sin embargo, estaremos a salvo si siempre usamos un nuevo IV para cada encriptación, y el nuevo IV debe generarse usando un buen generador de números pseudoaleatorios, por lo que es impredecible para los adversarios.

Consulte otro laboratorio de SEED (Laboratorio de generación de números aleatorios) para obtener detalles sobre cómo generar números pseudoaleatorios criptográficamente fuertes.

9 Tarea 7: Programación usando la biblioteca criptográfica

Esta tarea está diseñada principalmente para estudiantes de Informática/Ingeniería o campos relacionados, donde se requiere programación. Los estudiantes deben consultar con sus profesores para ver si esta tarea es necesaria para sus cursos o no.

En esta tarea, se le proporciona un texto sin formato y un texto cifrado, y su trabajo es encontrar la clave que se utiliza para el encriptación Conoces los siguientes hechos:

- El cifrado aes-128-cbc se utiliza para el cifrado.
- La clave utilizada para cifrar este texto sin formato es una palabra en inglés de menos de 16 caracteres; la palabra se puede encontrar en un diccionario inglés típico. Dado que la palabra tiene menos de 16 caracteres (es decir, 128 bits), se agregan signos de almohadilla (#: el valor hexadecimal es 0x23) al final de la palabra para formar una clave de 128 bits.

Su objetivo es escribir un programa para averiguar la clave de cifrado. Puede descargar una lista de palabras en inglés de Internet. También hemos incluido uno en el archivo Labsetup.zip. El texto sin formato, el texto cifrado y IV se enumeran a continuación:

```
Texto sin formato (total 21 caracteres): Este es un alto secreto.
Texto cifrado (en formato hexadecimal): 764aa26b55a4da654df6b19e4bce00f4
                                        ed05e09346fb0e762583cb7da2ac93a2
IV (en formato hexadecimal):            aabbccddeeff00998877665544332211
```

Es necesario prestar atención a las siguientes cuestiones:

- Si elige almacenar el mensaje de texto sin formato en un archivo y enviar el archivo a su programa, debe verificar si la longitud del archivo es 21. Si escribe el mensaje en un editor de texto, debe tener en cuenta que algunos editores puede agregar un carácter especial al final del archivo. La forma más fácil de almacenar el mensaje en un archivo es usar el siguiente comando (el indicador -n le dice a echo que no agregue una nueva línea al final):

```
$ echo -n "Esto es un alto secreto". > archivo
```

- En esta tarea, se supone que debe escribir su propio programa para invocar la biblioteca criptográfica. No se otorgará ningún crédito si simplemente usa los comandos de openssl para realizar esta tarea. El código de muestra se puede encontrar en la siguiente URL:

```
https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html
```

- Cuando compile su código usando gcc, no olvide incluir el indicador -lcrypto, porque su el código necesita la biblioteca criptográfica. Vea el siguiente ejemplo:

```
$ gcc -o myenc myenc.c -lcrypto
```

Nota para los instructores. Alentamos a los instructores a generar su propio texto sin formato y texto cifrado utilizando una clave diferente; de esta forma los estudiantes no podrán obtener la respuesta de otro lugar o de cursos anteriores.
Los instructores pueden usar los siguientes comandos para lograr este objetivo (reemplace la palabra ejemplo con otra palabra secreta y agregue la cantidad correcta de signos # para que la longitud de la cadena sea 16):

```
$ echo -n "Esto es un alto secreto". > texto sin formato.txt $ echo -n
"ejemplo#########" > clave $ xxd -p clave 6578616d706c65232323232323232323
$ openssl enc -aes-128-cbc -e -in texto sin formato.txt -out textocifrado.bin \
-K 6578616d706c65232323232323232323 \ -iv
01020304050607080900a0b0c0d0e0f \ $ xxd -p textocifrado.bin
        e5accdb667e8e569b1b34f423508c154226311904ce9b625
```

10 Sumisión

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que son interesantes o sorprendentes.
Enumere también los fragmentos de código importantes seguidos de una explicación. Simplemente adjuntar el código sin ninguna explicación no recibirá créditos.

11 Reconocimiento

Queremos agradecer la contribución de las siguientes personas y organizaciones:

- Jiamin Shen desarrolló lo siguiente: el código que se ejecuta dentro del contenedor y la versión del contenedor de la tarea en predecible IV.
- La Fundación Nacional de Ciencias de EE. UU. proporcionó los fondos para el proyecto SEED de 2002 a 2020.
- La Universidad de Syracuse proporcionó los recursos para el proyecto SEED desde 2001 en adelante.