Key Wrapping

CS6025 Data Encoding

Yizong Cheng

3-3-15

Symmetric Key Encryption

- AES is a symmetric key encryption scheme that requires the encoder and decoder share the same secret key.
- The key is a secret shared by the sender and the receiver and must be delivered to both before AES can be used.
- One key distribution scheme requires each user to share a secret key with a distribution center and two ends of a transmission get a shared secret issued by the center, wrapped in the secret key (the key encryption key or kek) they share with the center.
- Other schemes have hierarchy of keys and low-level keys are used more often and top-level keys rarely but are used as keks.

A Robust Encryption Mode

- In other existing modes of operation, the last block of plaintext only influences the last block of ciphertext. The first block of ciphertext is only derived from the first block of plaintext.
- A robust mode should have every bit of the output depends on every bit of the input in nontrivial fashion.
- the value of key data is greater than the value of other data.
- key wrapping is only used for small amount of plaintext.
- Efficiency is not a major concern.

RFC 3394

2.2 Algorithms

The specification of the key wrap algorithm requires the use of the AES codebook [AES]. The next three sections will describe the key wrap algorithm, the key unwrap algorithm, and the inherent data integrity check.

Inputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}, and Key, K (the KEK).
Outputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.

1) Initialize variables.

Set A0 to an initial value (see 2.2.3)
For i = 1 to n
R[0][i] = P[i]

2) Calculate intermediate values.

For t = 1 to s, where s = 6n
A[t] = MSB(64, AES(K, A[t-1] | R[t-1][1])) ^ t
For i = 1 to n-1
R[t][i] = R[t-1][i+1]
R[t][n] = LSB(64, AES(K, A[t-1] | R[t-1][1]))

3) Output the results.

Set C[0] = A[t]
For i = 1 to n
C[i] = R[t][i]

Symbols in Stallings 12.8

$\text{MSB}_{64}(W)$	most significant 64 bits of W
$\text{LSB}_{64}(W)$	least significant 64 bits of W
W	temporary value; output of encryption function
$\oplus$	bitwise exclusive-OR
$\|$	concatenation
K	key encryption key
n	number of 64-bit key data blocks
s	number of stages in the wrapping process; $s = 6n$
P_i	ith plaintext key data block; $1 \leq i \leq n$
C_i	ith ciphertext data block; $0 \leq i \leq n$
$A(t)$	64-bit integrity check register after encryption stage t; $1 \leq t \leq s$
$A(0)$	initial integrity check value (ICV); in hexadecimal: A6A6A6A6A6A6A6A6
$R(t, i)$	64-bit register i after encryption stage t; $1 \leq t \leq s$; $1 \leq i \leq n$

Inputs: Plaintext, n 64-bit values $(P_1, P_2, \ldots, P_n)$
Key encryption key, K

Outputs: Ciphertext, $(n+1)$ 64-bit values $(C_0, C_1, \ldots, C_n)$

1. **Initialize variables.**

```
A(0) = A6A6A6A6A6A6A6A6
  for i = 1 to n
    R(0, i) = P_i
```

2. **Calculate intermediate values.**

```
for t = 1 to s
W = E(K, [A(t-1) || R(t-1, 1)])
A(t) = t ⊕ MSB_64(W)
R(t, n) = LSB_64(W)
for i = 1 to n-1
    R(t, i) = R(t-1, i+1)
```

3. **Output results.**

```
C_0 = A(s)
for i = 1 to n
    C_i = R(s, i)
```

Inputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}, and Key, K (the KEK).
Outputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}.

1) Initialize variables.

```
Set A = IV, an initial value (see 2.2.3)
For i = 1 to n
   R[i] = P[i]
```

2) Calculate intermediate values.

```
For j = 0 to 5
   For i=1 to n
      B = AES(K, A | R[i])
      A = MSB(64, B) ^ t where t = (n*j)+i
      R[i] = LSB(64, B)
```

3) Output the results.

```
Set C[0] = A
For i = 1 to n
   C[i] = R[i]
```

Inputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P1, P2, ..., Pn}.

1) Initialize variables.

```
Set A[s] = C[0] where s = 6n
For i = 1 to n
    R[s][i] = C[i]
```

2) Calculate the intermediate values.

```
For t = s to 1
    A[t-1] = MSB(64, AES-1(K, ((A[t] ^ t) | R[t][n]))
    R[t-1][1] = LSB(64, AES-1(K, ((A[t]^t) | R[t][n]))
    For i = 2 to n
        R[t-1][i] = R[t][i-1]
```

3) Output the results.

```
If A[0] is an appropriate initial value (see 2.2.3),
Then
    For i = 1 to n
        P[i] = R[0][i]
Else Return an error
```

Key Unwrapping

The key unwrapping algorithm can be defined as follows:

Inputs: Ciphertext, $(n+1)$ 64-bit values $(C_0, C_1, \ldots, C_n)$
Key encryption key, K

Outputs: Plaintext, n 64-bit values $(P_1, P_2, \ldots, P_n)$, ICV

1. **Initialize variables.**

```
A(s) = C0
for i = 1 to n
R(s, i) = Ci
```

2. **Calculate intermediate values.**

```
for t = s to 1
W = D(K, [(A(t) ⊕ t) || R(t, n)])
A(t-1) = MSB64(W)
R(t-1, 1) = LSB64(W)
for i = 2 to n
      R(t-1, i) = R(t, i-1)
```

3. **Output results.**

```
if A(0) = A6A6A6A6A6A6A6A6
then
  for i = 1 to n
    P(i) = R(0, i)
else
  return error
```

Inputs: Ciphertext, (n+1) 64-bit values {C0, C1, ..., Cn}, and Key, K (the KEK).
Outputs: Plaintext, n 64-bit values {P0, P1, K, Pn}.

1) Initialize variables.

```
Set A = C[0]
For i = 1 to n
    R[i] = C[i]
```

2) Compute intermediate values.

```
For j = 5 to 0
    For i = n to 1
        B = AES-1(K, (A ^ t) | R[i]) where t = n*j+i
        A = MSB(64, B)
        R[i] = LSB(64, B)
```

3) Output results.

If A is an appropriate initial value (see 2.2.3),
Then

```
    For i = 1 to n
        P[i] = R[i]
```

Else Return an error

H13A.java

```
public static void main(String[] args){
   if (args.length < 2){
     System.err.println("Usage: java H13A kek key");
     return;
   }
   H13A h13 = new H13A();
   h13.makeLog();
   h13.buildS();
   h13.readKek(args[0]);
   h13.expandKey();
   h13.readPlaintext(args[1]);
   h13.keyWrap();
   h13.outputResult();
}
```

```
void readPlaintext(String filename){
  Scanner in = null;
  try {
    in = new Scanner(new File(filename));
  } catch (FileNotFoundException e){
    System.err.println(filename + " not found");
    System.exit(1);
  }
  String line = in.nextLine();
  in.close();
  numberOfKeyDataBlocks = line.length() / 2 / keyDataBlockSize;
  R = new int[numberOfKeyDataBlocks][keyDataBlockSize];  // set R
  for (int i = 0; i < numberOfKeyDataBlocks; i++){
    int offset = i * keyDataBlockSize * 2;
    for (int j = 0; j < keyDataBlockSize; j++)
      R[i][j] = Integer.parseInt(
        line.substring(offset + j * 2, offset + (j + 1) * 2), 16);
  }
  for (int j = 0; j < keyDataBlockSize; j++) AA[j] = 0xa6;  // set A
}
```

keyWrap in H13A

```
void keyWrap(){
  for (int j = 0; j < 6; j++)   // s = 6n rounds
    for (int i = 0; i < numberOfKeyDataBlocks; i++){
       for (int k = 0; k < keyDataBlockSize; k++){
          state[keyDataBlockSize + k] = AA[k];
          state[k] = R[i][k];
       }     // set state
       blockCipher();   // AES block cipher
       for (int k = 0; k < keyDataBlockSize; k++){
          AA[k] = state[keyDataBlockSize + k];
          R[i][k] = state[k];
       }     // A || R from state
       AA[keyDataBlockSize - 1] ^= j * numberOfKeyDataBlocks + i + 1;
    }
}
```

Output Result as Hexadecimal String

```
void outputResult(){
  for (int k = 0; k < keyDataBlockSize; k++)
    if (AA[k] < 16) System.out.print("0" + Integer.toHexString(AA[k]));
    else System.out.print(Integer.toHexString(AA[k]));
  for (int i = 0; i < numberOfKeyDataBlocks; i++)
    for (int k = 0; k < keyDataBlockSize; k++)
      if (R[i][k] < 16) System.out.print("0" + Integer.toHexString(R[i][k]));
      else System.out.print(Integer.toHexString(R[i][k]));
}
```

Homework 13: due 3-9-15

- Complete H13B.java that decodes H13A.java.
  - A successful implementation of H11B is needed.
- Test your program on stallingskey.txt (as kek) and test13.txt as the concatenation of A and R's in hexadecimal.
- Your output should recover the original A (a6a6a6a6...) followed by the key data used below to get test13.txt
  - java H13A stallingskey.txt keydata.txt > test13.txt
  - java H13B stallingskey.txt test13.txt (recovers keydata.txt)