Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Sicherheit in der Informationstechnik

Sichere Implementierungen kryptographisher Verfahren WS 2015-2016 Übungsbericht:

Implementierung des Advanced Encryption Standards (AES) Algorithmus auf dem ATmega644V Mikrokontroller

Author: Zhiwei Han
E-mail: hanzw356255531@icloud.de
LRZ Kennung: ga63vuf
Matrikelnummer: 03672554

1 Beschreibung der AES Implementierung

1.1 S-box

Erklären Sie ausführlich Ihre Implementierung der S-box. Beschreiben Sie Ihre Strategie und nennen Sie ggf. Beispiele (Pseudocode).

Das Ausgangstext des Zwischenzustands nach SubByte in diesem Programm wird nicht durch eine on-the-fly Algorithmus durchgeführt, sondern durch Suchen einer sogenannten Lookup-Tabelle, um die Berechnungszeit zu verkürzen.

Die S-Box-Tabelle wird voraussichtlich durch das Schlusswort "PROGMEM" in Programmspeicher gespeichert. Während des S-Box Prozess wird jede eigene Wert des Eingangstexts im Speicher durchgesucht. Und dann geht das Programm in den nächsten Schritt ein.

Pseudocode:

Algorithm 1 SBox Implementation

Input: *acht Bit Wert*

Output: *Das durch SubByte Ergebnis*

$const\ Array\ SBox[16][16] \leftarrow Die\ 256\ Elemente\ in\ Lookup-Tabelle$

.....

.....

.....

$SubByte(P0) \leftarrow SBox[RowNum][ColumnNum]$

1.2 MixColumns

Erklären Sie ausführlich Ihre Implementierung der MixColumns und nennen Sie ggf. Beispiele (Pseudocode).

Als den komplexesten Teil der Algorithmus handelt sich MixColumns hier um eine Lookup-Strategie.

1. In dieser Implementation werden andere zwei Lookup-Tabellen für Multiplikation auch gespeichert. Jeder erhält 256 Elemente(ein Byte) in ihren einzelnem Fall , also wenn das Polynom mit X oder (X+1) multipliziert.

2. Die Reihenfolge der XOR Berechnung laut des MCB Matrix wird jedoch vorgerechnet und in der Wertzuordnung eingesetzt. Die XOR Berechnung wird nach der Suche der Tabellen eingeführt.

Pseudocode:

Algorithm 2 MixColumns Implementation

Input: 128 *Bit Wert*

Output: *Das durch Mixcolumns Ergebnis*

$const\ Array\ Multi2[256] \leftarrow Polynome * X$
$const\ Array\ Multi3[256] \leftarrow Polynome * (X + 1)$
.....
$Temp0 \leftarrow P0$
$Temp1 \leftarrow P1$
$Temp2 \leftarrow P2$
$Temp3 \leftarrow P3$
Round1:
$Mixcolumns(P0) \leftarrow Multi2[Temp0] \oplus Multi3[Temp1] \oplus Temp2 \oplus Temp3$
$Mixcolumns(P1) \leftarrow Temp0 \oplus Multi2[Temp1] \oplus Multi3[Temp2] \oplus Temp3$
$Mixcolumns(P2) \leftarrow Multi3[Temp0] \oplus Temp1 \oplus Multi2[Temp2] \oplus Temp3$
$Mixcolumns(P3) \leftarrow Temp0 \oplus Temp1 \oplus Multi2[Temp2] \oplus Multi3[Temp3]$
.....
Round2:
$Mixcolumns(Temp0) \leftarrow Multi2[P0] \oplus Multi3[P1] \oplus P2 \oplus P3$
$Mixcolumns(Temp1) \leftarrow P0 \oplus Multi2[P1] \oplus Multi3[P2] \oplus P3$
$Mixcolumns(Temp2) \leftarrow Multi3[P0] \oplus P1 \oplus Multi2[P2] \oplus P3$
$Mixcolumns(Temp3) \leftarrow P0 \oplus P1 \oplus Multi2[P2] \oplus Multi3[P3]$
.....
Round9:
$Mixcolumns(P0) \leftarrow Multi2[Temp0] \oplus Multi3[Temp1] \oplus Temp2 \oplus Temp3$
$Mixcolumns(P1) \leftarrow Temp0 \oplus Multi2[Temp1] \oplus Multi3[Temp2] \oplus Temp3$
$Mixcolumns(P2) \leftarrow Multi3[Temp0] \oplus Temp1 \oplus Multi2[Temp2] \oplus Temp3$
$Mixcolumns(P3) \leftarrow Temp0 \oplus Temp1 \oplus Multi2[Temp2] \oplus Multi3[Temp3]$
.....

1.3 Key Schedule

Erklären Sie ausführlich Ihre Implementierung des Key Schedules und nennen Sie ggf. Beispiele (Pseudocode).

Alle elf Rundschlüssen werden in der aes128_init() Funktion vorberechnet und in dem Programmspeicher gespeichert. In der Rundberechnung werden sie durchgesucht.

1. Zunächst wird malloc() aufgerufen, um die Speicherraum in Größe von 176 Bytes(*RK) zu erstellen. Danach speichert man Benutzer's Schlüsse also von RK[0] bis RK[3] in die ersten 16 Bytes.
2. Dann wird RK[4] durch G Funktion der höchsten 4 Bits von RK[3] und XOR mit RK[0]. Am Ende können andere vier Bytes Schlüssen leicht berechnet werden.
3. Alle andere 10 Rundschlüssen werden einer nach dem anderem auf dieser Weise hergestellt und gespeichert.

Pseudocode:

Algorithm 3 Key Schedule

Input: *Benutzer's Schlsse Pointer*
Output: *Rundschlsse Pointer*

$RK[0] \leftarrow UserKey[0]$
$RK[1] \leftarrow UserKey[1]$
$RK[2] \leftarrow UserKey[2]$
$RK[3] \leftarrow UserKey[3]$
$i \leftarrow 1$
while i **do**
  $b0 \leftarrow (RK[3] >> 24)\&0xff$
  $b1 \leftarrow (RK[3] >> 16)\&0xff$
  $b2 \leftarrow (RK[3] >> 8)\&0xff$
  $b3 \leftarrow RK[3]\ \&\ 0Xff$
  $RK[4] \leftarrow (b3 << 24) \oplus (b0 << 16) \oplus (b1 << 8) \oplus b2 \oplus RK[0]$
  $RK[5] \leftarrow RK[4] \oplus RK[1]$
  $RK[6] \leftarrow RK[5] \oplus RK[2]$
  $RK[7] \leftarrow RK[6] \oplus RK[3]$
  if $++i = 10$ **then**
    return $RK - 36$
  end if
  $RK+ = 4$
end while

1.4 Throughtput Optimierungen

Nennen Sie Ihre Optimierungen für einen höhen Durchsatz.

1. Keine Subfunktion wird in der Implementation gerichtet oder aufgerufen, weil die zeitaufwändig ist. Alle Runde in Funktion aes128_encrypt() bestehen sich nur in einer Schleife.
2. Statt direkter on-the-fly Berechnung werden die Ergebnisse der Multiplikation und SBox in den gespeicherten Lookup-Tabellen nachgeschlagen. Deswegen kann eine große Menge von Zeit bei der Berechnung gespart werden.

3. Zudem werden alle Berechnungen in dieser Algorithmus auf der acht-Bit Weise aufgrund des 8-Bit Prozessors durchgeführt.
4. Die Rundschlüssen werden voraussichtlich berechnet und im Speicher geschrieben. Das Programm braucht nur die Rundschlüssen im Speicher nachschlagen. Das kann auch zu Zeitsparen beitragen.

1.5 Codesize Optimierungen

Nennen Sie Ihre Optimierungen für eine kleine Codesize.

1. Bei der Implementation wird eine Schleifstruktur benutzt, um die Schleifenabwicklung zu vermeiden.
2. Eine genaue Größe der Datentype also uint8_t für Konstant ist benutzt.

1.6 RAM Optimierungen

Nennen Sie Ihre Optimierungen für wenig RAM-Verbrauch.

1. Alle Lookup-Tabellen werden in Programmspeicher gespeichert.
2. Statt der Schleifabwicklung wird eine Schleifstruktur eingesetzt. Auf dieser Weise kann auch ein bisschen RAM gespart werden.
3. Eine genaue Größe der Datentype z.B uint8_t(acht Bits) statt int (16 Bits), die für SBox durchsuchen geeignet ist, wird angewandt.

1.7 Results

Nennen Sie Ihre Ergebnisse inkl. Version des Compilers und die angewendete Optimization flags.

Durchschnittliche Laufzeit ist 7459,47.
Standardabweichung der Krypto-Laufzeit ist 2,87209.
RAM-Nutzung in RAM ist 301.
Programmspeichernutzung ist 5684.

Die Version des Compilers ist gcc 4.8.2 (Ubuntu 4.8.2-19ubuntul).

Die angewendete Optimization flags nach Reihenfolgen sind:
1. Throughtput Optimierungen.
2. RAM Optimierungen.
3. Codesize Optimierungen.

2 Bibliography

[1] J. Daemen and V. Rijmen. Aes proposal: Rijndael. NIST Web page, March 1999.
[2] J. Daemen and V. Rijmen. The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, 2002.
[3] National Institute of Standards and Technology. Federal information processing standards FIPS 197. Advanced Encryption Standard (AES), 2001.
[4] Joan Daemen and Vincent Rijmen: AES Proposal: Rijndael, available online at http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
[5] Intel Corporation: Intel® Advanced Encryption Standard(AES) Instructions Set, available online at: http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set
[6] Çetin Kaya Koç (Ed.), Cryptographic Engineering, Springer Verlag, 2009.
[7] National Institute of Standards and Technology (NIST): FIPS-197: Advanced Encryption Standard, 2001.
[8] National Institute of Standards and Technology (NIST): FIPS-46-3: Data Encryption Standard, 1999.