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. This document describes an existing algorithm that satisfies this requirement.
TABLE OF CONTENTS

STATUS OF THIS MEMO..................................................1
ABSTRACT..............................................................1

1. INTRODUCTION....................................................3

2. REQUIREMENTS FOR THIS ENCRYPTION ALGORITHM..................3

3. DESCRIPTION OF ALGORITHM......................................4

4. INTELLECTUAL PROPERTY CONSIDERATIONS.........................5

5. ACKNOWLEDGEMENTS...............................................5

6. SECURITY CONSIDERATIONS........................................5

7. REFERENCES ......................................................6

8. AUTHOR’S ADDRESS...............................................6

APPENDIX..........................................................7

A. TEST VECTORS....................................................7
B. SAMPLE CODE.....................................................8
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 satisfies 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 an 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-existing protocols and IETF-based protocols.

2. Requirements for this Encryption Algorithm

The algorithm described here has been chosen because it is compatible with one of the most popular encryption algorithms in the browser market. It is potentially useful in several environments, including TLS [TLS] and IPSEC [IPSEC]. There are existing Internet Drafts that describe how it can be applied, see [TLS] and [Caronni].

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 use of 40-bit keys.

Compatibility of the algorithm with commercial algorithms is determined 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], page 397-398, in the chapter entitled "Other Stream Ciphers and Real Random-Sequence Generators".

1. Allocate an array of 8 by 8 8 bit counters as an S-box, label it
   \[ S[0] .. S[255]. \]

2. Initialize the S-box. Fill each entry first with it’s index:
   \[ S[0] = 0; S[1] = 1; \text{etc. up to } S[255] = 255; \]

3. Fill another array of the same size (256) with the key, repeating bytes as necessary.
   \[ S2[0] = \text{key}[0]; S2[1] = \text{key}[1]; ... \]

4. Initialize the S-box from it’s preloaded value and the key. Set \( j \) to zero and perform 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. For either encryption or decryption, the input text is processed one byte at a time. A 'random' byte k is generated:

Initialize i to zero; initialize j to zero.

\[
i = (i+1) \mod 256;
\]

\[
j = (j + S[i]) \mod 256;
\]

\[
temp = S[i];
\]

\[
S[i] = S[j];
\]

\[
S[j] = temp;
\]

\[
t = (S[i] + S[j]) \mod 256;
\]

\[
K = S[t];
\]

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.

5. Acknowledgements

This work was based on conversations with several colleagues 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 robust. If the key length is significantly shorter, specifically 40 bits, then there are known attacks 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 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. The experience that can be gained by developing a full implementation and deploying it may provide sufficient benefit that 40-bit "weak" encryption is appropriate. There are examples in the commercial environment where this logic has been successfully applied.
7. References


[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.


8. Author’s Address

Rodney Thayer
Sable Technology Corporation
246 Walnut Street
Newton Massachusetts 02160
rodney@sabletech.com
+1 617 332 7292
Fax +1 617 332 7970
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
B. Sample Code

[Ed. note: In the style of other IETF documents, a sample program could be included here. However this has been removed in order to comply with export controls. The source code text of Appendix B is available separately, if an appropriate repository, such as that at MIT, could be found.]