TP2 - Exercício 1

Grupo 1

Diogo Coelho da Silva A100092

Pedro Miguel Ramôa Oliveira A97686

Problema proposto:

Considere a descrição da cifra A5/1 que consta no documento Lógica Computacional . Informação complementar pode ser obtida no artigo da Wikipedia.

Pretende-se:

1. Definir e codificar, em Z3 e usando o tipo BitVec para modelar a informação, uma FSM que descreva o gerador de chaves.
2. Considere as seguintes eventuais propriedades de erro:
   a. ocorrência de um "burst" (b^t que ocorre em 2^t passos ou menos.
   b. ocorrência de um "burst" de tamanho t que repete um "burst" anterior no mesmo output em $2^{t/2}$ passos ou menos.

Tente codificar estas propriedades e verificar se são acessíveis a partir de um estado inicial aleatoriamente gerado.

Proposta de resolução:

O código apresentado descreve um gerador de chaves do algoritmo de cifra A5/1, e tem como objetivo de modelar o comportamento de três dispositivos de deslocamento (LFSRs) e verificar propriedades de segurança no output gerado. Cada função específica dos LFSRs calcula o feedback com base em bits determinados do registo, utilizando operações como deslocamento lógico e extração de bits. A função que calcula o bit de maioria determina se os LFSRs devem ser atualizados com base na soma dos bits mais baixos de cada LFSR. A simulação, realizada ao longo de um número definido de passos, imprime o estado inicial e atual dos LFSRs, juntamente com o resultado do bit de maioria em cada iteração.

Além disso, o código inclui a deteção de propriedades, utilizando um solver Z3 para identificar a presença overflow de 0's. A execução da simulação gera estados iniciais aleatórios para os LFSRs e verifica se as propriedades de segurança podem ser alcançadas.

1. Importar as bibliotecas importantes

In [26]:
```
from z3 import *
import random
```

- `z3` : Importa a biblioteca Z3, um solver de satisfabilidade (SMT)
- `random` : Gera aleatoriedade para simular os eventos, como a seleção aleatória de estados ou a determinação de valores em cenários de simulação, proporcionando variabilidade nos resultados e comportamento do programa.

2. Define os dispositivos de deslocamento

As funções definidas implementam os passos de três registos de deslocamento (LFSRs) de acordo com o algoritmo da cifra A5/1. Cada função tem a seguinte finalidade:

- **`passoLfsr1(lfsr)`** : Calcula o feedback do primeiro LFSR com base nos bits nas posições 18, 17, 16 e 13. A função realiza um deslocamento lógico à direita (LSHR) do LFSR e insere o feedback na posição 18, retornando assim o novo estado do LFSR.
- **`passoLfsr2(lfsr)`** : Calcula o feedback do segundo LFSR utilizando os bits nas posições 21 e 20. Semelhante à função anterior, faz um deslocamento lógico à direita e atualiza a posição 21 com o feedback gerado, devolvendo o novo estado do LFSR.
- **`passoLfsr3(lfsr)`** : Obtém o feedback do terceiro LFSR a partir dos bits nas posições 22, 21, 20 e 7. A função também desloca o LFSR à direita e atualiza a posição 22 com o feedback, retornando o novo estado do LFSR.
- **`bitMaioria(x, y, z)`** : Calcula o bit de maioria entre os três LFSRs. Se pelo menos dois dos LFSRs têm um bit de saída igual a 1, a função retorna 1; caso contrário, retorna 0. Esta função é utilizada para determinar se os LFSRs devem ser atualizados com base no bit de maioria.

In [27]:
```
def calcFeedLfsr1(lfsr):
    #Calcula o bit de feedback usando uma operação XOR dos bits 18,17,19,13 e faz o mesmo para os outros
    feedback = (Extract(18, 18, lfsr) ^ Extract(17, 17, lfsr) ^
                Extract(16, 16, lfsr) ^ Extract(13, 13, lfsr))
    #LSHR é uma operação de que desloca o lfsr para a direita

    return LShR(lfsr, 1) | (feedback << 18)

def calcFeedLfsr2(lfsr):
    feedback = (Extract(21, 21, lfsr) ^ Extract(20, 20, lfsr))
    return LShR(lfsr, 1) | (feedback << 21)

def calcFeedLfsr3(lfsr):
    feedback = (Extract(22, 22, lfsr) ^ Extract(21, 21, lfsr) ^
                Extract(20, 20, lfsr) ^ Extract(7, 7, lfsr))
    return LShR(lfsr, 1) | (feedback << 22)

def bitMaioria(x, y, z):
#BitVecVal cria uma constante bit-vector cria 1 bit bit-vector com o valor 1
# Retorna 1 se maioria (2 ou mais) dos bits de entrada for 1, senão retorna 0
    return If((Extract(0, 0, x) + Extract(0, 0, y) + Extract(0, 0, z)) >= 2, BitVecVal(1, 1), BitVecVal(0, 1))
```

3. Definição da função de simulação da cifra A5\1

A função **`simulaA5_1(lfsr1Inicial, lfsr2Inicial, lfsr3Inicial, passos)`** simula o gerador de chaves do algoritmo A5/1, utilizando três registos de deslocamento (LFSRs). A função executa as seguintes operações:

1. **Inicialização**: Os estados iniciais dos LFSRs são definidos a partir dos parametros de entrada e um array vazio é criado para armazenar o fluxo de saída.
2. **Impressão do Estado Inicial**: A função imprime o estado inicial de cada LFSR antes de iniciar a simulação.
3. **Loop de Simulação**: Para cada passo no número de iterações especificado:
   - **Cálculo do Bit de Maioria**: A função `bitMaioria` é chamada para determinar o bit de maioria entre os três LFSRs. Este bit é adicionado ao fluxo de saída.
   - **Atualização dos LSFRs**: Se o bit de maioria é igual a 1, os três LFSRs são atualizados utilizando as funções `calcFeedLfsr1`, `calcFeedLfsr2` e `calcFeedLfsr3`.
   - **Avaliação e Impressão dos Valores**: Os valores do bit de maioria e da saída são simplificados e impressos junto com os estados atuais dos LFSRs.
4. **Retorno do Fluxo de Saída**: Após completar todos os passos, a função retorna o fluxo de saída gerado durante a simulação.

Esta função é essencial para observar como os estados dos LFSRs evoluem ao longo do tempo e como o fluxo de saída é gerado com base no bit de maioria.

In [28]:
```
def simulaA5_1(lfsr1Inicial, lfsr2Inicial, lfsr3Inicial, passos):
    # Inicialização dos registos com os valores iniciais
    lfsr1 = lfsr1Inicial
    lfsr2 = lfsr2Inicial
    lfsr3 = lfsr3Inicial
    output = [] # Lista para armazenar os bits de saída

    # Imprime o estado inicial dos três registos
    print(f"Estado inicial:\n LFSR1: {lfsr1}\n LFSR2: {lfsr2}\n LFSR3: {lfsr3}\n")

    # Executa a simulação pelo número de passos especificado
    for passo in range(passos):
        # Calcula o bit de maioria dos três registos
        maioria = bitMaioria(lfsr1, lfsr2, lfsr3)
        output.append(maioria) # Adiciona o bit de maioria à saída

        # Atualiza os registos apenas se o bit de maioria for 1
        if maioria == BitVecVal(1, 1):
            lfsr1 = calcFeedLfsr1(lfsr1)
            lfsr2 = calcFeedLfsr2(lfsr2)
            lfsr3 = calcFeedLfsr3(lfsr3)

        # Simplifica os valores para impressão
        #A função simplify definida no z3 simplica as expressões simbolicas... converte as expressões bit-vector em valores mais legiveis
        valorMaioria = simplify(maioria)
        valorSaida = simplify(output[-1])

        # Imprime o estado atual de cada registo, bit de maioria e saída
        print(f"Passo {passo + 1}:")
        print(f" LFSR1: {lfsr1} | LFSR2: {lfsr2} | LFSR3: {lfsr3} | Maioria: {valorMaioria} | Saída: {valorSaida}\n")

    return output # Devolve a sequência de bits gerada
```

4. Restrições do burst

As funções **`detectarBurstoDeZeros(solver, fluxoSaida, t)`** e **`detectarBurstoRepetido(solver, fluxoSaida, t)`** têm como objetivo verificar propriedades de segurança no fluxo de saída gerado pelo algoritmo A5/1, focando na deteção de padrões que podem comprometer a aleatoriedade e a robustez da chave gerada.

- **`detectarBurstoDeZeros(solver, fluxoSaida, t)`** : Esta função é responsável por identificar a ocorrência de um "burst" de zeros, que consiste em uma sequência contínua de t bits com valor 0. Para cada posição no fluxo de saída, a função verifica se há uma sequência de t zeros consecutivos. Além disso, a função garante que essa sequência de zeros ocorra em até 2^t passos a partir do início da sequência, adicionando as restrições necessárias ao solver. Essa propriedade é crucial, pois uma sequência longa de zeros pode indicar fraquezas no gerador de chaves, permitindo previsibilidade na saída.

- **`detectarBurstoRepetido(solver, fluxoSaida, t)`** : Esta função procura identificar se há repetições de um "burst" de tamanho t que ocorrem em momentos diferentes no mesmo fluxo de saída. A verificação assegura que a repetição do burst anterior aconteça em um intervalo de até 2^t passos. Se as condições forem satisfeitas, as restrições são adicionadas ao solver para futura verificação. A deteção de repetições de bursts é importante, pois pode sugerir que o gerador não está produzindo uma sequência suficientemente aleatória, o que poderia facilitar a quebra da cifra.

Essas funções ajudam a garantir que o fluxo de saída do gerador de chaves A5/1 não apresenta padrões previsíveis que poderiam comprometer a segurança da cifra. Assim, são consideradas as seguintes propriedades de erro:

1. A ocorrência de um "burst" (b^t (ou seja, t zeros) que ocorre em 2^t passos ou menos, indicando um potencial problema de segurança.
2. A ocorrência de um "burst" de tamanho t que repete um "burst" anterior no mesmo output em 2^t passos ou menos, que poderia permitir a um atacante prever partes da chave gerada.

A implementação destas verificações permite uma análise formal da segurança do gerador de chaves, contribuindo para a robustez do sistema de criptografia.

In [29]:
```
def detectaBurstDeZeros(solver, fluxoSaida, t):
    # Percorre o fluxo de saída procurando sequências de t zeros consecutivos
    for i in range(len(fluxoSaida) - t + 1):
        # Cria uma expressão que verifica se existe uma sequência de t zeros
        burstZero = And([fluxoSaida[i + j] == BitVecVal(0, 1) for j in range(t)])
        # Garante que a sequência ocorre dentro de 2^t passos
        for j in range(i + 1, min(i + 2**t, len(fluxoSaida) - t + 1)):
            solver.add(burstZero)

def detectaBurstRepetido(solver, fluxoSaida, t):
    # Percorre o fluxo de saída procurando padrões repetidos de tamanho t
    for i in range(len(fluxoSaida) - t + 1):
        # Extrai o primeiro padrão (burst1)
        burst1 = fluxoSaida[i:i + t]
        # Procura um segundo padrão idêntico
        for j in range(i + 1, len(fluxoSaida) - t + 1):
            burst2 = fluxoSaida[j:j + t]
            # Verifica se os padrões são iguais e ocorrem dentro de 2^(t/2) passos
            if j - i <= 2**(t // 2):
                solver.add(burst1 == burst2)
```

5. Simulação e apresentação de resultados

Esta função executa uma simulação da cifra A5/1, utilizando registradores de deslocamento linear (LFSRs). A cada execução, são gerados estados iniciais aleatórios para os LFSRs, o que resulta em diferentes fluxos de saída.

Parâmetros

- **`t`**: Um inteiro que representa o tamanho do burst a ser verificado.
- **`passos`**: Um inteiro que indica o número de passos na simulação.

Funcionamento

1. **Criar um Solver**: Um solver do Z3 é criado para ajudar na verificação de propriedades.
2. **Gerar Estados Iniciais Aleatórios**:
   - Três estados iniciais aleatórios são gerados para os LFSRs:
     - `lfsr1Inicial` : um número aleatório de 19 bits.
     - `lfsr2Inicial` : um número aleatório de 22 bits.
     - `lfsr3Inicial` : um número aleatório de 23 bits.
3. **Gerar Fluxo de Saída**: O fluxo de saída é gerado a partir dos estados iniciais aleatórios usando a função `simularA5_1` .
4. **Verificar Propriedades**:
   - A função verifica se o fluxo contém um burst de zeros utilizando `detectarBurstoDeZeros` .
   - A função também verifica se há um burst repetido com `detectarBurstoRepetido` .
5. **Verificação de Satisfatibilidade**: O solver é usado para verificar se as propriedades especificadas são satisfatórias.
   - Se satisfatórias, imprime que as propriedades são atingíveis a partir do estado inicial aleatório.
   - Caso contrário, imprime que as propriedades não são atingíveis.

In [30]:
```
def executaSimulacaoA5_1(t, passos):
    # Cria um objeto solver para verificar as propriedades
    solver = Solver()

    # Gera estados iniciais aleatórios para os três registos LFSR
    # LFSR1: 19 bits, LFSR2: 22 bits, LFSR3: 23 bits
    lfsr1Inicial = BitVecVal(random.randint(0, 2**19 - 1), 19)
    lfsr2Inicial = BitVecVal(random.randint(0, 2**22 - 1), 22)
    lfsr3Inicial = BitVecVal(random.randint(0, 2**23 - 1), 23)

    # Gera a sequência de saída usando os estados iniciais
    output = simulaA5_1(lfsr1Inicial, lfsr2Inicial, lfsr3Inicial, passos)

    # Verifica a propriedade de sequência de zeros
    detectaBurstDeZeros(solver, output, t)

    # Verifica a propriedade de padrões repetidos
    detectaBurstRepetido(solver, output, t)

    # Verifica se as propriedades são satisfeitas com os estados iniciais dados
    if solver.check() == sat:
        print("Propriedades atingíveis a partir do estado inicial aleatório!")
    else:
        print("Propriedades não atingíveis a partir do estado inicial aleatório.")

# Example execution
t = 3  # Tamanho da sequência a verificar
passos = 16  # Número de passos da simulação
executaSimulacaoA5_1(t, passos)
```

```
Estado inicial:
 LFSR1: 199543
 LFSR2: 1366995
 LFSR3: 4308725

Passo 1:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 2:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 3:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 4:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 5:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 6:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 7:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 8:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 9:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 10:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 11:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 12:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 13:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 14:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 15:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Passo 16:
 LFSR1: 199543 | LFSR2: 1366995 | LFSR3: 4308725 | Maioria: 0 | Saída: 0

Propriedades atingíveis a partir do estado inicial aleatório!
```