Writeup: Matrixfun II (Crypto)

Flag: `ENO{l1ne4r_alg3br4_i5_ev3rywh3re}`

Descripción del reto

El servidor nos da un cifrado que funciona así:

1. **Alfabeto:** Base64 estándar (65 caracteres: `a-z`, `A-Z`, `0-9`, `+`, `/`, `=`).
2. **Parámetros:** Matriz aleatoria (A \in \mathbb{Z}*{65}^{16 \times 16}) y vector (b \in *mathbb{Z}*{65}^{16}).
3. **Proceso:**
   - El mensaje se codifica en Base64.
   - Se rellena con `=` hasta que la longitud sea múltiplo de 16.
   - Se divide en bloques de 16 símbolos. Cada símbolo se sustituye por su índice en el alfabeto (0–64).
   - Para cada bloque (x \in \mathbb{Z}_{65}^{16}):
     [
     c = A \cdot x + b \pmod{65}
     ]

Al conectar, el servidor imprime el cifrado de la flag y luego actúa como **oráculo**: podemos enviar mensajes en hex y nos devuelve su cifrado con la misma (A) y (b).

Idea: es un cifrado afín por bloques

El cifrado es **afín** en cada bloque:

[
c = A,x + b \pmod{65}
]

Si recuperamos (A) y (b), podemos despejar:

[
x = A^{-1}(c - b) \pmod{65}
]

y así descifrar cualquier texto (y en particular la flag). La vulnerabilidad es que tenemos **oráculo de cifrado** con la misma clave.

Recuperar (b)

El vector (b) es el cifrado del bloque cuyos 16 índices son 0. Es decir, del bloque formado por 16 veces el primer carácter del alfabeto: `'a'` (índice 0).

Necesitamos enviar al servidor un mensaje en hex que, tras Base64, tenga como primer bloque exactamente 16 veces `'a'`.
Eso se consigue con un mensaje cuya codificación Base64 sea, por ejemplo, `"aaaaaaaaaaaaaaaa..."`.
En la solución se usa `block_from_symbols(b'a'*16)`: se decodifica `b'a'*16` desde Base64 (queda una cadena de bytes corta) y se vuelve a codificar para obtener exactamente el bloque canónico de 16 `'a'`. Esos bytes se envían en hex.

Al cifrar ese mensaje, el primer bloque de 16 símbolos es (x = (0,0,\ldots,0)), así que:

[
c_0 = A \cdot 0 + b = b
]

Las primeras 16 salidas del oráculo para ese mensaje son exactamente (b).

Recuperar (A) (columna a columna)

Para la columna (j) de (A), usamos el vector (e_j) (0 en todas las posiciones salvo un 1 en la posición (j)).
En el alfabeto: posición (j) con el carácter `'b'` (índice 1) y el resto con `'a'` (índice 0).

Enviamos un mensaje cuya Base64 tenga como primer bloque exactamente ese contenido (16 `'a'` salvo en la posición (j), donde va `'b'`). El oráculo devuelve:

[
c_j = A \cdot e_j + b = \text{(columna } j \text{ de } A) + b
]

Por tanto:

[
\text{columna } j \text{ de } A = c_j - b \pmod{65}

]

Haciendo esto para (j = 0, 1, \ldots, 15) obtenemos la matriz (A) completa. En total: 1 consulta para (b) y 16 para las columnas de (A) (17 consultas al oráculo).

Invertir (A) módulo 65

(\mathbb{Z}*{65}) *no es un cuerpo ((65 = 5 \times 13)), así que (A) puede no ser invertible. En la práctica, la (A) aleatoria del servidor suele ser invertible módulo 65. La solución usa* ***eliminación de Gauss–Jordan*** *en (\mathbb{Z}*{65}):

- En cada columna se busca un pivote cuyo valor sea coprimo con 65 (para poder calcular su inverso módulo 65).
- Se escala la fila para que el pivote sea 1 y se eliminan el resto de entradas de esa columna.
- Al final, la parte derecha de la matriz ampliada es (A^{-1} \pmod{65}).

Si en algún paso no hay pivote válido, (A) no sería invertible y habría que manejar el error (en este reto no fue necesario).

Descifrado

Con (A^{-1}) y (b):

1. Para cada bloque de 16 valores de cifrado (c):
   [
   x = A^{-1}(c - b) \pmod{65}
   ]
2. Cada componente de (x) es un índice en el alfabeto Base64; se convierte de nuevo a caracteres.
3. Se obtiene la cadena Base64 completa; se quita el padding de `=` al final (probando 0–15 caracteres si hace falta).
4. Se decodifica Base64 y se obtiene la flag en texto plano.

Resumen del ataque

Paso	Acción
1	Conectar y guardar el ciphertext de la flag del banner.
2	Enviar mensaje cuya Base64 empiece en 16× `'a'` → respuesta = (b).
3	Para (j = 0..15), enviar mensaje con bloque (e_j) (1 en posición (j)) → (c_j - b) = columna (j) de (A).
4	Calcular (A^{-1} \pmod{65}) con Gauss–Jordan.
5	Para cada bloque (c) del ciphertext: (x = A^{-1}(c - b)), mapear a Base64, decodificar y ajustar padding.

La flag es: **`ENO{l1ne4r_alg3br4_i5_ev3rywh3re}`**

Referencia rápida del código

- **`recover_Ab(oracle)`**: obtiene (b) con un bloque de 16 `'a'` y (A) con 16 bloques del tipo (e_j).
- **`mat_inv_mod(A, 65)`**: inversa de (A) en (\mathbb{Z}_{65}).
- **`decrypt(cipher_list, A, bvec)`**: aplica (x = A^{-1}(c - b)), convierte a Base64 y decodifica.

El mensaje que enviamos al oráculo debe ser el **bytes** que, al ser codificado en Base64 por el servidor, produzca exactamente el bloque de índices que queremos (por eso `block_from_symbols` decodifica y vuelve a codificar para usar la representación canónica).