V I T
U N I V E R S I T Y

(Estd. u/s 3 of UGC Act 1956)
Vellore - 632 014, Tamil Nadu, India

School of Information Technology & Engineering

B.Tech. – Information Technology

ITE320 Network and Information Security

RC2 Cryptosystem

MUKUL DEV (13BIT0269)

Algorithm

In cryptography, RC2 (also known as ARC2) is a symmetric-key block cipher designed by Ron Rivest in 1987. "RC" stands for "Ron's Code" or "Rivest Cipher"; other ciphers designed by Rivest include RC4, RC5, and RC6.The development of RC2 was sponsored by Lotus, who were seeking a custom cipher that, after evaluation by the NSA, could be exported as part of their Lotus Notes software. The NSA suggested a couple of changes, which Rivest incorporated. After further negotiations, the cipher was approved for export in 1989. Along with RC4, RC2 with a 40-bit key size was treated favourably under US export regulations for cryptography.Initially, the details of the algorithm were kept secret — proprietary to RSA Security — but on 29 January 1996, source code for RC2 was anonymously posted to the Internet on the Usenet forum, sci.crypt. Mentions of CodeView and SoftICE (popular debuggers) suggest that it had been reverse engineered. A similar disclosure had occurred earlier with RC4. In March 1998 Ron Rivest authored an RFC publicly describing RC2 himself. RC2 is a 64-bit block cipher with a variable size key. Its 18 rounds are arranged as a source-heavy unbalanced Feistel network, with 16 rounds of one type (MIXING) punctuated by two rounds of another type (MASHING). A MIXING round consists of four applications of the MIX transformation, as shown in the diagram. RC2 is vulnerable to a related-key attack using 234 chosen plaintexts (Kelsey et al., 1997). This was designed as a proposal to replace the existing DES Algorithm.

RC2 works on 64-bit blocks which are divided into four words of each sixteen bits. It is an iterated block cipher where the ciphertext is computed as a function of the plaintext and the secret key in a number of rounds. There are two kinds of rounds in RC2, the mixing rounds and the mashing rounds. There are in total 16 mixing rounds and two mashing rounds. In each round each of the four sixteen-bit words in an intermediate ciphertexts is updated as a function of the other words. Each of the mixing rounds takes a 16-bit subkey. The 64 subkeys are derived from the user selected key which can be of length from one to

128 bytes. An additional parameter of the algorithm is the effective key length, which will be explained below. We note that the decryption operation does not equal the encryption operation which may have unfortunate impacts on implementations. Also, RC2 is not a fast cipher and an optimized version of DES and any of the five AES finalists is likely to produce higher throughputs than RC2.

Logic/design elements used in the algorithm

Key Explanation

The key-schedule takes a user-selected key and a number representing the maximum effective key length. The latter is a feature not seen in any other block ciphers as far as this author is informed. Assume that the user-selected key consists of T bytes where $1 \leq T \leq 128$. Let L be a key buffer (an array) of 128 bytes. The T bytes are loaded into L[0], ..., L[T −1] of the key buffer. The maximum effective key length in bits is denoted T1. The effective key length in bytes T8 is defined as T8 = dT1/8e and a mask TM as TM = 255 mod 28(1−T8)+T1 . As a first observation we found that the latter is equivalent to TM = 2T1 mod 8 − 1.

The key expansion consists of the following two iterations, where Π is a table consisting of a permutation of the numbers 0, . . . , 127 derived from the expansion of π:

1. for i = T, T + 1, ..., 127 do L[i] = Π[L[i − 1] + L[i − T]], where addition is modulo 256
2. L[128 − T8] = Π[L[128 − T8] & TM]
3. for i = 127 − T8, ..., 0 do L[i] = Π[L[i + 1] $\oplus$ L[i + T8]]

Finally, define the 64 subkeys, K[i] for i = 0, . . . , 63 as follows: K[i] = L[2i] + 256 × L[2i + 1].

The terms T8 and TM ensure that the expanded key table is derived from only T1 bits, such that an exhaustive search can be performed in 2T1 operations independent of the length of the user-selected key.

Encryption / Decryption

The two kinds of rounds in RC2 are defined via the operations MIX and MASH. The plaintext is divided into four words of each sixteen bits denoted R[0], . . . , R[3].

The MIX operation is defined as follows, where s[0] = 1, s[1] = 2, s[2] = 3, and s[3] = 5.

R[i] = R[i] + K[j] + (R[i − 1] & R[i − 2]) + (∼R[i − 1] & R[i − 3]);

j = j + 1;

R[i] = R[i] <<< s[i];

Here the indices of R are computed modulo 4, and j is a pointer such that K[j] is the first key word in the expanded key which has not yet been used in a MIX operation. A mixing round consists of four consecutive MIX steps, such that each of the words R[0], R[1], R[2], and R[3] are modified and in that order.

The MASH operation is defined as follows:

R[i] = R[i] + K[R[i − 1] & 003fx],

in other words, the least significant six bits of R[i − 1] are used to select one of the 64 subkeys. A mashing round consists of four MASH operations such that each of the words R[0], R[1], R[2], and R[3] are modified.

1. Let the words R[0], ..., R[3] hold the 64-bit plaintext block.
2. Perform the key expansion such that the words K[0], . . . , K[63] hold the subkeys.
3. Initialize j to zero.
4. Do five mixing rounds.
5. Do one mashing round.
6. Do six mixing rounds.
7. Do one mashing round.
8. Do five mixing rounds.
9. The ciphertext is defined as the resulting values of R[0], ..., R[3].

Decryption is the reverse of encryption. Clearly, it suffices to define the inverse operations of the MIX and MASH operations. The inverses of the MIX operations are defined as follows:

R[i] = R[i] >>> s[i];

R[i] = R[i] − K[j] − (R[i − 1] & R[i − 2]) − (

∼R[i − 1] & R[i − 3]);

j = j − 1;

The inverses of the MASH operations are defined as follows:

R[i] = R[i] − K[R[i − 1] & 003fx].

An inverse mixing round consists of four consecutive inverse MIX steps, such that each of the words R[3], R[2], R[1], and R[0] are modified and in that order, and similarly for the inverse mashing rounds. Decryption can now be defined.

1. Let the words R[0], . . . , R[3] hold the 64-bit ciphertext block.
2. Perform the key expansion such that the words K[0], . . . , K[63] hold the subkeys.
3. Initialize j to 63.
4. Do five inverse mixing rounds.
5. Do one inverse mashing round.
6. Do six inverse mixing rounds.
7. Do one inverse mashing round.
8. Do five inverse mixing rounds.
9. The plaintext is defined as the resulting values of R[0], . . . , R[3]

Strength of the algorithm

The primary advantage of public-key cryptography is increased security and convenience: private keys never need to be transmitted or revealed to anyone. In a secret-key system, by contrast, the secret keys must be transmitted (either manually or through a communication channel) since the same key is used for encryption and decryption. A serious concern is that there may be a chance that an enemy can discover the secret key during transmission.

Another major advantage of public-key systems is that they can provide digital signatures that cannot be repudiated. Authentication via secret-key systems requires the sharing of some secret and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming the shared secret was somehow compromised by one of the parties sharing the secret. For example, the Kerberos secret-key authentication system involves a central database that keeps copies of the secret keys of all users; an attack on the database would allow widespread forgery. Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

Weakness of the algorithm

A disadvantage of using public-key cryptography for encryption is speed. There are many secret-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used with secret-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public- and secret-key systems in order to get both the security advantages of public-key systems and the speed advantages of secret-key systems. Such a protocol is called a digital envelope, which is explained in more detail in Question 2.2.4.

Public-key cryptography may be vulnerable to impersonation, even if users' private keys are not available. A successful attack on a certification authority will allow an adversary to impersonate whomever he or she chooses by using a public-key certificate from the compromised authority to bind a key of the adversary's choice to the name of another user.

Attacks possible

Vulnerable to various Differential and Brute Force attacks.
RC2 is a currently "unbroken" block cipher; the best known attack is a "related-key attack", i.e. something which does not really apply in practice (related-key attacks are about specific properties when the victim uses two distinct keys, that the attacker does not know, but with a difference between them that the attacker knows).

Attacks impossible

Known Plaintext attack: RC2 is provably immune to linear cryptanalysis attack.
Chosen plaintext only: RC2 is immune to differential analysis with any number of texts.
Combination Attack: This cipher is immune to the combination attacks currently known in the literature.

Drawbacks

It uses 64-bit blocks. This makes it unsuitable to processing large data sets, the cut-off being around 2n/2 blocks when blocks consist in n bits. For a 64-bit block cipher, this amounts to about 30 gigabytes with a single key, which is large but not unattainable in some contexts. Encrypting more data than that implies a risk of revealing part of the plaintext. Note that 3DES suffers from exactly the same limitation.

Its performance sucks. About as much as that of 3DES; on my (underpowered) laptop, OpenSSL's implementation achieves about 10 MB/s with 3DES, 12 MB/s with RC2. There again, for emails, this should not be an issue.

RC2, in its description, uses a lot of data-dependent lookup tables. In that sense, it would be very difficult to make an implementation that is "constant-time" (i.e. does not leak information to attackers that can run their own code on the same hardware, through cache-induced timing differences); at least, it would be very difficult to make one without a massive slowdown (like a 100x slowdown factor). Note that classic 3DES implementations are not constant-time either, but that can be done with only a "moderate" (relatively) slowdown (about 3.5x with my own code).

RC2 has a configurable key length, between 1 and 128 bytes (i.e. 8 to 1024 bits, and multiple of 8). Thus, while RC2 itself can be a tolerably fine algorithm, it can still be used with a key which is way too short for ensuring a decent level of security. RC2 also includes an additional parameter (called "effective key length") that can be used to limit the brute-force resistance. Historically, RC2 has been much used in setups meant to comply with the pre-2000 US crypto export rules, with a typical strength equivalent to 40 bits (i.e. not strong at all).

Computational power and other resources required

RC2 works on 64-bit blocks which are divided into four words of each sixteen bits. It is an iterated block cipher where the ciphertext is computed as a function of the plaintext and the secret key in a number of rounds. There are two kinds of rounds in RC2, the mixing rounds and the mashing rounds. There are in total 16 mixing rounds and two mashing rounds. In each round each of the four sixteen-bit words in an intermediate ciphertexts is updated as a function of the other words. Each of the mixing rounds takes a 16-bit subkey. The 64 subkeys are derived from the user selected key which can be of length from one to 128 bytes.

Application

In SSL v2 there are two options for data encryption with RC2, one using a 40-bit key and one using a 128-bit key. One option in SSL v3 for data encryption is RC2 with a 40-bit key. As far as we are informed, in all theses implementations RC2 is used for data encryption using the standard CBC mode. The only potential problem with the above implementations is the use of the 40-bit keys, which give only very little security. It has been demonstrated that with special-purpose hardware a 56-bit DES keys can be found by exhaustive search in a matter of days. Moreover, a 40-bit key can be found quickly using general-purpose machines.

Suggestions/proposal for Extension/Enhancement/Improvement/Modification of this Cryptosystem

Decrease the number of rounds because it makes the algorithm slow

Program: C/C++/Java code

```
// RC2 C++ source code (Encryption and Decryption)

#include <string.h>
#include <assert.h>

void RC2Keyschedule::schedule( unsigned short K[64], const unsigned char
L[128], unsigned T8, unsigned TM )

        {
        unsigned char x;
        unsigned i;

        static const unsigned char PITABLE[256] = {
            217,120,249,196, 25,221,181,237, 40,233,253,121, 74,160,216,157,
            198,126, 55,131, 43,118, 83,142, 98, 76,100,136, 68,139,251,162,
             23,154, 89,245,135,179, 79, 19, 97, 69,109,141,  9,129,125, 50,
            189,143, 64,235,134,183,123, 11,240,149, 33, 34, 92,107, 78,130,
             84,214,101,147,206, 96,178, 28,115, 86,192, 20,167,140,241,220,
             18,117,202, 31, 59,190,228,209, 66, 61,212, 48,163, 60,182, 38,
            111,191, 14,218, 70,105,  7, 87, 39,242, 29,155,188,148, 67,  3,
            248, 17,199,246,144,239, 62,231,  6,195,213, 47,200,102, 30,215,
              8,232,234,222,128, 82,238,247,132,170,114,172, 53, 77,106, 42,
            150, 26,210,113, 90, 21, 73,116, 75,159,208, 94,  4, 24,164,236,
```

```
            194,224, 65,110, 15, 81,203,204, 36,145,175, 80,161,244,112, 57,
            153,124, 58,133, 35,184,180,122,252,  2, 54, 91, 37, 85,151, 49,
             45, 93,250,152,227,138,146,174,  5,223, 41, 16,103,108,186,201,
            211,  0,230,207,225,158,168, 44, 99, 22,  1, 63, 88,226,137,169,
             13, 56, 52, 27,171, 51,255,176,187, 72, 12, 95,185,177,205, 46,
            197,243,219, 71,229,165,156,119, 10,166, 32,104,254,127,193,173
        };

        assert(len > 0 && len <= 128);
        assert(bits <= 1024);
        if (!bits)
                bits = 1024;

        memcpy(xkey, key, len);

        for (i = 0; i < 128; i++) {
                L[i] = PITABLE[L[i-1] + L[i-T]];
        }

        T8 = (T1+7) >> 3;
        TM = 255 MOD 2^(8 + T1 - 8*T8);

        L[128-T8] = PITABLE[L[128-T8] & TM];

        for (i = 0; i < 127-T8; i++) {
                L[i] = PITABLE[L[i+1] XOR L[i+T8]];
        };

        i = 63;

        K[i] = L[2*i] + 256*L[2*i+1];
        };

void RC2Encryption::ProcessBlock( const unsigned short K[64], )
        {
        unsigned R3, R2, R1, R0, i;

        for (i = 0; i < 16; i++) {
                R0 += (R1 & ~R3) + (R2 & R3) + K[4*i+0];
                R0 = R0 << 1;

                R1 += (R2 & ~R0) + (R3 & R0) + K[4*i+1];
                R1 = R1 << 2;

                R2 += (R3 & ~R1) + (R0 & R1) + K[4*i+2];
                R2 = R2 << 3;

                R3 += (R0 & ~R2) + (R1 & R2) + K[4*i+3];
                R3 = R3 << 5;

                if (i == 4 || i == 10) {
                        R0 += K[R3 & 63];
                        R1 += K[R0 & 63];
                        R2 += K[R1 & 63];
                        R3 += K[R2 & 63];
                }
        }
```

```
void RC2Decryption::ProcessBlock( const unsigned short K[64], )
        {
        unsigned R3, R2, R1, R0, i;


        for (i = 0; i < 16; i++) {
                R3 = R3 << 5;
                R3 += (R0 & ~R2) + (R1 & R2) + K[4*i-3];

                R2 = R2 << 3;
                R2 += (R3 & ~R1) + (R0 & R1) + K[4*i-2];

                R1 = R1 << 2;
                R1 += (R2 & ~R0) + (R3 & R0) + K[4*i-1];

                R0 = R0 << 1;
                R0 += (R1 & ~R3) + (R2 & R3) + K[4*i-0];

                if (i == 4 || i == 10) {
                        R3 -= K[R2 & 63];
                        R2 -= K[R1 & 63];
                        R1 -= K[R0 & 63];
                        R0 -= K[R3 & 63];
                }
        }
```