Trabalho 1

Algoritmo S-AES

Encrypt

A função abaixo é a função escrita para encriptar um inteiro de 16 bits (plaintext) com uma chave também de 16 bits usando o algoritmo S-AES.
Como é possível ver, o processo é divido em múltiplas sub-rotinas que modificam um estado, que é uma matriz de 4 nibbles.

```
std::uint16_t saes_encrypt(uint16_t plainText, uint16_t key) {
  std::vector<std::vector<uint8_t> > state = convert_int_to_state_matrix(plainText);

  std::vector<uint16_t> keys = saes_key_expansion(key);

  state = saes_add_round_key(state, keys[0]);

  // Primeira rodada
  state = saes_nibble_substitution(state, Sbox);
  state = saes_shift_rows(state);
  state = saes_mix_columns(state);
  state = saes_add_round_key(state, keys[1]);

  //Segunda Rodada
  state = saes_nibble_substitution(state, Sbox);
  state = saes_shift_rows(state);
  state = saes_add_round_key(state, keys[2]);

  return result;
}
```

Expansão da chave

O processo de expansão da chave tem como objetivo gerar 3 chaves de 16 bits a partir de uma única chave de 16 bits que posteriormente vão ser utilizadas no processo de encriptação.
Como é possível ver no código abaixo a primeira chave é a própria chave passada por argumento, posteriormente para irmos criando as outras chaves o algoritmo trabalha dividindo as chaves em duas partes de 8 bits, chamados aqui de `wn`.
Para gerar os primeiros 8 bits da segunda chave, o `w2`, usamos o `w0` (8 bits mais significativos da chave original) e aplicamos algumas operações nele, a primeira sendo um `xor` com uma constante `RCON` (round constant) que é um valor fixo para a primeira rodada de geração de chave, feito isso realizaremos um outro `xor`, mas agora com o resultado da função `key_expansion_subnibble(rotate_nibble(w1))` que consiste primeiro em rotacionar os 4 primeiros bits de `w1` com os 4 últimos, e aplicar a substituição com a `SBOX`, que é uma tabela do S-AES utilizada para substituir valores de 4 bits por outros pré definidos.
Tendo o `w2` em mãos basta realiza a operação `w2 xor w1` para obter o valor de `w3` e assim gerar a segunda chave.
E para obter a terceira chave basta repetir o processo realizado para gerar a segunda chave, porém ao invés de utilizar a chave original para iniciar o processo, é utilizada a chave 2, e a constante `RCON` também deve ser alterada para a devida constante da segunda rodada.

```
#define NIBBLE_MASK 0x0F
uint8_t Sbox[16] =
{
    0x9,0x4,0xA,0xB,
```

```
    0xD,0x1,0x8,0x5,
    0x6,0x2,0x0,0x3,
    0xC,0xE,0xF,0x7
};
uint8_t RCON[2] = { 0x80, 0x30 };

uint8_t rotate_nibble(uint8_t word) {
  uint8_t upper = word << 4;
  uint8_t lower = (word >> 4) & NIBBLE_MASK;
  return upper | lower;
}

uint8_t key_expansion_subnibble(uint8_t word) {
  uint8_t upper = (word >> 4) & NIBBLE_MASK;
  uint8_t lower = word & NIBBLE_MASK;
  return Sbox[upper] << 4 | Sbox[lower];
}

std::vector<uint16_t> saes_key_expansion(uint16_t key) {
  std::vector<uint16_t> keys (3, 0);
  keys[0] = key;

  uint8_t w0 = (keys[0] >> 8) & 0xFF;
  uint8_t w1 = keys[0] & 0xFF;
  uint8_t w2 = w0 ^ RCON[0] ^ key_expansion_subnibble(rotate_nibble(w1));
  uint8_t w3 = w2 ^ w1;

  keys[1] = w2 << 8 | w3;

  uint8_t w4 = w2 ^ RCON[1] ^ key_expansion_subnibble(rotate_nibble(w3));
  uint8_t w5 = w4 ^ w3;

  keys[2] = w4 << 8 | w5;

  return keys;
}
```

Adicionar chave da rodada

O processo de adicionar chave da rodada é bem simples, basta realizarmos um `xor` do estado atual do S-AES com a chave da rodada correspondente.
Ele é feito 3 vezes durante o S-AES, no começo do algoritmo e no fim de cada uma das duas rodadas.

```
std::vector<std::vector<uint8_t> > saes_add_round_key(std::vector<std::vector<uint8_t> > currentState, uint16_t
key) {
  uint16_t currentStateNumber = convert_state_matrix_to_int(currentState);

  uint16_t newStateNumber = currentStateNumber ^ key;

  return convert_int_to_state_matrix(newStateNumber);
}
```

Substituição de Nibbles

O processo de substituição de nibbles funciona pegando o estado, que é uma matriz de 4 nibbles, e trocando esses valores por valores tabelados pela `Sbox`.
Esse processo ocorre nas duas rodadas.

```
std::vector<std::vector<uint8_t> > saes_nibble_substitution(std::vector<std::vector<uint8_t> > currentState,
uint8_t* Sbox) {
  std::vector<std::vector<uint8_t> > result(2, std::vector<uint8_t>(2, 0));
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      result[i][j] = Sbox[currentState[i][j]];
    }
  }

  return result;
}
```

Troca de linhas

O processo de troca de linhas é um processo simples que pega o estado do S-AES e troca as duas nibbles da segunda linha da matriz de lugar.
Esse processo ocorre em cada rodada depois da substituição de nibbles.

```
std::vector<std::vector<uint8_t> > saes_shift_rows(std::vector<std::vector<uint8_t> > currentState) {
  std::vector<std::vector<uint8_t> > result(2, std::vector<uint8_t>(2, 0));

  // primeira linha igual
  result[0][0] = currentState[0][0];
  result[0][1] = currentState[0][1];

  // inverte a segunda linha
  result[1][0] = currentState[1][1];
  result[1][1] = currentState[1][0];

  return result;
}
```

Mistura de colunas

O processo de mistura de colunas é o processo mais complexo do algoritmo S-AES, ele consiste em multiplicar o estado por uma outra matriz:

```
|1|4|
|4|1|
```

Porém as operações são feitas campo de Galois e devem ser reduzidas módulo de $x^4 + x + 1$, para a implementação dessa etapa foi utilizado um trecho de código copiado de outro repositório público que já o implementava https://github.com/mostsfamahmoud/Simplified-AES.
Essa operação é realizada na primeira rodada, depois da troca de linhas.

```
#define GF_MUL_PRECOMPUTED_TERM    0x03

uint8_t GF_MultiplyBy(uint8_t data, uint8_t mulValue) {
  uint8_t result = data;

  switch (mulValue) {
    case 2: /* Used in SAES Decryption */
      if ((result >> 3) == 1) {
        result = ((result << 1) & 0x0F) ^ GF_MUL_PRECOMPUTED_TERM;
      } else {
        result = (result << 1) & 0x0F;
      }
```

```
      break;
    case 4: /* Used in SAES Encryption */
      for (int i = 0; i < 2; i++) {
        if ((result >> 3) == 1) {
          result = ((result << 1) & 0x0F) ^ GF_MUL_PRECOMPUTED_TERM;
        } else {
          result = (result << 1) & 0x0F;
        }
      }
      break;
    case 9: /* Used in SAES Decryption */
      for (int i = 0; i < 3; i++) {
        if ((result >> 3) == 1) {
          result = ((result << 1) & 0x0F) ^ GF_MUL_PRECOMPUTED_TERM;
        } else {
          result = (result << 1) & 0x0F;
        }
      }
      result = result ^ data;
      break;
    default:
      break;
  }
  return result;
}

std::vector<std::vector<uint8_t> > saes_mix_columns(std::vector<std::vector<uint8_t> > currentState) {
  std::vector<std::vector<uint8_t> > result(2, std::vector<uint8_t>(2, 0));

  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      result[i][j] = currentState[i][j] ^ GF_MultiplyBy(currentState[(i + 1) % 2][j], 4);
    }
  }

  return result;
}
```

Decrypt

O processo de decriptar o texto cifrado é bem parecido com o processo de encriptação e segue o caminho inverso dele, ou seja, aplica as mesmas operações só que na ordem inversa.
Os únicos processo que mudam são a substituição de nibbles e a mistura de colunas.
A substuição de nibbles passa a usar a `InverseSbox`.
E a mistura de colunas realizar a multiplicação do estado com a matriz:

```
|9|2|
|2|9|
```

```
uint8_t InverseSbox[16] =
{
    0xA,0x5,0x9,0xB,
    0x1,0x7,0x8,0xF,
    0x6,0x0,0x2,0x3,
    0xC,0x4,0xD,0xE
};

std::vector<std::vector<uint8_t> > saes_inverse_mix_columns(std::vector<std::vector<uint8_t> > currentState) {
  std::vector<std::vector<uint8_t> > result(2, std::vector<uint8_t>(2, 0));
```

```
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      result[i][j] = GF_MultiplyBy(currentState[i][j], 9) ^ GF_MultiplyBy(currentState[(i + 1) % 2][j], 2);
    }
  }

  return result;
}

uint16_t saes_decrypt(uint16_t cipherText, uint16_t key) {
  std::vector<std::vector<uint8_t> > state = convert_int_to_state_matrix(cipherText);

  std::vector<uint16_t> keys = saes_key_expansion(key);

  state = saes_add_round_key(state, keys[2]);

  // Primeira rodada
  state = saes_shift_rows(state);
  state = saes_nibble_substitution(state, InverseSbox);
  state = saes_add_round_key(state, keys[1]);

  // Segunda rodada
  state = saes_inverse_mix_columns(state);
  state = saes_shift_rows(state);
  state = saes_nibble_substitution(state, InverseSbox);
  state = saes_add_round_key(state, keys[0]);

  uint16_t result = convert_state_matrix_to_int(state);

  return result;
}
```

Rodando o S-AES no terminal

Ao rodar o S-AES com a string "ok" e a chave "oe" temos os seguintes resultados:

- **Encrypt**

```
Bem vindo ao programa SAES do vini
Por favor selecione uma operação
1. Testar o programa SAES
2. Testar o programa SAES_ECB
3. Compare os diferentes modos de opereração do AES
4. Sair
1

Escolha a operação:
1 - Encrypt
2 - Decrypt
1
Digite a string base64 a ser encpriptada (até 16bits): b2s=
Digite a chave de 16bits em base64: b2U=
Expansão da chave:
Chave 1: 0×6f65
Chave 2: 0×f792
Chave 3: 0×65f7
Estado após a adição da chave 1: 0×e
Estado após a substituição de nibbles: 0×999f
Estado após a troca de linhas 1: 0×9f99
Estado após a mistura de colunas: 0×dbb
Estado após a adição da chave 2: 0×fa29
Estado após a substituição de nibbles: 0×70a2
Estado após a troca de linhas 2: 0×72a0
Estado após a adição da chave 3: 0×1757
Texto cifrado em base64: F1c=
```

- **Decrypt**

```
Bem vindo ao programa SAES do vini
Por favor selecione uma operação
1. Testar o programa SAES
2. Testar o programa SAES_ECB
3. Compare os diferentes modos de opereração do AES
4. Sair
1

Escolha a operação:
1 - Encrypt
2 - Decrypt
2
Digite a string base64 a ser decriptada (até 16bits): F1c=
Digite a chave de 16bits em base64: b2U=
Expansão da chave:
Chave 1: 0×6f65
Chave 2: 0×f792
Chave 3: 0×65f7
Estado após a adição da chave 3: 0×72a0
Estado após a troca de linhas 1: 0×70a2
Estado após a substituição de nibbles com a sbox invertida: 0×fa29
Estado após a adição da chave 2: 0×dbb
Estado após a mistura de colunas inversa: 0×9f99
Estado após a troca de linhas 2: 0×999f
Estado após a substituição de nibbles com a sbox invertida: 0×e
Estado após a adição da chave 1: 0×6f6b
Texto em claro em base64: b2s=
```

Algoritmo S-AES Modo ECB

Algoritmos de cifra de bloco como O AES e o S-AES podem operar em diferentes modos e esses modos são referentes a como o algoritmo vai processar os múltiplos blocos de informação.
O modo implementado nesse trabalho foi o ECB (Elastic CodeBlock), ele funciona de forma bem simples, o

texto em claro é divido em blocos de 16 bits e esse blocos então são encriptados separadamente e no final são concatenados de volta em sequência para formar o texto cifrado.

```
std::string ecb_saes_decrypt(std::string base64Input, std::string key) {
  // Decode the input
  std::vector<BYTE> decodedInput = base64_decode(base64Input);

  std::vector<uint16_t> inputWords;

  // transform the input into 16-bit words
  for (size_t i = 0; i < decodedInput.size() - 1; i += 2) {
    char firstChar = decodedInput[i];
    char secondChar = decodedInput[i + 1];

    uint16_t word = (uint8_t) firstChar << 8 | (uint8_t) secondChar;

    inputWords.push_back(word);
  }

  // Decode the key
  std::vector<BYTE> decodedKey = base64_decode(key);

  // Check if the key is 16 bits
  if (decodedKey.size() > 2) {
    std::cout << "A chave deve ter 16 bits" << std::endl;
    return "";
  }

  // Transform the key into a 16-bit word
  uint16_t keyWord = 0;
  for (size_t i = 0; i < decodedKey.size(); i++) {
    keyWord = (keyWord << 8) | decodedKey[i];
  }

  std::vector<uint16_t> outputWords(inputWords.size());

  for (size_t i = 0; i < inputWords.size(); i++) {
    outputWords[i] = saes_decrypt(inputWords[i], keyWord);
  }

  // base64_encode(outputWords);
  std::string outputStr;
  for (size_t i = 0; i < outputWords.size(); i++) {
    char firstChar = (char) (outputWords[i] >> 8) & 0xFF;
    char secondChar = (char) (outputWords[i] & 0xFF);

    outputStr += firstChar;
    outputStr += secondChar;
  }

  // Encode the output string to base64
  std::string output = base64_encode((unsigned char*) outputStr.c_str(), outputStr.size());

  return output;
}
```

Execução

```
Bem vindo ao programa SAES do vini
Por favor selecione uma operação
1. Testar o programa SAES
2. Testar o programa SAES_ECB
3. Compare os diferentes modos de opereração do AES
4. Sair
2

Escolha a operação:
1 - Encrypt
2 - Decrypt
1
Digite a string base64 a ser encpriptada: TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQuIExvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0LiAKCg==
Digite a chave de 16bits em base64: b2U=
Texto cifrado em base64: dvVKGcf6×8pLmcf67jbm9hvWmeEcloLccRPgonb1ShnH+sfKS5nH+u425vYb1pnhHJaC3HET4KIE7w==
```

Para testar a execução do programa foi encriptada a string `"Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet. "` que em base64 é `TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQuIExvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0LiAKCg==`, usando a chave `b2U=`, obtivemos a string base64 `dvVKGcf6x8pLmcf67jbm9hvWmeEcloLccRPgonb1ShnH+sfKS5nH+u425vYb1pnhHJaC3HET4KIE7w==`, que em hexadecimal é igual a `76f54a19c7fac7ca4b99c7faee36e6f61bd699e11c9682dc7113e0a276f54a19c7fac7ca4b99c7faee36e6f61bd699e11c9682dc7113e0a204ef`.
Analisando esse texto cifrado em hexadecimal conseguimos observar o maior problema do modo ECB, blocos iguais resultam em textos cifrados iguais, o que pode levar ao reconhecimento de padrões em um texto, ou por exemplo se formos encriptar imagens, as imagens continuam com o mesmo contorno de objetos por exemplo.

Modos do AES

Usando a biblioteca Cryptopp iremos comparar os diferentes modos de execução do AES, ao rodar um script que encripta uma mesma mensagem nos diferentes modos mostrando sua saída e posteriormente mostra a média do tempo de execução ao rodar cada script 100 vezes.
A média do tempo de execução ficou como na tabela abaixo:

Modo	Tempo
ECB	71ms
CBC	75ms
CFB	75ms
OFB	76ms
CTR	72ms

Podemos ver que mesmo com modos de operação diferentes os resultados foram basicamente iguais, isso ocorre pois provavelmente a biblioteca utilizada não otimiza o modo ECB ou CTR para usarem paralelismo.
Do ponto de vista da segurança é sabido que modos como CBC, CFB e OFB são mais seguros que o ECB pois nesses modos cada bloco é encriptado levando em conta o contexto de todos os blocos que vieram anteriormente.