USING ALGORITHMS AS KEYS IN STREAM CIPHERS

Neal R. Wagner[1]
Paul S. Putter
Marianne R. Cain

Drexel University
Mathematics and Computer Science
Philadelphia, PA 19104 USA

ABSTRACT.

This paper discusses the use of an arbitrary bit-sequence generating algorithm as the cryptographic key for a stream cipher. Emphasis is placed on methods for combining stream generators into more complex ones, with and without randomization. Threshold schemes give a generalization of many combination techniques.

1. INTRODUCTION.

Some years ago Chaitin [Cha66] [Cha69] [Cha75] and Kolmogorov [Kol65] gave a remarkable strong definition of a random bit sequence: a random sequence of length n requires an algorithm nearly of length n to generate it. Martin-Löf [Mar66] suggested that infinite random sequences should withstand all recursively enumerable statistical tests, and then proved that these sequences satisfy the Chaitin-Kolmogorov definition. Such a sequence is said to have no *succinct representation.* In general, it is an undecidable problem to find the shortest algorithm that will generate a given sequence, and even finding a succinct representation, assuming one exists, is believed to be an intractable problem.

Interesting weaker definitions of pseudo-random sequences have recently been proposed [Yao82] [Blu84] [Ko84], and this is now an active research area. See [Kra84] for a survey.

In cryptography one wants to generate a long pseudo-random bit sequence from a succinct secret key. It is a short step to think of using an arbitrary bit generating algorithm (= succinct representation) as a "key" for cryptographic use. This idea goes against the conventional practice that the cryptographic algorithm itself should not be secret, only the particular key used by the algorithm.

[1]Research supported in part by NSF grant DCR-8403350 and by a Research Scholar award from Drexel University.

F. Pichler (Ed.): Advances in Cryptology - EUROCRYPT '85, LNCS 219, pp. 149-155, 1986.

2. ALGORITHMS AS CRYPTOGRAPHIC KEYS.

We propose a keyspace consisting of *all possible* algorithms for generating a bit stream. (By definition, the algorithm must not execute indefinitely at some point without producing a bit.) We want an arbitrarily long bit stream, so if the algorithm halts, repeat the initial sequence indefinitely. In practice we do not want the algorithm to be too long, but we place no upper bound on its length. Similarly, we favor algorithms that produce n bits in $O(n)$ time, but there is no such requirement. At the CRYPTO 84 meeting, in a slightly different context, it was argued that the keyspace was not really infinite, since one could always place a bound on the key size. In practice an opponent must choose a bound that is not ridiculous, and then he always faces the possibility that the size of the key exceeds this bound.

We would like to list several advantages of using algorithms as keys.

- Since the key can be an arbitrarily complex algorithm, one can never be certain that a given key has been discovered, no matter how much plaintext-ciphertext has been matched up using some algorithm. (The actual key might say, "After the 10000000^{th} bit, use this other algorithm.")
- There is no bound on the size of the key or on the number of keys -- there are infinitely many possible keys.
- This method immediately adapts to the best current technology for generating secure bit streams.
- One can adjust the key length and key complexity to the desired level of security.
- This contains every other stream cipher based on xor with a pseudo- random sequence as a special case.

We now come to the practicalities of choosing a key (= algorithm). We do not want to choose a "random" algorithm. In fact we do not know any reasonable way to make such a choice. Besides, a random choice might be unacceptably inefficient or it might not be at all secure. Note that the opponent does not necessarily know what level is "unacceptable." Knuth [Knu81, p. 4] nicely illustrates the dangers of using a "random" algorithm for a pseudo-random number generator.

In fact it is an undecidable problem to tell whether a candidate for a key is actually an algorithm, i.e., whether it does not execute indefinitely at some point without generating any more bits. Notice that this is more of a problem for the opponent attempting cryptanalysis than for the person generating the key. The opponent cannot with certainty eliminate such candidate algorithms, while the person generating the key has no such problem.

Instead of a random choice, we propose starting with "prime" bit stream generators that seem strong (Section 3.1) and propose combining them in ways that seem strong (Sections 3.2 and 3.3). Section 3.3 also includes randomization techniques as well as combination methods. It is important that the notation allow the specification of an arbitrary algorithm, so that the opponent cannot rule any out.

3. CASCADES.

For high security in conventional cryptography, it is natural to think of combinations of drastically different cryptosystems -- perhaps a composition of block ciphers, or the exclusive or of stream ciphers, or some other combination. We will refer to such combinations as *cascades* . With one model there is a proof that the cascade of two ciphers is at least as hard to break as either individually [Eve84]. We will be focusing on stream ciphers and on ways of combining and enhancing them.

3.1 BASIC PSEUDO-RANDOM BIT-STREAMS.

Pseudo-random bit sequences are the "prime parts" from which one might build up strong stream ciphers. Two or more such sequences can then be combined in various ways as described in Sections 3.2 and 3.3 below.

There has been a great deal of work recently on appropriate definitions of pseudo-random sequences and on means for constructing cryptographically secure examples [Blu84] [Sha83] [Yao82]. From our point of view, one should just have a repertory of families of pseudo-random streams -- to be augmented as new ones become available.

3.2 CASCADES WITH NO EXPANSION.

This section presents methods of combining or enhancing bit-stream ciphers without any randomization or expansion. We will use notation for figures similar to that in [Riv83]. In particular R stands for a true random sequence, P for a pseudo-random sequence, and B for a source bit-stream being modified.

(a) *Exclusive or.* This is the most common method for producing cascades of bit-stream ciphers.

(b) *Bitwise addition, with carry.*

(c) *Pseudo-random deletions.* Here the output of one pseudo-random stream is used to delete bits from another stream. For example, P = "01010101 ..." would delete every other bit. Even simple alternating deletions was mentioned as a possible strong method for preventing cryptanalysis of a string given by the expansion of an algebraic number [Kan84].

(d) *Pseudo-random alternation between streams.* Use the output of one stream to select from n streams. (See Figure 1.)

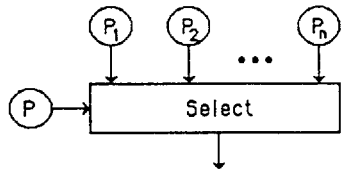

Figure 1.

As a special case, if $n = 2$ and P = "01010101 ...", then this just gives alternation between the two bit streams P_1 and P_2. If $n = 2$ and P starts with 10000000 0's followed by all 1's, this method just switches from P_1 to P_2 after 10000000 bits.

(e) *Pseudo-random selection from a buffer.* This scheme is similar to the previous one and is inspired by [Knu81, p. 32].

3.3 CASCADES WITH RANDOMIZATION AND EXPANSION.

(a) *Pseudo-random interspersing of random bits* . This simple randomization method can be used at any stage of the encryption process. See Figure 2, and for terminology refer to [Riv83] and the beginning of Section 3.2. This will expand the bit stream B by about a factor of two.

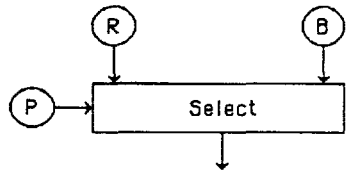

Figure 2.

(b) *Random interspersing of random bits* . This is a technique from [Riv83], shown in Figure 3. It also has an expansion of about two.

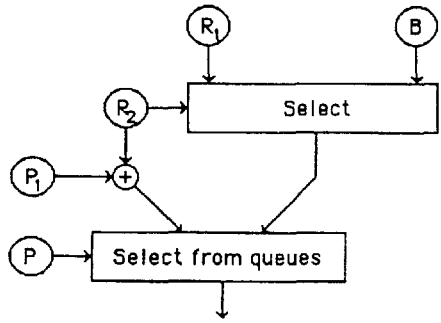

Figure 3.

Here the final selection based on the pseudo-random sequence P is just concatenation in [Riv83]. If P is the alternating sequence "01010101 ...", then we would get concatenation by alternating bits. If an arbitrary pseudo-random stream is used for P, we expect one of the streams selected to get ahead of the other, so buffers will be needed for these streams. On the average, after n bits of a random stream, either 0's or 1's will be ahead of the other by a quantity asymptotic to $\sqrt{n}$, so there is no upper bound on the necessary buffer size even in the average case. For this reason we might want to use a stream P in which the excess of 0's or 1's is never more than some fixed number which we could use as our buffer size.

(c) *Block-oriented randomization in the stream setting.* One can translate the block-oriented methods of [Riv83] by replacing a block encryption step with the exclusive or of a pseudo-random bit stream, and by replacing concatenation with selection based on a pseudo-random bit stream P. Six of the translated schemes in [Riv83] are special cases of Figure 4.

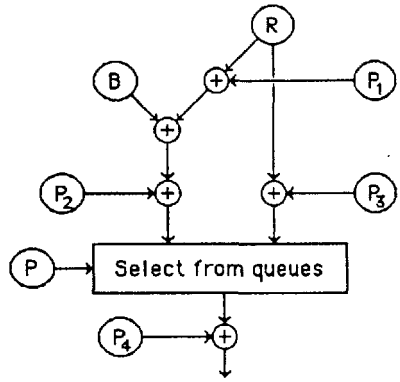

Figure 4.

Four of these special cases are obtained by deleting three out of four of the P_i's. One might also use two, three, or all four of the P_i's. All such schemes have an expansion factor of two.

Suppose an opponent can obtain these individual selected streams. This would be the case for example if P_4 were missing and P were simple alternation. The opponent could then take the exclusive or of the two streams and obtain $B \oplus P_1 \oplus P_2 \oplus P_3$. So we might as well use an xor of the streams with no randomization. In the block cipher setting, these techniques do enhance security even with some simple form of concatenation.

The provably secure but impractical *Rip Van Winkel* cipher [Mas85] is obtained as a very special case of Figure 4 by deleting all four of the P_i's and by letting P start with an enormous number of 1's, followed by alternating 0's and 1's.

(d) *Asmuth-Blakley scheme.* In the stream setting, the Asmuth-Blakley scheme for combining two cryptosystems [Asm81] takes the form of Figure 5.

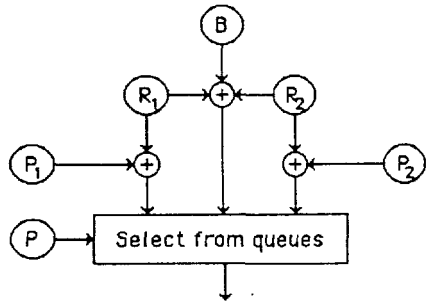

Figure 5.

The output will have an expansion factor of three. The selector stream P is a stream of ternary digits. As before, if an opponent can untangle the three selected streams then he can take the exclusive or of them to obtain $P_1 \oplus B \oplus P_2$. So again one might as well take an xor of the three streams.

(e) *Threshold schemes.* Rivest and Sherman suggested using threshold schemes in randomization [Riv83]. The methods discussed as items (c) and (d) above are just special cases of a single more general threshold scheme. In what follows we only have a need for the special case of a (k,k) threshold scheme. Threshold schemes do the opposite of combining several streams into one. They allow one to split a stream into several parts and allow later recombination and recovery of the original stream.

For example, the one-time pad and example (c) above are both essentially (2,2) threshold schemes in which the source bit-stream B is broken into two shadows: R and $R \oplus B$, for some random stream R. Both shadow streams are required to recover the source, and an opponent who learns one stream still has no information about the other. (This threshold scheme provides *Shannon perfect security* [Bla81].) Most of the methods of (c) involve encrypting one or the other of these shadows, or of encrypting the concatenation of the two shadows.

Similarly example (d) above breaks B into three shadows: R_1, R_2, and $R_1 \oplus B \oplus R_2$. This is a (3,3) threshold scheme in which all three shadow streams are necessary and sufficient to recover the source B. (Here again, knowing any two of the streams gives no information about the third.) Asmuth and Blakley encrypt (in block mode) R_1 and R_2, though it would also make sense to encrypt any two or all three of these shadows.

Now suppose we have a (k,k) threshold scheme. An implementation that is a generalization of the above examples employs k-1 random streams and the xor of these with the message stream for the k^{th} stream. Alternatively one could use another (k,k) threshold scheme, such as, for example, Shamir's Langrange interpolation threshold scheme [Sha79]. With both these methods, any k-1 out of the k streams give no information about the source stream, and again one has Shannon perfect security up to threshold k. The first has an expansion factor of k and the second somewhat greater.

We would normally expect to encrypt one or more of these k shadow streams. Even if an opponent can decrypt all but one of the encrypted shadows, he would still have no information at all about the original stream. Intuitively, this scheme is at least as strong as any of the component encryption schemes used.

There also exist (k,k) threshold schemes with less data expansion, though without at least k-fold expansion we can no longer say that k-1 shadows give *no* information about the source. In [Bla84] a spectrum of threshold schemes is described which includes the extreme special cases of k-fold expansion with perfect security, and little or no expansion with only a small measure of security.

REFERENCES.

[Aho74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "Design and Analysis of Computer Algorithms," Addison-Wesley, 1974.

[Asm81] C. G. Asmuth, and G. R. Blakley, "An efficient algorithm for constructing a cryptosystem which is harder to crack than two other cryptosystems," *Comps. and Maths. with Applications* 7 (1981), pp. 447-450.

[Bla79] G. R. Blakley, "Safeguarding cryptographic keys," *Proc. NCC* , Vol. 48, AFIPS Press, 1979, pp. 313-317.

[Bla81] G. R. Blakley, and L. Swanson, "Security proofs for information protection systems," *Proceedings of the 1981 Symposium on Security and Privacy* , IEEE Computer Society, 1982, pp. 75-88.

[Bla84] G. R. Blakley, and C. Meadows, "Security of ramp schemes," *Proceedings of Crypto 84* , Springer-Verlag, New York (1984).

[Blu84] M. Blum, and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Computing* 13, 4 (Nov. 1984), pp. 850-864.

[Cha66] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *Journal of the ACM* 13 (1966), pp.. 547-569.

[Cha69] G. J. Chaitin, "On the length of programs for computing finite binary sequences: statistical considerations," *Journal of the ACM* 16 (1969), pp. 145-159.

[Cha75] G. J. Chaitin, "A theory of program size formally identical to information theory," *Journal of the ACM* 22 (1975), pp. 329-340.

[Eve84] S. Even, and O. Goldreich, "On the power of cascade ciphers," *ACM Transactions on Computer Systems* 3, 2 (1985), pp. 108-116.

[Kan84] R. Kannan, A. K. Lenstra, and L. Lovasz, "Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers," *ACM Symposium on the Theory of Computing* , 1984, pp. 191-200.

[Knu81] D. Knuth, *The Art of Computer Programming: Seminumerical Algorithms* , 2nd Edition, Addison-Wesley, 1981.

[Ko84] K. Ko, "A definition of infinite pseudo-random sequences," manuscript.

[Kol65] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Prob. of Inform. Transmission* 1,1 (1965), pp. 1-7.

[Kra84] E. Kranakis, "Theoretical aspects of the security of public key cryptography," *Technical Report 331* , Dept of Computer Science, Yale Univ., Sept. 1984.

[Mar66] P. Martin-Löf, "On the definition of random sequences," *Inform. and Control* 9 (1966), pp. 602-619.

[Mas85] J. L. Massey, and I. Ingemarson, "Toward a practical, computationally-secure cipher," presentation at Eurocrypt 85.

[Riv83] R. L. Rivest, and A. T. Sherman, "Randomized encryption techniques," *Advances in Cryptology: Proceedings of Crypto 82* , ed. by D. Chaum, et al., Plenum, 1983, pp. 145-163.

[Sha79] A. Shamir, "How to share a secret," *Communications of the ACM* 22, 11 (Nov. 1979), pp. 612-613.

[Sha83] A. Shamir, "On the generation of cryptographically strong pseudo-random sequences," *ACM Transactions on Computer Systems* 1,1 (Feb. 1983), pp. 38-44.

[Yao82] A. C. Yao, "Theory and applications of trapdoor functions," *Proc. 23rd IEEE Symp. on Found. of Computer Science* , 1982, pp. 80-91.