INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

ALGEBRA LINEAL

Proyecto del curso

Elaboró: Ruiz Oropeza Emiliano Yahel

Bernal Jardon Saul Andres

Proyecto de Álgebra Lineal: Algoritmo de Compartición de Secretos (SSS)

En este proyecto, implementamos el algoritmo de compartición de secretos de Shamir (SSS) en MATLAB. Este algoritmo permite distribuir una contraseña entre varias partes de manera que sólo sea posible recuperarla si se reúnen suficientes fragmentos.

Introducción

El algoritmo de Shamir distribuye una contraseña (representada como un número entero) utilizando un polinomio aleatorio de grado k-1. Se generan coordenadas (i, p(i)) que representan fragmentos de la información, y la contraseña puede ser recuperada mediante interpolación de Lagrange si se reúnen al menos k fragmentos.

Objetivos

1. Generar un polinomio aleatorio para distribuir una contraseña.

2. Crear fragmentos en forma de coordenadas.

3. Recuperar la contraseña utilizando los fragmentos seleccionados.

Funciones del Proyecto

A continuación, definimos las funciones utilizadas en el proyecto:

1. `generar_polinomio`: Crea un polinomio aleatorio de grado k-1 con un término independiente que representa la contraseña.

2. `generar_coordenadas`: Genera las coordenadas (i, p(i)) a partir del polinomio.

3. `seleccionar_puntos`: Selecciona k fragmentos al azar.

4. `recuperar_contraseña`: Recupera la contraseña usando interpolación de Lagrange.

```
% Generar un polinomio aleatorio de grado k-1
function p = generar_polinomio(a0, k)
    coef = [a0, randi([1, 100], 1, k-1)]; % Coeficientes aleatorios
    p = @(x) polyval(fliplr(coef), x); % Crear polinomio
end

% Generar coordenadas (i, p(i))
function coords = generar_coordenadas(p, n)
    coords = [(1:n)', arrayfun(p, 1:n)']; % Coordenadas (i, p(i))
end

% Seleccionar k puntos aleatorios
function puntos = seleccionar_puntos(coords, k)
    idx = randperm(size(coords, 1), k); % Índices aleatorios
    puntos = coords(idx, :); % Seleccionar puntos
end

% Recuperar el polinomio y la contrasena usando interpolacion de Lagrange
function a0 = recuperar_contrasena(puntos)
    x = puntos(:, 1); % Coordenadas x
    y = puntos(:, 2); % Coordenadas y
    n = length(x);
    a0 = 0;
    for i = 1:n
        L = 1; % Base de Lagrange
        for j = [1:i-1, i+1:n]
            L = conv(L, [1, -x(j)]) / (x(i) - x(j));
        end
        a0 = a0 + y(i) * L(end); % Sumar al resultado final
    end
end
```

Ejemplo de Encriptación y Desencriptación

A continuación, mostramos un ejemplo práctico donde:

- k = 3 (mínimo de fragmentos necesarios).

- n = 9 (número total de fragmentos generados).

- Contraseña: 1103.

```
% Configuración inicial
k = 3; % Número mínimo de fragmentos necesarios
n = 9; % Número total de fragmentos generados
a0 = 1103; % Contraseña

% Encriptación
disp('--- ENCRIPTACIÓN ---');
p = generar_polinomio(a0, k); % Generar polinomio
coords = generar_coordenadas(p, n); % Generar coordenadas
disp('Coordenadas generadas:');
disp(coords);

% Seleccionar k puntos para desencriptar
disp('--- DESENCRIPTACIÓN ---');
puntos = seleccionar_puntos(coords, k);
disp('Puntos seleccionados:');
disp(puntos);

% Recuperar contraseña
a0_recuperada = recuperar_contrasena(puntos);
disp(['Contraseña recuperada: ', num2str(a0_recuperada)]);
```

```
--- ENCRIPTACIÓN ---
Coordenadas generadas:
           1        1247
           2        1543
           3        1991
           4        2591
           5        3343
           6        4247
           7        5303
           8        6511
           9        7871

--- DESENCRIPTACIÓN ---
Puntos seleccionados:
           9        7871
           3        1991
           6        4247


Contraseña recuperada: 1103
```

Resultados Esperados

El algoritmo debe generar un conjunto de coordenadas (i, p(i)) y permitir la recuperación de la contraseña original utilizando al menos k fragmentos seleccionados al azar. En este ejemplo:

- Contraseña original: 1103

- Contraseña recuperada: 1103