Relatório de Estudo: Criptografia Dinâmica com Função de Recorrência Linear

Paulo H. A. de Andrade[1]

[1]Departamento de Computação – Universidade Federal Rural do Semi-Árido (UFERSA)

paulo.andrade@alunos.ufersa.edu.br

1. Definição do problema

Na busca de aprimorar o algoritmo de Cifra de César, passado em sala de aula, aplicando funções polinomiais de segundo grau, viu-se que a dificuldade no processo de quebra da criptografia muda pouquíssimo, pois um caracter só terá um único equivalente estático.

Dessa forma, iniciou-se uma busca por uma estratégia mais segura, que dificulte a quebra de criptografia e aprimore ainda mais o desafio.

2. Estudo do problema

2.1. Definições matemáticas iniciais

A princípio, definiu-se a solução utilizando funções simples para melhorar a compreensão dos primeiros passos.

Considere X_n o equivalente numérico da letra na posição n, e as constantes a e b variáveis de input do usuário. Considere também a sequência definida por:

$$\begin{cases} C_n = aX_n + b & n = 0 \\ C_n = aX_n + b + C_{n-1} & x > 0 \end{cases}$$

Para as funções inversas usadas na descriptografia, onde X_n agora se refere ao equivalente numérico da letra criptografada, tem-se:

$$\begin{cases} D_n = \frac{X_n - b}{a} & n = 0 \\ D_n = \frac{X_n - b - D_{n-1}}{a} & x > 0 \end{cases}$$

3. Limitação da tabela de caracteres

Note que o algoritmo facilmente retorna um erro quando o usuário informa textos longos ou valores altos para os coeficientes da equação. O problema ocorre devido à limitação da tabela de caracteres, que por padrão do JavaScript é a UTF-8. A tabela, apesar de bem mais extensa que a ASCII ainda não é grande o suficiente, requerendo uma abordagem diferente para impedir o erro. Para melhor visualização do conflito, confira o exemplo da Tabela 1:

Função	Letra	Equivalente da letra	Criptografia
$314x + 15$	”d”	100	31415

Tabela 1. Exemplo de extrapolação da tabela de caracteres

A proposta deste algoritmo será de usar módulo de congruência para associar os valores extrapolantes a equivalentes dentro da limitação da tabela. Porém, apenas a aplicação da aritmética modular não será o suficiente, pois a função de criptografia deixaria de ser injetora.

Note que, durante a descriptografia, um valor dentro da faixa de caracteres pode ter vários equivalentes definidos pelo módulo de congruência, gerando uma dificuldade na escolha precisa de qual dos equivalentes é o original, como mostra a figura 1.

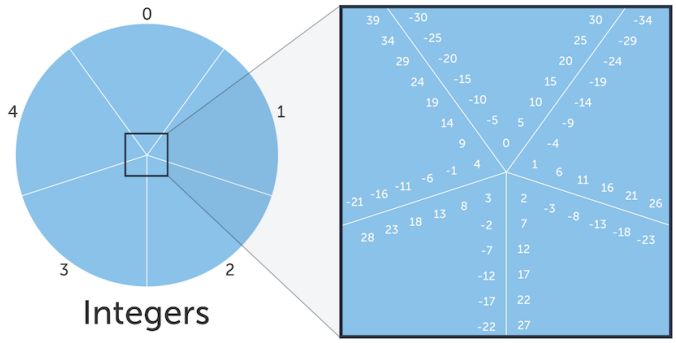

Figura 1. Módulo de congruência
Fonte: https://en.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/congruence-modulo

Surge então mais uma etapa da criptografia. Agora, além de gerar equivalentes para cada letra, o *output* irá dispor de um outro caractere, adjacente a cada símbolo criptografado, que indicará a parte inteira da divisão no cálculo do módulo de congruência.

Uma última correção será considerar apenas os caracteres válidos da tabela ASCII, evitando esbarrar em qualquer símbolo especial que prejudicará a formatação da resposta. Essa implementação será feita em conjunto com a aritmética modular, forçando os valores a começarem em 32 e se estendendo até 126.

Dessa forma, a nova expressão para a criptografia do caractere principal, usando as definições de a e b como coeficientes da função chave informada pelo usuário e X_n o equivalente numérico do caractere, será:

$$\begin{cases} C_n = [(aX_n + b) \mod 95] + 32 & n = 0 \\ C_n = [(aX_n + b + C_{n-1}) \mod 95] + 32 & x > 0 \end{cases}$$

E para os caracteres adjacentes teremos:

$$A_n = \lfloor \frac{C_n}{95} \rfloor + 32$$

No contexto da descriptografia deve-se, primeiramente, obter o valor original da criptografia através do caractere adjacente onde X_n se refere ao caractere a ser descriptografado e A_n ao seu adjacente:

$$O_n = (A_n - 32) * 95 + X_n$$

Tendo o valor original em mãos, a função inversa se define por:

$$\begin{cases} D_n = \frac{O_n - b}{a} & n = 0 \\ D_n = \frac{O_n - b - O_{n-1}}{a} & x > 0 \end{cases}$$

Resgatando o exemplo problemático definido na Tabela 1, a nova definição da solução trás uma satisfatória comparação de saídas apresentada na Tabela 2.

	Função	Letra	Equivalente da letra	Criptografia
Sem congruência	$314x + 15$	”d”	100	31415
Com congruência	$314x + 15$	”d”	100	97

Tabela 2. Comparação das soluções diante de uma extrapolação da tabela de caracteres

3.1. Implementação em código

Para realizar a implementação da solução, foi escolhida a linguagem JavaScript devido à sua simplicidade e facilidade de integração com interface, visto que nesta etapa ainda não há a preocupação em otimizações de performance.

A estratégia foi de criar duas funções: ”criptografar()”e ”descriptografar()”, para distribuir as responsabilidades de cada. Ambas usam laços de repetição *for* para percorrer a lista de caracteres, que serão submetidas às funções matemáticas corretas de acordo com a posição da letra em relação ao texto de entrada, como mostrado nas implementações abaixo.

```
function criptografar(texto, funcao) {
    let cript = "";

    for (let posicao = 0; posicao < texto.length; posicao++) {
        const letra = texto[posicao];
        if (posicao === 0) {
            const valor_bruto = funcao['a'] * letra.charCodeAt(0) + funcao['b'];
            cript += String.fromCharCode(Math.floor(valor_bruto / 95) + 32);
            cript += String.fromCharCode((valor_bruto % 95) + 32);
        } else {
            const valor_bruto = funcao['a'] * letra.charCodeAt(0) + funcao['b'] + cript.charCodeAt(2 * posicao - 1);
            cript += String.fromCharCode(Math.floor(valor_bruto / 95) + 32);  // Caractere adjacente
            cript += String.fromCharCode((valor_bruto % 95) + 32);
        }
    }

    return cript;
}
```

Listing 1. Versão linear da função criptografar()

```
function descriptografar(texto, funcao) {
    let decript = "";

    for (let posicao = 0; posicao < texto.length; posicao++) {
        const letra = texto[posicao];
        if (posicao === 0) {
            const valor_original = (letra.charCodeAt(0) - 32) * 95 + (
    texto.charCodeAt(posicao + 1) - 32);
            const traducao = Math.floor((valor_original - funcao['b'])
    / funcao['a']);
            decript += String.fromCharCode(traducao);
        } else {
            if (posicao % 2 === 0) {
                const valor_original = (letra.charCodeAt(0) - 32) * 95
    + (texto.charCodeAt(posicao + 1) - 32);
                const traducao = Math.floor((valor_original - funcao['b
    '] - texto.charCodeAt(posicao - 1)) / funcao['a']);
                decript += String.fromCharCode(traducao);
            }
        }
    }

    return decript;
}
```

Listing 2. Versão linear da função descriptografar()