Cryptography using modular software elements*

by HERBERT S. BRIGHT and RICHARD L. ENISON
Computation Planning, Inc.
Bethesda, Maryland

ABSTRACT

Protection of information within a computer/communication system can be provided through reversible cryptographic transformation of the information itself into a form that can be returned to usable form only through use of control information known as "key."

It is not necessary, in order to achieve access control, that the encryption algorithms, random number generator, or system organization be kept secret; in fact, a basic requirement of modern cryptographic technology is that it must be effective although a would-be penetrator is assumed to have full access to all of that information and the facilities and competence to apply it. Only the key can be assumed to be, and must be, physically secure.

The building-block approach outlined makes use of pre-programmed software elements for providing all specialized algorithms, including the Proposed Federal Data Encryption Standard (DES), together with necessary nonnumeric generalized support routines for use with application programs written in conventional procedural higher languages (FORTRAN, COBOL, etc.). Both Strong Algorithm and Long Key methods can be used as required by security-level-vs-cost tradeoff considerations.

This method is useful in conjunction with specialized hardware; for testing of programs and hardware; in some cases instead of hardware; and can support multiple-level security applications.

The entire scheme, including the Tausworthe-Lewis-Payne bitwise linear recurrence modulo 2 quasirandom number generator, is based irrespective of hardware type on a standardized 64-bit data element.

INTRODUCTION

In spite of appropriate bars and locks on parts of a system: If the information it handles is in a form that can be understood, used, or damaged by unauthorized persons without such access being immediately evident to management, the information is vulnerable.

Loss of management control over sensitive information or operations is then prevented only by the integrity of system controls.

Unauthorized access may be for improper or malevolent purpose or—much more probably—may occur by accident; but in any case it is ultimately management's responsibility to avoid errors and omissions in planning that can lead to such vulnerability.

In the course of physical and logical configuration planning, management has some choices of "appropriate bars and locks" in hardware/software installation.

Protection of the information itself, through privacy transformation using cryptographic technology, provides an additional level of management control that can be relatively low in cost and high in effectiveness.

Elementary applications of cryptographic protection can include:

- User authentication (off-line)
- Terminal authentication (on-line)**
- Data link protection
- Network protection
- File access protection

The basic concept of modern cryptographic protection for information is that access to it can be limited to only those properly authorized to make such access, merely by protecting an information key.

This paper will outline a method by which preprogrammed software elements can be combined in modular fashion to provide a broad range of cryptographic transformation capabilities.

CRYPTOGRAPHY CONCEPTS

Cryptographic transformation of discrete data elements consists of applying deterministic modification processes that prevent the data from being recognized or used, or modified without such modification becom-

* This work was done in connection with software product development.

** An advanced two-way authentication scheme is outlined in Reference 1.
ing obvious. This paper will be limited to reversible crypto processes (i.e., those which are capable of subsequently reversing the encryption (i.e., "decrypting") in order to recover the original text.

**Basic ideas**

Figure 1 defines the nomenclature and shows schematically the relationship between input data (known as Plaintext or Cleartext); the encrypted data (known as Cipher or Ciphertext); the data used to control encryption and decryption (known as Key); and the encrypt/decrypt operations.

A system using the arrangement of elements shown in Figure 1, constituting the simplest and most basic implementation of cryptography, is known by the term "Electronic Code Book" or ECB. Many writers distinguish between encoding (conversion of message elements to different form using a substitution process) and enciphering (transforming message elements using an algorithmic process). Enciphering transformations include both resequencing and in-sequence conversion.

Shannon, whose basic 1949 work is fundamental to much subsequent work and is the foundation for the work reported in the present paper, distinguished between

- "Concealment Systems," also known as Steganography, in which the very existence of sensitive information is concealed;
- "Privacy Systems," in which the physical form of information is transformed and reconverted by special equipment assumed to be not possessed by unauthorized users; and
- "True Secrecy Systems," in which the information is modified only logically, and unauthorized persons can be assumed to be aware of its existence and to have any equipment needed to decrypt it into Plaintext form.

We will consider only what Shannon called True Secrecy Systems: We assume that the algorithm(s) used for data transformation, and all necessary system information, are completely public and that only the key information is physically secure; viz., that a key is known only to authorized persons who are authorized to access the information that is to be protected by cryptographic transformation. This assumption is consistent with the National Bureau of Standards' Draft Guidelines in which D. K. Branstad established the basic assumptions underlying development and use of the Proposed (U.S.) Federal Data Encryption Standard (DES).

Shannon showed that effective cryptographic transformation processes can be implemented using either algorithmic complexity or key length for achieving strength: He showed that what he called "product systems," consisting of a combination of simpler cryptographic processes, form a linear associative algebra with a unit element, thus permitting techniques to be concatenated without losing the required deterministic nature of the resulting process.

The NBS Proposed DES is an example of a Shannon Product System, consisting of an algorithm using bitwise permutation, addition mod 2, and substitution in each 64-bit data block in a highly complex sequence, involving multiple iterations through parts of the process, under control of a 64-bit Key (56 data bits plus 8 parity bits). Its basic character is outlined in gross schematic form in Figure 2, which is taken from Branstad.

Shannon represented all simple substitution processes in the form of character set arithmetic (for En-
lish alphabetic sets, arithmetic base 26); the Vernam system\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} used bit-by-bit addition modulo 2.

A simple monoalphabetic substitution process, known as a Caesar System (perhaps first introduced to some readers in the form of the Little Orphan Annie Secret Code Wheel), can be represented by

\[ y_i := x_i + u \]

where \( u \) is a constant and the \( y_i \) are the elements of a substitution encoding scheme.

It may be seen by inspection that a Caesar-encoded alphabetic message can be solved by the method of exhaustion, writing it 26 times and choosing one that seems to follow the required language syntax and semantic requirements. Such\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} processes have not been used by military people since Biblical times.

Vigenère systems\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} use a constantly-changing alphabetic substitution process, stepped synchronously with the message being transformed.

If a single transformation process is stepped to the next sequential substitution using a recirculating transformation representation (initially implemented as a paper tape loop, thus called a "single loop Vigenère system") the transformation can be represented by the sequence

\[ a_0 a_1 \ldots a_n \]

A system in which two recirculation transformations are synchronously cascaded can be represented by

\[ a_0 a_1 \ldots a_n b_0 b_1 \ldots b_n \]

It is customary to choose \( n \) and \( \gamma \) to be relatively prime.

Tuckerman\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} demonstrated that a general solution to \( n \)-loop Vigenère systems could be provided by statistical methods, and showed fully-developed breakings of single- and double-loop Vigenère systems, using real examples.

Shannon proved that a single-loop Vigenère system with infinite "loop" length is unbreakable. Such a scheme is known as "single use code." It requires that the key be at least as long as the message and that the key be kept physically secure.

The significance of the previous paragraph is that the simple Addition Modulo 2 or Exclusive Or algorithm, which is its own inverse, can be used for both encryption and decryption providing that an infinite-length key or an acceptable substitute therefor can be provided and maintained secure. We will use Shannon's designation of such a system as Vernam.

As originally published\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} the Vernam system applied bit-by-bit add-mod-2 (reversible) transformation of a binary message without changing its length.

It may be seen that a similar transformation, also reversible, can be applied through use of sequential logic by merging random noise with the plaintext information to form cipher that will be increased in length by the amount of noise inserted. Subsequent decryption will extract the binary key information and return the cipher to the original plaintext content and length.

Both bitwise operations (Vernam) and sequential logic schemes (merge) depend for their cryptographic strength on the quality of the random number stream used as the Long Key.

The long key information can be retained as physically secure, or can be generated from seed by using a short actual key to start an appropriate random number generator. A third choice is to generate random blocks of key data that can be randomly sequenced to form a pseudo-long-key binary stream. The decision between these implementation approaches should be made in consideration of the required technical performance (in the security sense) and economics of each problem situation.

In the remainder of this paper we will consider only encryption schemes that can be sufficiently strong so that they are economically infeasible to break. We will categorize them into the two broad groups implied in the above discussion:

- Those depending for their strength upon the complexity and effectiveness of a known algorithm, using a nominal-length key, will be called strong algorithm systems.
- Those depending for their strength upon the non-predictability (i.e., the random-bit quality) of a long key, used with relatively trivial algorithms, will be called long key systems.

**Strong algorithm**

Much of the substantive content of the classic Shannon paper\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} and others\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} culminating in current algorithm developments has led to what Shannon called Product Systems. Because it has been broadly published and scrutinized, and because current indications are that it satisfies the basic requirements of strength, use of short key, and generality, we suggest that the NBS Proposed DES Algorithm\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} is an appropriate archetype for Strong Algorithm schemes. Its use will be assumed in this section, with the understanding that it could be replaced by a different strong algorithm if such replacement is appropriate.

**Proposed Data Encryption Standard (DES)**

The National Bureau of Standards (NBS) has selected and published\footnote{Martin\cite{Martin} described as a particularly poor example of encryption a method that has been suggested for use with Selectric\textsuperscript{8} type writer terminals: use scrambled positions for the characters on the type element, and physically secure this removable alphabet.} a Proposed Federal Data Encryption Standard. The announcement includes the statement that:

"Data may be protected against unauthorized dis-
closure by generating a random key and issuing it to the authorized users of the data. The cipher that has been produced by performing the steps of the encryption algorithm on data using a particular key can only be returned to the original data by use of the decryption algorithm using the identical key. Unauthorized recipients of the cipher who may have the algorithm but who do not have this key cannot derive the original data. A standard algorithm based on a user-generated key thus provides a basis for compatible cryptographic protection of computer data while preventing unauthorized use of the data in cipher form.” (The Proposed Standard also states that “Only hardware implementations of the algorithm . . . will be considered as complying with the standard.”)

Several manufacturers are developing LSI-chip-based hardware for implementation of the DES algorithm. We are not aware that any of this hardware has been released for sale as of the date of submission of this paper; however, we assume that such release will occur shortly after official confirmation of the Federal Standard.

We have developed exact software Emulators of the DES algorithm for several kinds of hardware, as part of a generalized nonnumeric software element support package. The emulators maintain, and will support package. The emulators maintain, and will occur shortly after official confirmation of the package provide capabilities for data preparation, testing and other auxiliary functions without interrupting on-line availability.

NBS has developed two sets of test data for validating implementations of the algorithm:

- What we have called Test A is a set of 24 64-bit key/data pairs designed to demonstrate the power of the algorithm by showing the large effects on cipher of small changes in either datum or key, and the behavior of the encryption process on a variety of bit patterns.
- What we have called Test B is a set of 19 key/data pairs generated as pseudorandom numbers and chosen because the corresponding 19 encryptions reference at least once all of the 512 entries in the “S-box” substitution cipher tables. These 19 pairs were found experimentally by NBS and independently confirmed by us. Correct execution of Test B (in which all cipher produced correspond to presumably correct cipher from a different implementation, and in which all of the S-box table entries are referenced at least once) provides a high order of confidence in the correctness of encryption for any values of key/data.

Result of exercising one of the DES Emulators executing 15 examples from A and B, as described above, is given in Figure 3. In this test output the column headed “CIPHER (ENCRYPTED)” gives the result of machine encryption; the column headed “DECRYPTION OF CIPHER” gives the result of the machine decryption of the machine-produced cipher to recover the original plaintext. All keys and data are shown in hexadecimal notation to correspond with NBS examples.

For all key/data pairs of Tests A and B, the Emulators give results identical to those given by NBS. From Test A (the first 12 rows of Figure 3), it may be seen that change of a single bit value or a 1-bit shift of a subpattern position within datum or key provides essentially complete change of encrypt/decrypt results. This requires complete accuracy of transmission; conversely, the process can thus be used to display with great sensitivity even slight errors in data entry or transmission.

To show this effect, we repeated Example A9 with the 5th and 6th data digits interchanged (BC becomes CB). Note that the resulting cipher (for Example A25) shows no evident resemblance to the cipher for A9.

Example A26 is a demonstration of the odd-parity check (optional under the Proposed Standard); it consists of Example A10 modified by giving the first eight key bits even parity.

The general-distribution versions of the DES Emulators use medium-speed, medium-space techniques. NCRYPT/DCRYPT requires less than 9000 bytes on 360/370 or 2200 words on 1108. Execution speed is over 100 encrypts or decrypts per second with machines in the 370/155 or 1108 class.

It may be seen that the strong-algorithm process, if executed entirely by software, will be economical for fairly small data volumes but will be costly for large-volume applications (such as, e.g., encrypting all but control elements of continuous high-speed data streams or sizable data bases). For large-scale applications, the long-key methods outlined in the next section (after either a time delay for about a million machine instructions executed in initial (from seed) startup of the high-performance random number generator provided, or time to load a 1042-word restart table) operate orders of magnitude faster than the Emulator.

Applications of the software DES emulator include:

- Testing of application ideas and methods before hardware is available;
- Debugging and production testing of programs independently of on-line hardware;
Long-key systems

Any long-key system depends upon the use of a key stream that must be assumed to be physically secure.

The key stream can be provided in two basic ways:

(a) It can be generated from "seed" (i.e., a key unique to that key stream) in a fully-deterministic process when used for either encryption or decryption or both; or

(b) It can be generated, stored, copied as may be required, and played back from the stored form when and where needed.

In case (a), decryption of a particular ciphertext data stream requires that a key-stream generator (software or hardware) identical to that used in encryption of that stream must be available at the time and place of key stream entry for decryption.

Physical security must be provided for the key to be used: in case (b), this will require security of tape or diskpack; in case (a), only the seed need be kept secure. Obviously, physical and geographical considerations will affect the generate-or-playback choice.

An important consideration is that long messages would be in hazard from even trivial communication or other hardware errors; loss of absolute synchronization would generate chaos, effectively preventing recovery of plaintext beyond the point at which, say, a one-bit loss occurred.

The very vulnerability of ciphertext, as noted here, provides a potentially useful and highly sensitive detection scheme: even small errors in ciphertext transmission will result in gross and obvious format and other changes in decrypted output.

Use of message blocking (we have chosen 64 bits as the standard block length corresponding to DES practice\textsuperscript{4}), and careful block numbering and accounting, prevents loss of more than a single block for a single small error and provides an audit trail for recovery.

As noted in References 3 and 4, validity of the encryption process as secure in and of itself, depending only upon key security, requires that the system not depend upon secrecy of an algorithm or of hardware/software configuration.

With either of the long-key methods discussed below, in order to maximize security and integrity of the encryption process, "leakage" of key stream control information elements should be inhibited in spite of the simplicity of the actual encryption algorithms. Thus, it is desirable that the appearance of cyclic or unchanging (i.e., transparent) bit patterns in plaintext (see, e.g., the all-zeroes and alternate-ones data of Examples A10 and A13 of Figure 6) be suppressed by compression or other means.

Long-key generation from seed

One effective means of key stream generation is a process that produces uniformly distributed random numbers. The process used here is a computer program quasirandom number stream generator of the

- Preparation and evaluation, by manufacturers, of hardware design test data;
- Testing (both validation and maintenance) of installed hardware;
- Operational encryption/decryption where the Federal Standard is not applicable; and
- Bidirectional authentication procedures (See, e.g., Reference 1).
Tausworthe-Lewis-Payne bitwise linear recurrence modulo 2 type, of which the developmental background and characteristics are outlined in the Appendix.

Chaitin expressed proof that a truly random string cannot be specified by an information string shorter than itself. It may be inferred that a perfect single-use code cannot have its key stream generated from a short seed expanded by any program of limited complexity.

We believe, however, that despite its finite complexity a publicly known quasirandom number stream generator of long period and good statistical performance, such as that described in the Appendix, when operating on a secret seed, can produce a sufficiently close approximation to a random key stream so that the fundamental objective of the present paper is met: the resulting encryption will be economically infeasible to break.

Figure 4 shows schematically the process by which the quasirandom number generator produces a 64-bit block key stream starting from a 64-bit (or less) seed, continually accessing a recirculating seed matrix of about a thousand words as outlined in the Appendix.

Initial startup of the generator from a seed requires a half million machine instructions to be executed, after which the process proceeds at a speed corresponding to only a dozen machine instructions executed per number generated.

Restart of the process can be accomplished readily through reloading of the recirculating seed matrix. Backwards re-generation from a previous checkpoint, as is required for some system problems, offers no difficulties; the generator can operate in either direction. Pre-iteration count is a user-controlled parameter.

For encryption purposes this Tausworthe-Lewis-Payne generator offers the advantage that a large amount of information (521 64-bit blocks) would be needed to initiate or restart a quasirandom string. There is no explicit way of identifying the initially-used element of the generated string, other than knowing the actual seed and the starting iteration count. The fact that both seed and period are of great length permits both Vernam and merge/extract encipherments to have considerable cryptographic strength.

On a 360/65, after restart (time to load the current seed string matrix) or startup (2.3 seconds for a pre-iteration count of 20,000p), the generation of 64-bit unsigned quasirandom integers takes 24.1 milliseconds per thousand numbers.

The period of this Tausworthe-Lewis-Payne type generator is essentially infinite (viz., $2^{521} - 1$). Its gross performance has been checked in dimensionality up to 8-distributivity. Small changes in seed cause large changes in generated key stream.

**Long-key Vernam (bitwise)**

The classical Vernam single-use-code concept, implemented explicitly, is shown schematically in Figure 5.

As noted previously, this scheme does not change message length.

The building-block approach uses prefabricated software elements for executing the encryption and control logic and for data preparation and testing.

A Vernam test is shown in Figure 6. This test uses as plaintext input data, read from cards, the 15 examples used in Figure 3 above to test the DES Emulator, consisting of 13 examples of NBS "Tests A and B" plus slight modifications of two of the Test A examples. The key stream was generated by the TLP quasirandom number generator from the 64-bit seed 012357BD14905694. Speed of this encryption or decryption on a 360/65 is one 64-bit block per 12.3 microseconds.

**Long-key sequential logic**

The capability for controlled merging and extracting of noise into and from binary information streams has several potential uses and should be considered in a comprehensive encryption plan:

- Provide a high level of cryptographic protection when used with a long quasirandom number control stream and appropriately generated noise data;
- Provide an intermediate level of cryptographic protection when used with shorter or recycling control stream and a noise data stream of any length;
- Suppress redundancy or cyclic patterns in plaintext or in ciphertext, for the purpose of raising the cryptographic strength of DES or Vernam-type high-level cryptography systems; and
- Permit, in combination with the DES and/or Vernam techniques, the implementation of multiple-level cryptography systems of controllable...
Cryptography Using Modular Software Elements

VERNAM ENCIPHERING USING QUASIRANDOMLY GENERATED KEYS.
DATA ARE FROM NBS TESTS A & B. SEED IS 0123578014905694.

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>KEY STREAM</th>
<th>DATA</th>
<th>CIPHER (ENCRYPTED)</th>
<th>DECRYPTION OF CIPHER</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 4</td>
<td>E514844CA3584619</td>
<td>05730C5206B37492</td>
<td>E063FF8746D088</td>
<td>05730C5206B37492</td>
</tr>
<tr>
<td>A 10</td>
<td>5A0E160489EF784</td>
<td>0000000000000000</td>
<td>5A0E160489EF784</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>A 11</td>
<td>975EF992FF56F61</td>
<td>0000000000000000</td>
<td>975EF992FF56F61</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>A 12</td>
<td>25FB4C258466EC4</td>
<td>1111111111111111</td>
<td>34E49D949B77AD8</td>
<td>1111111111111111</td>
</tr>
<tr>
<td>A 13</td>
<td>4520C8697619123</td>
<td>AAAAAAAAAAAAAAA</td>
<td>0F8756C3UC53689</td>
<td>AAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>A 14</td>
<td>06E666DLO154071</td>
<td>7A42CC764AD82DA</td>
<td>7A42CC764AD82DA</td>
<td>7A42CC764AD82DA</td>
</tr>
<tr>
<td>A 16</td>
<td>933314DLO3309533</td>
<td>C66618BB66C06</td>
<td>C66618BB66C06</td>
<td>C66618BB66C06</td>
</tr>
<tr>
<td>A 18</td>
<td>489F88E46993104</td>
<td>101CAD3B8CA6C01</td>
<td>101CAD3B8CA6C01</td>
<td>101CAD3B8CA6C01</td>
</tr>
<tr>
<td>A 19</td>
<td>601C1064C7AFBEC4</td>
<td>3546931921AD891</td>
<td>3546931921AD891</td>
<td>3546931921AD891</td>
</tr>
<tr>
<td>A 20</td>
<td>104D3A3D11AA25D0C</td>
<td>4B8F8644FF7E88</td>
<td>4B8F8644FF7E88</td>
<td>4B8F8644FF7E88</td>
</tr>
<tr>
<td>A 21</td>
<td>CE8FED1BO55941B</td>
<td>0573C85206B37492</td>
<td>0573C85206B37492</td>
<td>0573C85206B37492</td>
</tr>
<tr>
<td>A 22</td>
<td>EE62F743L1900777</td>
<td>C6CC26135B80640</td>
<td>C6CC26135B80640</td>
<td>C6CC26135B80640</td>
</tr>
<tr>
<td>A 23</td>
<td>EA085F4BRL9590AC</td>
<td>0141D65039776742</td>
<td>0141D65039776742</td>
<td>0141D65039776742</td>
</tr>
<tr>
<td>A 24</td>
<td>ED8926F361930D77</td>
<td>1686A32818634C4</td>
<td>1686A32818634C4</td>
<td>1686A32818634C4</td>
</tr>
<tr>
<td>A 25</td>
<td>D9326928B0728</td>
<td>30553228606F295A</td>
<td>E943D17120D7E71</td>
<td>30553228606F295A</td>
</tr>
</tbody>
</table>

Figure 6—Vernam encryption/decryption results

complexity and with controllable strength/cost tradeoff at each level.

The encryption and decryption processes can be controlled from fixed mask (MASK=constant); lists or tables of masks (MASK=variable); or from a recycling or open-ended mask stream, with corresponding levels of complexity and security.

Use of a constant merge control mask produces weak cipher that has recognizable patterns, especially when used with simple bit pattern plaintext and examined in binary form; a variable mask produces higher-grade cipher.

A sequential logic test is shown in Figure 8. In this case a constant quasirandom 64-bit key (actually one of the generated keys, chosen because it happens to contain exactly 32 one-bits and 32 zero-bits, generated from the same seed used in the Vernam test above) is used as the control mask for noise (the same 15 generated quasirandom keys) to be merged with plaintext to form a kind of cipher. This mask, because of the 50/50 one/zero mask ratio, produces cipher just twice the length of the plaintext. The two sequential 64-bit cipher blocks produced from each plaintext block are shown together in the double column headed CIPHER (ENCRYPTED). The decrypted plaintext, recovered by extraction of the noise stream under control of the same mask, is shown in the right-hand column labeled DECRYPTION OF CIPHER.

The requirements for bitwise testing and processing make this sequential process slower than the Vernam process.

PROGRAM PROTECTION

The protection of proprietary programs from unauthorized use, copying, or alteration is a largely unexploited area of application of cryptography that offers fascinating opportunities for strategy and counter-strategy development.

Protection technology for file and system access, as outlined previously, provides the basic mechanisms for program transformation, detection of alteration, and audit trail development.

A basic decision that must be made at the outset of development of a program protection method is whether personnel and organizations having some level of authorization for system software maintenance are to be considered part of the world against which protection (from errors and omissions as well as malfeasance) is sought. If the answer is yes, the problem set escalates in difficulty and reasonable bounds on the protection objectives must be established.

APPENDIX

Background for a 64-bit quasirandom number string generator

The “mid-squares” random number generators used by pioneers Von Neumann and others in the late 1940s rapidly degenerated (to zeroes or cycles of short period) when used for long string generation. Like physical (e.g., electronic noise) generators, they were
unsuitable for many modern computing purposes. They were largely displaced by generators (programs) of many designs using congruential methods.

Generators of one type, suggested by Lehmer\(^\text{19}\) in 1961 and known as "linear congruential", can be described by the recursive formula

\[ x_i = (a x_{i-1} + c) \mod m, \]

where \(x_i\) is the sequence generated. The parameters are the seed \(x_0\), the multiplier \(a\), the additive constant \(c\) and the modulus \(m\). This is called the multiplicative congruential method when \(c=0\), and the additive method when \(m=2\). A large value of \(m\) is needed to avoid trivial sequences with small period. For example, if \(m=2\), the sequence is either a constant or an alternating 0, 1, 0, 1, \ldots. Uniform numbers between 0 to 1 are obtained by \(u_i = \frac{x_i}{m}\).

Generators of a second type, known as "additive congruential," can be described by

\[ x_i = x_{i-q} + x_{i-p} \mod m. \]

Both are special cases of the general linear congruential method:

\[ x_i = a_1 x_{i-1} + a_2 x_{i-2} + \ldots + a_p x_{i-p} + c \mod m. \]

The additive method has properties sufficiently different from the linear as to make it worth studying separately. A large modulus is not needed to ensure a long period; arbitrarily long period sequences can be generated with \(m=2\), provided judicious choices are made for \(p\) and \(q\).

Both linear and additive methods are described in Knuth Volume 2.\(^{19}\) Knuth observes that a good sequence of random numbers can be obtained by the additive congruential method, or more generally, by

\[ x_i = a_1 x_{i-1} + \ldots + a_p x_{i-p} \mod m. \]

The additive method has properties sufficiently different from the linear as to make it worth studying separately. A large modulus is not needed to ensure a long period; arbitrarily long period sequences can be generated with \(m=2\), provided judicious choices are made for \(p\) and \(q\).

Figure 8—Sequential encryption/decryption: Tests A and B

The Xi's are integers from 0 to \(m-1\). A large value of \(m\) is needed to avoid trivial sequences with small period. For example, if \(m=2\), the sequence is either a constant or an alternating 0, 1, 0, 1, \ldots. Uniform numbers between 0 to 1 are obtained by \(u_i = \frac{x_i}{m}\).

Generators of a second type, known as "additive congruential," can be described by

\[ x_i = x_{i-q} + x_{i-p} \mod m. \]

Both are special cases of the general linear congruential method:

\[ x_i = a_1 x_{i-1} + a_2 x_{i-2} + \ldots + a_p x_{i-p} + c \mod m. \]

The additive method has properties sufficiently different from the linear as to make it worth studying separately. A large modulus is not needed to ensure a long period; arbitrarily long period sequences can be generated with \(m=2\), provided judicious choices are made for \(p\) and \(q\).

Both linear and additive methods are described in Knuth Volume 2.\(^{19}\) Knuth observes that a good sequence of random numbers can be obtained by the additive congruential method, or more generally, by

\[ x_i = a_1 x_{i-1} + \ldots + a_p x_{i-p} \mod m. \]
number of times a k-tuple \((x_{k1}, x_{k2}, \ldots, x_{ki+k-1})\) of members of the sequence falls into a given k-dimensional cell. If this test is passed, the sequence is said to be k-distributed. It is not practical to divide the coordinate axes into too many subintervals when k is large, say, \(k > 3\). Other tests of k-distributivity are:

2a) Poker Test: One considers groups of k members of the sequence and counts the number of distinct values represented in each group.

2b) Maximum [Minimum] of k: One plots the distribution of the function \(\max \{\min\} (x_{k1}, \ldots, x_{ki+k-1})\).

2c) Sum of k: This is similar to tests described by Knuth. The same tests can be made for any function of k numbers, provided one can calculate an expected distribution for the function. This test is explicitly mentioned by MacLaren and Marsaglia as well as by Lewis and Payne, who refer to it as the Yules Test. Several authors use \(\sum x_i^2\), which is commonly called "the d² test".

3) Gap Test: One plots the distribution of gaps in the sequence of various lengths, i.e., consecutive members \(x_1, x_2, \ldots, x_{i+k-1}\), such that all members fall into a given interval, while the immediately preceding and succeeding members do not. A special case is where the interval is the set of numbers above (or below) the mean, in which case it is called the Runs Above (Below) the Mean Test.

4) Runs Test: One plots the distribution of maximal ascending (descending) runs of various lengths. This test (now called the Runs Up/Runs Down Tests) was mentioned by Moshman in the first random number generator paper ever published in an ACM serial.

5) Coupon Collector's Test: One studies the order relations between the members of the sequence in groups of k. Each of the \(k!\) possible orders should occur about equally often. If the universe is large, the probability of equality is small; otherwise, equal members may be disregarded.

6) Permutation Test: One studies the order relations between the members of the sequence in groups of k. Each of the \(k!\) possible orders should occur about equally often. If the universe is large, the probability of equality is small; otherwise, equal members may be disregarded.

7) Serial Correlation Test: One computes the correlation coefficient between consecutive members of the sequence. This gives the serial correlation for lag 1. Similarly, one may get the serial correlation for lag k by computing the correlation coefficient between \(x_i\) and \(x_{i+k}\). This is to show that the members of the sequence are independent.

Other tests have been proposed and used. Lewis and Payne ran a Conditional Bit Test, which tested the independence of each bit in a string from the others, as well as a Fourier Transform Test, also used by Covneyou and McPherson in 1967 and by Lewis, Goodman, and Miller. The latter was facilitated by the Fast Fourier Transform algorithm introduced by Cooley and Tukey.

All of the pre-Tausworthe papers referenced here proposed some sort of linear congruential generator. Those who subjected their generators to exhaustive tests admitted that some of the tests failed. Some only ran the simplest tests (e.g., equidistribution and serial test for pairs), and passed; however, in typical cases the same generators were later shown to fail some other test. The generators differed mainly in the choice of such parameters as multiplier, additive constant, and modulus. The modulus was usually the largest number that could be represented on some machine (most often 2³⁵), or a prime less than that number. The present problem required a 64-bit random number generator, which none of the above papers considered. Many authors seemed to choose various parameters arbitrarily and pick the ones that passed the most tests, although in some cases broad guidelines were given (but, in general, were not shown to be useful). One formula giving an approximation to the serial correlation of a linear congruential sequence was published by Coveyou. In certain cases his approximation was a poor one, but Greenberger corrected this flaw by adding another term. Some theoretical and empirical work by Marsaglia in 1968 showed that all linear congruential generators suffered from poor higher-dimensional distributions. There was then little hope of producing one generator that passed all tests.

A paper by Martin-Löf showed, using methods of recursive function theory, that there was a universal test for uniform random number sequences and that almost all sequences (in the sense of measure theory) passed it. Although this paper is quite abstract and not of much practical use in constructing a good random number generator, it did provide some hope that it could be done.

Knuth mentioned a property that seems to come close to the concept of a universal test, namely, complete equidistributivity, or \(\pi\)-distributivity: If a sequence is k-distributed, it is \(\pi\)-distributed for \(r < k\). A sequence \(\pi\)-distributed if it is k-distributed for all k.

A sequence that is \(\pi\)-distributed passes all the other tests we have considered. Such a sequence was constructed by Knuth, but, as he observed, his sequence is not of much use in machine generation of random numbers because it takes too long to converge to the desired properties. It was formed by starting with a short 1-distributed sequence, followed by a slightly longer 2-distributed sequence, etc.

The papers after Tausworthe's own paper that describe Tausworthe sequences show increasingly good results: it seems that such sequences can be constructed to pass more of the tests than the linear congruential generators. Whittlesey showed that linear congruential generators that had passed other
tests failed some autocorrelation tests he performed
with lags from 1 to 50. These tests are related to the
serial correlation test.

The Tausworthe sequence passed Whittlesley's test;
in fact, it can be shown to pass these and many other
tests on purely theoretical grounds. We found such
analytical support to be lacking in the papers proposing
linear congruential generators. Tootill, Robinson and
Adams showed that a Tausworthe sequence had good
Runs Up and Down properties, and Tootill, Robinson
and Eagle showed that a Tausworthe sequence they
generated was indistinguishable, by empirical tests,
from one that was $\infty$-distributed. It appears to be consensual among recent authors
that, in generating a Tausworthe sequence, it is best to
use a trinomial $x^p + x^{10} + 1$ whose degree $p$ is such that
$2^p - 1$ is relatively prime to various parameters of the
tests. Also, better results are obtained if decimation is
used, which means that one does not pick consecutive
groups of bits from the sequence but rather spaces
them out; and the amount of spacing should be rela­tive to
$2^p - 1$.

One would expect the best results from a Tausworthe
generator if the degree $p$ were such that $2^p - 1$ were it­self a prime number. Such primes are called Mersenne
primes, and $p$ is called a Mersenne exponent. A table
of primitive trinomials whose degree is a Mersenne
exponent was published by Zierler. It included the 23
then known Mersenne exponents, which form a con­secutive set, the largest of which is 11213. The largest
one having a primitive trinomial is 9689.

The most promising generator we have found seems to be the one described by Lewis and Payne. They use a technique that appears to be superior to decima­tion: to generate $r$-bit numbers, each of the $r$ bits is
chosen from a different part of the same Tausworthe
sequence with a constant gap between bits. Lewis and
Payne suggest that a gap of at least 100p should be
sufficient. They also suggest that the sequence will be
k-distributed for $k \leq rp$.

Our Tausworthe-Lewis-Payne (TLP) generator uses the
trinomial $x^{64} + x^{3} + 1$ to generate 64-bit numbers
which are 8-distributed and have good k-distributivity
for $k > 8$. The period of the sequence is $2^{64} - 1$.

The degree-521 TLP generator requires 521 bits
(not all zero) to start, which must somehow be ex­panded
from the initial 64-bit seed. If these 521 bits are “not very random”, the next few members of the
sequence will also have this weakness; however, Lewis
and Payne suggest that about 5000p (here, 2.6 million)
bit-iterations of a Tausworthe generator should sup­press such non-randomness. Our initial statistical tests
appear to confirm that suggestion if iteration count is
increased to 20,000p. We have considered but not
implemented use of the primitive trinomial $x^{16} + x^{14} + 1$ with the TLP method for generating the seed string.

This two-level TLP may reduce the number of iter­ations required.

We have used the prefix “quasi-” which, according to
Webster's 3ID, means “seemingly, almost” together with
the word “random” to identify effective generators.
Most authors have used the prefix “pseudo-” which, per Webster, connotes “false, sham, feigned,
fake, counterfeit, spurious”. Subsequent test results
have shown, alas, that “pseudo” was often an app­ropriate descriptor. It is hoped that the performance of
the generator described here will prove, on further
testing, to have justified our use of the term “quasi­random number generator”.

We gratefully acknowledge discussions on applicabil­ity of congruential methods with Juncosa whose
work was used by many subsequent authors, and on
primitive trinomials with Tausworthe whose basic
concepts underlies the most promising current develop­ments.

REFERENCES

8. Feistel, H., Cryptographic Coding for Data-Bank Privacy.