Security Issues in Cloud Computing

Modern Cryptography I – Symmetric Cryptography

Peter Schwabe

October 14, 2011

The binary alphabet

So far we considered the alphabet $A, B, \ldots, Z$, and noted that it could also be written as $0, \ldots, 25$. Encryption used addition modulo 26.
Modern cryptography is targeted at computations carried out by a computer, it typically uses the alphabet $0, 1$ (binary alphabet).

- We call a letter at this alphabet *bit*.
- A word of length 8 is called a *byte*.
- We use the term n-bit string for a word of length n.
- Most important operation is addition modulo 2, denoted $\oplus$, we call this operation xor (exclusive or). This operation can be extended to n-bit strings by applying it letter-wise, for example:

$$\begin{array}{ccccccccc} & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ \oplus & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{array}$$

- Addition is the same as subtraction modulo 2. Adding (xoring) the same value twice results in the orignial value.
- All information stored on typical computers is represented as words over this alphabet.

Stream ciphers

Recall Vernam's cipher: Encryption is performed by adding (now: xoring) a random key stream to the message. The main disadvantage is that the random key needs to be as long as the message.

Idea: Use a relatively short random key. Use that to deterministically generate a random-looking (pseudo-random) key stream, xor this with the message. To obtain different key streams from the same key one usually uses a non-secret *initalization vector* (or *nonce*), as additional input. This nonce ("number used once") must not be used twice!

Security:

- Obtaining part of the key stream used to encrypt some message is the same as for Vernam: This part of the message can be deciphered but nothing else.
- Obtaining the key used to genereate the key stream completely breaks all ciphertexts.
- It must be impossible to try all keys.
- Typical lengths of keys:
  - 64 bits (very low security, can be broken in practice).
  - 80 bits (low security, can probably be broken by exhaustive search)
  - 128 bits (high security, exhaustive search is impossible)
  - 192 or 256 bits (very high security).
- It must be impossible to draw conclusions from the key stream to the key (Known-plaintext attack). Best attack should be exhaustive search.
- Keys must be chosen carefully at random.

RC4 and eSTREAM

RC4 was (and maybe still is) one of the most widely used stream ciphers.

- Invented in 1987 by Ron Rivest;
- Leaked to cypherpunks mailinglist and further to the sci.crypt newsgroup in 1994: `http://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0`;
- Key size: initially 40 bits (due to US export restrictions), now 104 or 232 bits;
- It was used in Wired Equivalent Privacy (WEP), the WiFi encryption.
- Weakness in how RC4 was used allowed to find a 104-bit WEP in less than 1 minute (Tews, Weinmann, Pychkine, 2007)
- RC4 is used in a more secure way in WPA and SSL.

In 2004 the European Network of Excellence in Cryptography (ECRYPT) Started eSTREAM:

- Call for submissions for a new stream-cipher algorithm.
- 2 profiles:

    - Profile 1 : designed for software implementation aiming at high throughput.
    - Profile 2 : designed for hardware implementation with restricted resources.

- 23 submissions for profile 1 and 25 for profile 2 (with overlaps)

- 3 phases of public evaluation until April 2008.

- Suggested portfolio:

  Profile 1:

    - HC-128 (Hongjun Wu);
    - Rabbit (Martin Boesgaard, Mette Vesterager, Thomas Christensen, and Erik Zenner);
    - Salsa20/12 (Daniel J. Bernstein);
    - SOSEMANUK (Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Herv Sibert)

  Profile 2:

    - Grain v1 (Martin Hell, Thomas Johansson and Willi Meier);
    - MICKEY (Steve Babbage and Matthew Dodd);
    - Trivium (Christophe De Cannière and Bart Preneel);

- The use of all algorithms in the eSTREAM portfolio is not restricted by any patents.

Block Ciphers

Definition:
A block cipher maps n-bit blocks of plaintext to n-bit ciphertext blocks under the use of a key K. For a fixed key K, it is an invertible map

$$E_K : \{0,1\}^n \to \{0,1\}^n.$$

The inverse is

$$E_K^{-1} : \{0,1\}^n \to \{0,1\}^n.$$

How do we use block ciphers?

Consider input $M_1, \ldots, M_t$ of n-bit message blocks. Apply the function E_K using one of the following modes of operation.

ECB (Electronic Code Book Mode)

Encryption:
Obtain ciphertext $C_1, \ldots, C_t$ as $C_i = E_K(M_i), i = 1, \ldots, t$

Decryption:
Obtain the plaintext from $C_1, \ldots, C_t$ as $M_i = E_K^{-1}(C_i), i = 1, \ldots, t$

CBC (Cipher Blockchaining mode)

CBC uses a (non-secret) initialization vector (IV) of n bits.

Encryption:
Obtain ciphertext $C_1, \ldots, C_t$ as $C_i = E_K(M_i \oplus C_{i-1}), i = 1, \ldots, t; C_0 = IV$

Decryption:
Obtain message from $C_1, \ldots, C_t$ as $M_i = E_K^{-1}(C_i) \oplus C_{i-1}, i = 1, \ldots, t; C_0 = IV$

CFB (Cipher Feedback Mode)

Also CFB uses a non-secret IV of n bits.

Encryption:
Obtain ciphertext $C_1, \ldots, C_t$ as $C_i = E_K(C_{i-1}) \oplus M_i, i = 1, \ldots, t; C_0 = IV$

Decryption:
Obtain plaintext from $C_1, \ldots, C_t$ as $M_i = E_K(C_{i-1}) \oplus C_i, i = 1, \ldots, t; C_0 = IV$

OFB (Output Feedback Mode)

OFB also uses a non-secret IV of n bits.

Encryption:
Generate $O_1, \ldots, O_t$ as $O_i = E_K(O_{i-1}), i = 1, \ldots, t, O_0 = IV$
Obtain $C_i = M_i \oplus O_i, i = 1, \ldots, t$

Decrytion:
Generate key stream $O_1, \ldots, O_t$ as in encryption
Obtain $M_i = C_i \oplus O_i, i = 1, \ldots, t$

CTR (Counter Mode)

The CTR mode uses a nonce N of l bits, $l < n$
CTR uses a counter start value C of $n - l$ bits.

Encryption:
Generate a Keystream $O_i, \ldots, O_t$ as $O_i = E_K(N|((C + i) \mod 2^{n-l}))$ Compute $C_i = M_i \oplus O_i, i = 1, \ldots, t$

Decryption:
Obtain the message as $M_i = C_i \oplus O_i, i = 1, \ldots, t$

Properties of modes of operation (and remarks)

- ECB is considered insecure if applied to more than one block, because identical input blocks map to identical output blocks. See the pictures at `http://en.wikipedia.org/wiki/Electronic_code_book#Electronic_codebook_.28ECB.29`.
- In CBC and CFB mode, the last ciphertext block C_t depends on all message blocks. In ECB, OFB, CTR modes each ciphertext block only depends on one plaintext block,
- CBC, CFB, and OFB encryption is not parallelizable. ECB and CTR encryption is parallelizable. CBC and CFB decryption is also parallelizable (also ECB, CTR)
- Modes with parallelizable decryption allow random access to the ciphertext.
- CBC and ECB require padding of the input to a multiple of the block length. CFB, OFB and CTR don't.
- For OFB, CFB and CTR, each two messages encryted with the same key must use a different IV(nonce)
- Most widely used: CBC and CTR

The Advanced Encryption Standard (AES)

For the description in comic form see
`http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html`

History

Sept. 1997: NIST issued a public call for a new block cipher, supporting a block length of 128 bits and lengths of 128, 192, and 256 bits.

August 98 and March 99: AES1 and AES2 conferences organized by NIST.

August 99: NIST announces 5 finalists:

- MARS(IBM)
- RCG (Rivest, Robshaw, Sidney, Yin)
- Rijndael (Daemen, Rijmen)
- Serpent (Anderson, Biham, Knudsen)
- Twofish (Schneier)

April 2000: AES3 conference

October 2, 2000: NIST announces that Rijndael has been selected as the proposed AES

Description of AES

AES has a block length of 128 bits (16 bytes), keylengths of 128, 192, and 256 bits Encryption transforms a 128-bit input in m rounds into a 128-bit output using m+1 16-byte round keys $K_0, \ldots, K_m$ derived from the AES key The number of rounds depends on the length of the key, for 128-bit keys it uses 10 rounds, for 192-bit keys 12 rounds and for 256-bit keys 14 rounds.

Algorithm 1 AES-128 encryption

Require: 128-bit input block B, 128-bit AES round keys $K_0, \ldots, K_{10}$
Ensure: 128-bit block of encrypted output

```
B ← AddRoundKey(B, K_0)
for i from 1 to 9 do
    B ← SubBytes(B)
    B ← ShiftRows(B)
    B ← MixColumns(B)
    B ← AddRoundKey(B, K_i)
end for
B ← SubBytes(B)
B ← ShiftRows(B)
B ← AddRoundKey(B, K_10)
return B
```