Initialisierung von Salsa20

Yusuf Enes Tamer
HTW-Aalen Cryptography Engineering

Abstract

Bei der Salsa20 handelt es sich um eine einfache Stromchiffre, was von moderen Prozessoren ausgeführt werden kann, um diese in diversen Datenkommunikationsprotokollen zu verwenden. Diese Chiffre wurde bereits im Jahr 2005 bekannt. Diese Chiffre hat 4 Eingaben.

- Plaintext
- Nonce
- Key
- Counter

1 C-Programmierung

Ziel ist es einen Algorithmus zu Programmieren, das 32-Bit-Integer auf eine Matrix 4x4 lagert bzw. strukturiert. Ein Feld in der Matrix hat einen Speicher von 32-Bit. Für den C-Programmierer müssen mehrere Konvertierungsvorgäng vorhanden sein zwischen, **INT8- INT32-** und **INT64**, da diese Integer beim Einlesen der Daten verwendet werden.
Der Datenstrom bzw. einzelne Blöcke sind 32-Bits groß, somit ist das die Kerneigenschaft von Salsa20-Chiffre.
Die Konvertierung zwischen den beiden Datentypen **uint64_t** und **uint32_t** erfolgt über den folgenden Algorithmus:

1. Zuerst muss das 64-Bit-Integer pro Byte zerlegt werden und damit 8-Bits eingelesen.
2. Beim Einlesen wird eine Schiebe-Operation sowie auch eine AND- und ODER-Maskierung verwendet.
3. Nach dem das Byte selektiert wurde, wird dieser in einer Queue gespeichert.
4. Aus der Queue werden die Bytes rausgelesen und diese dann partiell in einem 32-Bit-Datentyp Speicher geladen

Nach der Konvertierung können die Daten direkt in die 4x4 Matrix geladen werden, sodass es sich um eine Chiffre handelt, die aus 512-Bits besteht.

2 Quellcode

Hierbei handelt es sich um das Hauptprogramm, mit der zuerst ein Schlüsselvektor, daraufhin eine Konstante, weiterhin ein anderer Vektor, was den Plaintext beinhaltet, initialisiert. Anschließend wird dies in eine 4x4 Matrix geladen.

```
int main(char argv[], int argc)
{
     initializeKey();
     initializeConstant(c);
     initializeVector(v);
     initializeQuadraticRound(k, c, v, t);
     printField();
}
```

2.1 Initialisierung der Schlüsselvektoren

Bei der ersten Funktion findet eine initialisierung von den einzelnen Schlüsselvektoren statt. Der C-Code sieht folgendermaßen aus:

```
void initializeKey()
{
```

```
  uint64_t  K1 =  0x5857565557565554;
  uint64_t  K2 =  0x5655545355545352;
  uint64_t  K3 =  0x5453525153525150;
  uint64_t  K4 =  0x5251504951504948;

  for(int i = 0; i<8; i++)
k[i] = 0x00;

uint8_t temp  = 0x00;
for(int i = 0; i<32; i++)
{
if( i  < 8)
{
  temp  = K1 >> (i * 8)  & 0x00FF;
  bytes.push(temp);
  continue;
}
else if( i  >=8 && i<=15)
{ temp  = K2 >> ( (i *8) & 0x00FF);
  bytes.push(temp);
continue;
}
else if( i >= 16 && i<=23)
{
temp =  K3 >> ( (i * 8) & 0x00FF);
bytes.push(temp);
continue;
}
else if( i >= 24 && i<=31)
{
temp =  K4 >> ( (i  *8) & 0x00FF);
bytes.push(temp);
continue;
}
}
for(int i = 0; i<32; i++)
{
  uint8_t temp = bytes.front();
  k[i /4] =  (k[ i / 4  ] << 8) |  temp;
  bytes.pop();
}
}
```

2.1.1 Initialisierung der 4x4 Matrix

```
void initializeQuadraticRound(uint32_t * k, uint32_t * c,
{
 S[0][1] = k[0];
 S[0][2] = k[1];
 S[0][3] = k[2];
 S[1][0] = k[3];
 S[2][3] = k[4];
 S[3][0] = k[5];
 S[3][1] = k[6];
 S[3][2] = k[7];


 S[0][0] = c[0];
 S[1][1] = c[1];
 S[2][2] = c[2];
 S[3][3] = c[3];


 S[2][0] = t[0];
 S[2][1] = t[1];

 S[1][2] = v[0];
 S[1][3] = v[1];
  return;
}
```

3 Anmerkung

Bei der Parameterübergabe handelt es sich um Call-By-Reference, weshalb die vorhandenen Arrays über einen Funktionsparameter geladen werden, damit die Rundentransformation stattfinden kann.
Folgendermaßen wird die 4x4 Matrix angeordnet !

$$\begin{bmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{bmatrix} \tag{1}$$

Figure 1: .