INTERNET-DRAFT
14 July 1999
Expires: 17 December 1999

K.Kaukonen
R.Thayer
SSH

A Stream Cipher Encryption Algorithm "Arcfour"
<draft-kaukonen-cipher-arcfour-03.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html.

Abstract

This document describes an algorithm here called Arcfour that is believed to be fully interoperable with the RC4 algoritm. RC4 is trademark of RSA Data Security, Inc. There is a need in the Internet community for an encryption algorithm that provides interoperable operation with existing deployed commercial cryptographic applications. This interoperability will allow for a smoother transition to protocols that have been developed through the IETF standards process.

Contents

1. Introduction

There is a need in the Internet community for an encryption algorithm that provides interoperable operation with existing deployed commercial cryptographic applications. This interoperability allows for a smoother transition to protocols that have been developed through the IETF standards process. This document describes an existing algorithm that satisifies this requirement.

There is a large body of experience in developing and deploying encryption applications, especially in the HTTP/HTML browser/server markets. These browsers typically implement the RC4 encryption algorithm provided by [RSA]. It would be beneficial for the IETF standards processes to produce protocols that can be deployed into existing Internet environments. This would allow graceful addition of new (IETF-developed) protocols. It would allow less disruption of existing users, since there would be more interoperability between pre-exisiting protocols and IETF-based protocols.

2. Requirements for this Encryption Algorithm

The algorithm described here is called Arcfour, and it has been chosen because it is compatible with the RC4(TM) algorithm that is one of the most popular encryption algorithms in the browser market. (See chapter Intellectual Property Considerations.) Arcfour is potentially useful in several environments, including IPSEC [IPSEC], SSH [SSH], and TLS [TLS]. There are existing Internet Drafts that describe how it can be applied, see e.g. [Caronni], [SSH], and [TLS].

The algorithm can be used with a variety of key lengths. It specifically can be operated with 40-bit keys and with 128-bit keys. See the Security Considerations section for comments on the use of 40-bit keys.

Compatibility of the algorithm with commercial algorithms can be

tested by comparing the encrypted data that is produced by the test vectors listed in the appendix to this document.

3. Description of Algorithm

The algorithm itself is documented in [Schneier], pages 397-398, in the chapter titled "Other Stream Ciphers and Real Random Sequence Generators".

3.1 Key Setup

1. **Allocate an 256 element array of 8 bit bytes to be used as an S-box, label it**

   S [0] .. S [255].

2. **Initialize the S-box.** Fill each entry first with it's index:

   S [0] = 0; S [1] = 1; etc. up to S [255] = 255;

3. **Fill another array of the same size (256) with the key, repeating bytes as necessary.**

```
for (i = 0; i < 256; i = i + 1)
  S2 [i] = key [i % keylen];
```

4. **Set j to zero and initialize the S-box like this:**

```
for (i = 0; i < 256; i = i + 1)
  {
  j = (j + S [i] + S2 [i]) % 256;
  temp = S [i];
  S [i] = S [j];
  S [j] = temp;
  }
```

5. **Initialize i and j to zero.** If superuser priviledged program sniffing is feared (that is, always) set also the S2 array and the key array to zero. That gives a slightly better protection since the key is believed to be not feasible to calculate after it has been zeroed and thus forgotten.

3.2 Stream Generation

For either encryption or decryption, the input text is processed one

byte at a time. A pseudorandom byte K is generated:

```
i = (i+1) % 256;
j = (j + S[i]) % 256;
temp = S [i];
S [i] = S [j];
S [j] = temp;
t = (S [i] + S [j]) % 256;
K = S [t];
```

To encrypt, XOR the value K with the next byte of the plaintext. To decrypt, XOR the value K with the next byte of the ciphertext.

4. Intellectual Property Considerations

This document does not address Intellectual Property issues. No claim is made as to who owns this algorithm, of the performance of the algorithm, its cryptographic security or any other liability issues related to the algoritm itself, its implementation or use.

The Arcfour algorithm is believed to be fully interoperable with the RC4 algorithm. "RC4" is believed to be trademark of RSA Data Security, Inc. Contact [RSA] if RC4(TM) algorithm is needed.

5. Acknowledgements

This work was based on conversations with several collegues within the IETF.

6. Security Considerations

This algorithm can be operated with several different key sizes. If the key is 128 bits in length then this algorithm is believed to be secure. If the key length is significantly shorter, specifically 40 bits, then there are known attacts that have been successfully applied. For this algorithm to be operated in a cryptographically sound manner it is believed that a key length of 128 bits or more should be used.

On the other hand, the 40-bit version of this algorithm is specifically regulated by the U.S. Government. This means that deployment of 40-bit implementations may be easier to export than alternative algorithms.

It must be strongly recommended that no two plaintexts are encrypted

with the same key. Otherwise the plaintext can usually be broken, and often even quite easily. If the two encrypted messages are XORed together, the result is the XOR of the original plaintexts. Given the encrypted messages are text strings, credit card numbers, or other byte streams with some known properties, the plaintexts can be estimated with great accuracy. See the [DAWSON AND NIELSEN] for more details.

Initial cryptanalysis results are favorable, but the current literature should be consulted to assess the security of this cipher. A good starting point for a citation search would be [GOLIC]. For Internet news group posting, start with [FINNEY], [JENKINS] and [ROOS].

7. References

[Caronni] Caronni, G., Waldvogel, M. "The ESP Stream Transform", ftp://ds.internic.net/internet-drafts/draft-caronni-esp-stream-01.txt, September, 1996.

[COMMERCE] Test vectors issued by United States Department of Commerce, Bureau of Export Administration, Office of Strategic Trade and Foreign Policy, Strategic Trade Controls Division.

[CRYPTLIB] Gutmann, P, Young, E., Plumb, C. "Cryptlib, A Portable Encryption Library", Version 2.00. http://www.cs.auckland.ac.nz/~pgut001/cryptlib.html, 1996.

[DAWSON AND NIELSEN] Dawson E. and Nielsen L.: Automated Cryptoanalysis of XOR Plaintext Strings, Cryptologia, April 1996, Volume XX, Number 2.

[FINNEY] Finney, H. Internet message posted to sci.crypt 21 September, 1994.

[GOLIC] Golic, J. "Linear Statistical Weakness of Alleged RC4 Keystream Generator." In, W. Fumy (ed.), Proceedings of Eurocrypt

'97, 226-238, Springer-Verlag, 1997.

[IPSEC] Atkinson, R, "Security Architecture for the Internet Protocol", ftp://ds.internic.net/rfc/rfc1825.txt, August 1995.

[JENKINS] Jenkins, B. Internet message posted to sci.crypt 22 September, 1994.

[ROOS] Roos, A. Internet message posted to sci.crypt 28 September, 1995.

[RSA] RSA Data Security, Inc., http://www.rsa.com, Address: RSA Data Security, Inc. 100 Marine Parkway, Suite 500, Redwood City, CA 94065-1031.

[SCHNEIER] Schneier, B. "Applied Cryptography", Second Edition, http://www.counterpane.com. Published by John Wiley & Sons, Inc. ISBN 0-471-11709-9, 1996.

[SSH] Ylonen, T., "SSH Transport Layer Protocol", ftp://ietf.org/internet-drafts/draft-ietf-secsh-transport-00.txt, March, 1997.

[SSH ARCFOUR] Kaukonen, K. Long test vectors for Arcfour and RC4 algorithms issued by Kalle Kaukonen, SSH Communications Security, Ltd, July, 1997.

[TLS] Freier, A., Karlton, P., Kocher, P., Dierks, T., "The TLS Protocol", ftp://ds.internic.net/internet-drafts/draft-ietf-tls-protocol-00.txt, December, 1996.

8. Colophon

8.1 Authors' Addresses

Kalle Kaukonen
SSH Communications Security Oy
Tekniikantie 12
02150 Espoo
Finland
kalle@ssh.fi
http://www.ssh.fi

Rodney Thayer
SSH Communications Security, Inc.
650 Castro Street, Suite 220
Mountain View, CA 94041
rodney@ipsec.com

8.2 About this document

This document was written in plain text using 'ifmt', a text formatter designed to work on generic C platforms. Original source is in plain ascii, transliterated from the original NROFF source.

IETF draft boilerplate was current at the time this document was published.

8.3 Change History

This is revision 02 of draft-kaukonen-cipher-arcfour. This revision is a resubmission only.

Appendix

A. Test Vectors

1. Test Vectors from [CRYPTLIB]:

Plain Text:
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
Key:
0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF

Cipher Text:
0x74, 0x94, 0xC2, 0xE7, 0x10, 0x4B, 0x08, 0x79

2. Test Vectors from [COMMERCE]:

Plain Text:
0xdc, 0xee, 0x4c, 0xf9, 0x2c
Key:
0x61, 0x8a, 0x63, 0xd2, 0xfb
Cipher Text:
0xf1, 0x38, 0x29, 0xc9, 0xde

3. Test Vectors from [SSH ARCFOUR]:

Plain Text:
0x52, 0x75, 0x69, 0x73, 0x6c, 0x69, 0x6e, 0x6e,
0x75, 0x6e, 0x20, 0x6c, 0x61, 0x75, 0x6c, 0x75,
0x20, 0x6b, 0x6f, 0x72, 0x76, 0x69, 0x73, 0x73,
0x73, 0x61, 0x6e, 0x69, 0x2c, 0x20, 0x74, 0xe4,
0x68, 0x6b, 0xe4, 0x70, 0xe4, 0x69, 0x64, 0x65,
0x6e, 0x20, 0x70, 0xe4, 0xe4, 0x6c, 0x6c, 0xe4,
0x20, 0x74, 0xe4, 0x79, 0x73, 0x69, 0x6b, 0x75,
0x75, 0x2e, 0x20, 0x4b, 0x65, 0x73, 0xe4, 0x79,
0xf6, 0x6e, 0x20, 0x6f, 0x6e, 0x20, 0x6f, 0x6e,
0x6e, 0x69, 0x20, 0x6f, 0x6d, 0x61, 0x6e, 0x61,
0x6e, 0x69, 0x2c, 0x20, 0x6b, 0x61, 0x73, 0x6b,
0x69, 0x73, 0x61, 0x76, 0x75, 0x75, 0x6e, 0x20,
0x6c, 0x61, 0x61, 0x6b, 0x73, 0x6f, 0x74, 0x20,
0x76, 0x65, 0x72, 0x68, 0x6f, 0x75, 0x75, 0x2e,
0x20, 0x45, 0x6e, 0x20, 0x6d, 0x61, 0x20, 0x69,
0x6c, 0x6f, 0x69, 0x74, 0x73, 0x65, 0x2c, 0x20,
0x73, 0x75, 0x72, 0x65, 0x20, 0x68, 0x75, 0x6f,

```
0x6b, 0x61, 0x61, 0x2c, 0x20, 0x6d, 0x75, 0x74,
0x74, 0x61, 0x20, 0x6d, 0x65, 0x74, 0x73, 0xe4,
0x6e, 0x20, 0x74, 0x75, 0x6d, 0x6d, 0x75, 0x75,
0x73, 0x20, 0x6d, 0x75, 0x6c, 0x6c, 0x65, 0x20,
0x74, 0x75, 0x6f, 0x6b, 0x61, 0x61, 0x2e, 0x20,
0x50, 0x75, 0x75, 0x6e, 0x74, 0x6f, 0x20, 0x70,
0x69, 0x6c, 0x76, 0x65, 0x6e, 0x2c, 0x20, 0x6d,
0x69, 0x20, 0x68, 0x75, 0x6b, 0x6b, 0x75, 0x75,
0x2c, 0x20, 0x73, 0x69, 0x69, 0x6e, 0x74, 0x6f,
0x20, 0x76, 0x61, 0x72, 0x61, 0x6e, 0x20, 0x74,
0x75, 0x75, 0x6c, 0x69, 0x73, 0x65, 0x6e, 0x2c,
```



```
0x20, 0x6d, 0x69, 0x20, 0x6e, 0x75, 0x6b, 0x6b,
0x75, 0x75, 0x2e, 0x20, 0x54, 0x75, 0x6f, 0x6b,
0x73, 0x75, 0x74, 0x20, 0x76, 0x61, 0x6e, 0x61,
0x6d, 0x6f, 0x6e, 0x20, 0x6a, 0x61, 0x20, 0x76,
0x61, 0x72, 0x6a, 0x6f, 0x74, 0x20, 0x76, 0x65,
0x65, 0x6e, 0x2c, 0x20, 0x6e, 0x69, 0x69, 0x73,
0x74, 0xe4, 0x20, 0x73, 0x79, 0x64, 0xe4, 0x6d,
0x65, 0x6e, 0x69, 0x20, 0x6c, 0x61, 0x75, 0x6c,
0x75, 0x6e, 0x20, 0x74, 0x65, 0x65, 0x6e, 0x2e,
0x20, 0x2d, 0x20, 0x45, 0x69, 0x6e, 0x6f, 0x20,
0x4c, 0x65, 0x69, 0x6e, 0x6f
```

Key:

```
0x29, 0x04, 0x19, 0x72, 0xfb, 0x42, 0xba, 0x5f,
0xc7, 0x12, 0x77, 0x12, 0xf1, 0x38, 0x29, 0xc9
```

Cipher Text:

```
0x35, 0x81, 0x86, 0x99, 0x90, 0x01, 0xe6, 0xb5,
0xda, 0xf0, 0x5e, 0xce, 0xeb, 0x7e, 0xee, 0x21,
0xe0, 0x68, 0x9c, 0x1f, 0x00, 0xee, 0xa8, 0x1f,
0x7d, 0xd2, 0xca, 0xae, 0xe1, 0xd2, 0x76, 0x3e,
0x68, 0xaf, 0x0e, 0xad, 0x33, 0xd6, 0x6c, 0x26,
0x8b, 0xc9, 0x46, 0xc4, 0x84, 0xfb, 0xe9, 0x4c,
0x5f, 0x5e, 0x0b, 0x86, 0xa5, 0x92, 0x79, 0xe4,
0xf8, 0x24, 0xe7, 0xa6, 0x40, 0xbd, 0x22, 0x32,
0x10, 0xb0, 0xa6, 0x11, 0x60, 0xb7, 0xbc, 0xe9,
0x86, 0xea, 0x65, 0x68, 0x80, 0x03, 0x59, 0x6b,
0x63, 0x0a, 0x6b, 0x90, 0xf8, 0xe0, 0xca, 0xf6,
0x91, 0x2a, 0x98, 0xeb, 0x87, 0x21, 0x76, 0xe8,
0x3c, 0x20, 0x2c, 0xaa, 0x64, 0x16, 0x6d, 0x2c,
0xce, 0x57, 0xff, 0x1b, 0xca, 0x57, 0xb2, 0x13,
0xf0, 0xed, 0x1a, 0xa7, 0x2f, 0xb8, 0xea, 0x52,
```

```
0xb0, 0xbe, 0x01, 0xcd, 0x1e, 0x41, 0x28, 0x67,
0x72, 0x0b, 0x32, 0x6e, 0xb3, 0x89, 0xd0, 0x11,
0xbd, 0x70, 0xd8, 0xaf, 0x03, 0x5f, 0xb0, 0xd8,
0x58, 0x9d, 0xbc, 0xe3, 0xc6, 0x66, 0xf5, 0xea,
0x8d, 0x4c, 0x79, 0x54, 0xc5, 0x0c, 0x3f, 0x34,
0x0b, 0x04, 0x67, 0xf8, 0x1b, 0x42, 0x59, 0x61,
0xc1, 0x18, 0x43, 0x07, 0x4d, 0xf6, 0x20, 0xf2,
0x08, 0x40, 0x4b, 0x39, 0x4c, 0xf9, 0xd3, 0x7f,
0xf5, 0x4b, 0x5f, 0x1a, 0xd8, 0xf6, 0xea, 0x7d,
0xa3, 0xc5, 0x61, 0xdf, 0xa7, 0x28, 0x1f, 0x96,
0x44, 0x63, 0xd2, 0xcc, 0x35, 0xa4, 0xd1, 0xb0,
0x34, 0x90, 0xde, 0xc5, 0x1b, 0x07, 0x11, 0xfb,
0xd6, 0xf5, 0x5f, 0x79, 0x23, 0x4d, 0x5b, 0x7c,
0x76, 0x66, 0x22, 0xa6, 0x6d, 0xe9, 0x2b, 0xe9,
0x96, 0x46, 0x1d, 0x5e, 0x4d, 0xc8, 0x78, 0xef,
0x9b, 0xca, 0x03, 0x05, 0x21, 0xe8, 0x35, 0x1e,
0x4b, 0xae, 0xd2, 0xfd, 0x04, 0xf9, 0x46, 0x73,
0x68, 0xc4, 0xad, 0x6a, 0xc1, 0x86, 0xd0, 0x82,
0x45, 0xb2, 0x63, 0xa2, 0x66, 0x6d, 0x1f, 0x6c,
0x54, 0x20, 0xf1, 0x59, 0x9d, 0xfd, 0x9f, 0x43,
0x89, 0x21, 0xc2, 0xf5, 0xa4, 0x63, 0x93, 0x8c,
```



```
0xe0, 0x98, 0x22, 0x65, 0xee, 0xf7, 0x01, 0x79,
0xbc, 0x55, 0x3f, 0x33, 0x9e, 0xb1, 0xa4, 0xc1,
0xaf, 0x5f, 0x6a, 0x54, 0x7f
```

B. Sample Code

```
/*        This code illustrates a sample implementation
 *                of the Arcfour algorithm
 *         Copyright (c) April 29, 1997 Kalle Kaukonen.
 *                   All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or
 * without modification, are permitted provided that this copyright
 * notice and disclaimer are retained.
 *
 * THIS SOFTWARE IS PROVIDED BY KALLE KAUKONEN AND CONTRIBUTORS ``AS
 * IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL KALLE
 * KAUKONEN OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
```

```
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


typedef struct
{
  unsigned int x;
  unsigned int y;
  unsigned char state[256];
} ArcfourContext;

void arcfour_init(ArcfourContext *ctx, const unsigned char *key,
                  unsigned int key_len);
unsigned int arcfour_byte(ArcfourContext *ctx);
void arcfour_encrypt(ArcfourContext *ctx, unsigned char *dest,
                     const unsigned char *src, unsigned int len);


int main(int argc, char **argv)
{
  unsigned char dest[500];
  unsigned char mykey[] = {0x29, 0x04, 0x19, 0x72, 0xfb, 0x42,
                           0xba, 0x5f, 0xc7, 0x12, 0x77, 0x12,
                           0xf1, 0x38, 0x29, 0xc9};
  unsigned char src[]   = "Know thyself";

  ArcfourContext mycontext;

  /* Initialize the algoritm */
  arcfour_init(&mycontext, mykey, 16);

  /* Encrypt 13 bytes of the src string */
```



```
  arcfour_encrypt(&mycontext, dest, src, 13);

  /* Now "dest" contains the encrypted string. Do whatever
     you please with it... */

  return 0;
}
```

```
void arcfour_init(ArcfourContext *ctx, const unsigned char *key,
                  unsigned int key_len)
{
  unsigned int t, u;
  unsigned int keyindex;
  unsigned int stateindex;
  unsigned char *state;
  unsigned int counter;

  state = ctx->state;
  ctx->x = 0;
  ctx->y = 0;
  for (counter = 0; counter < 256; counter++)
       state[counter] = counter;
  keyindex = 0;
  stateindex = 0;
  for (counter = 0; counter < 256; counter++)
  {
    t = state[counter];
    stateindex = (stateindex + key[keyindex] + t) & 0xff;
    u = state[stateindex];
    state[stateindex] = t;
    state[counter] = u;
    if (++keyindex >= key_len)
      keyindex = 0;
  }
}


unsigned int arcfour_byte(ArcfourContext *ctx)
{
  unsigned int x;
  unsigned int y;
  unsigned int sx, sy;
  unsigned char *state;

  state = ctx->state;
  x = (ctx->x + 1) & 0xff;
  sx = state[x];
  y = (sx + ctx->y) & 0xff;
  sy = state[y];
  ctx->x = x;
  ctx->y = y;
  state[y] = sx;
  state[x] = sy;
```

```
  return state[(sx + sy) & 0xff];
}


void arcfour_encrypt(ArcfourContext *ctx, unsigned char *dest,
                     const unsigned char *src, unsigned int len)
{
  unsigned int i;
  for (i = 0; i < len; i++)
    dest[i] = src[i] ^ arcfour_byte(ctx);
}
```

C. Copyright Statement

This draft expires 9 December 1999.