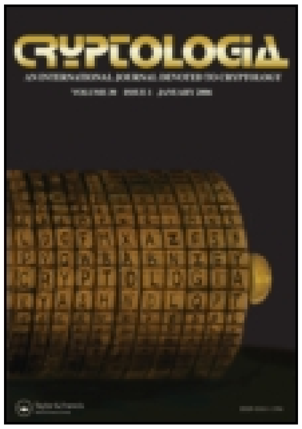


"FORWARDS AND BACKWARDS" ENCRYPTION

Ronald L. Rivest

Available online: 04 Jun 2010

"FORWARDS AND BACKWARDS" ENCRYPTION

Ronald L. Rivest

I present here a variation on the use of pseudo-random sequences for encryption which is extremely resistant to probable-word analysis. Decrypting any small fragment of the message does not directly allow the cryptanalyst to read the rest of the message.

The proposed method uses two pseudo-random sequences to encrypt each message. One sequence is generated left-to-right (forwards) in the normal manner, while the other is generated right-to-left (backwards); so that the first message symbol is encrypted using the first generated element of the forwards sequence and the last generated element of the backwards sequence. Each element of the ciphertext is the sum (modulo the alphabet size) of a message element and the corresponding elements of the two pseudo-random sequences.

We observe that this is not a stream cipher, since the entire message must be available before encryption can begin.

The pseudo-random sequences are chosen in such a way that the forwards sequence prevents a cryptanalyst from decrypting the portion of the message that preceded the location of a probable word attack, and the backwards sequence similarly prevents him from decrypting the text after the probable word. Together they make a probable-word attack futile.

We can model the generation of a pseudo-random sequence as follows. An initial state S_1 is chosen for the generator. This "seed" forms the cryptographic key for the sequence, since the entire sequence can be formed from the seed. A "next-state" function f is repeatedly used to form a sequence of states S_1, S_2, S_3, ... where $S_{i+1} = f(S_i)$. Finally, the pseudo-random sequence itself is typically not the state sequence but a transformed version of it: an "output function" g is used to transform the *i*th state S_i into the *i*th element of the pseudo-random sequence.

A probable word attack is often successful in discovering the state of the generator at the location of the attack. We wish to prevent this information from being useful to the cryptanalyst.

Consider the usual situation where only a single (forwards) generator is used. The cryptanalyst presumably knows the next-state and output functions f and g: the cryptographic key is the initial state S_1. Once the cryptanalyst discovers the *i*th state to be S_i with a probable word attack, he can easily decrypt the *i*th and all subsequent message elements since he can recreate the pseudo-random sequence from that point forwards. However, a well-chosen next-state function f can prevent the cryptanalyst from decrypting the previous $i-1$ message elements.

We will choose the next-state function f to be many-to-one to thwart the cryptanalyst in his efforts to use his knowledge of S_i to determine $S_1,\ldots,S_{i-1}$. If f is many-to-one for each state S_j there can be many states X such that $f(X) = S_j$; each of these states could have been S_{j-1}. As the cryptanalyst tries to use his determination of S_i to find S_{i-1}, S_{i-2}, etc., more ambiguity is introduced in each backwards step. If the rate at which ambiguity (uncertainty) is introduced exceeds the redundancy rate of the plaintext the cryptanalyst will be unable to decrypt the message prior to the *i*th element.

If we use a backwards pseudo-random sequence in addition to the forwards one,

This research was supported in part by NSF grant number MCS 78-05849.

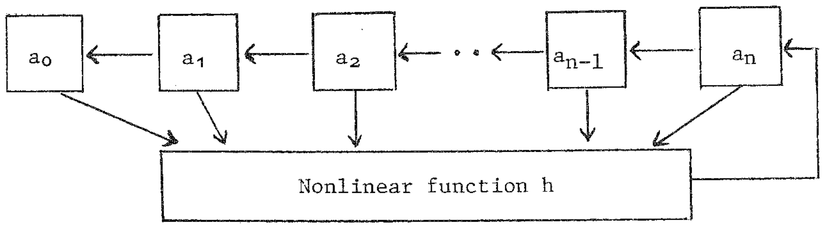
where both sequences are generated using many-to-one next-state functions, the cryptanalyst will be unable to use a probable word attack to determine *any* portion of the message, since the backwards sequence now prevents him from decrypting the message that follows the probable word.

This completes our description of the basic idea. In what follows, we present examples and suggestions on the choice of suitable many-to-one next-state functions.

There is no reason why the same next-state function f could not be used in the generation of both the forwards and the backwards sequences. The cryptographic key will typically consist of two seeds: one to initialize the generation of each sequence.

One simple choice of the next-state function is to use an f of the form $f(x) \equiv x^2 + a \pmod p$. Here p is a randomly chosen large prime number and a is another randomly chosen number $0 < a < p$. For each state S_i, where $i > 1$, of the pseudo-random state sequence $S_1, S_2, \ldots,$ there will generally be *two* solutions to the equation $f(X) = S_i$. If the output function g produces one bit of output for each state the ambiguity is introduced at a high rate.

Another technique is to use a nonlinear feedback shift-register of the form:



Each register cell holds one bit a_i of information; the next state of the register is $(a_1, a_2, \ldots, a_{n-1}, a_n, h(a_0, \ldots, a_n))$. As long as $h(a_0, \ldots, a_n)$ is not of the form $a_0 \oplus h'(a_1, a_2, \ldots, a_n)$ the next-state function will be many-to-one.

It is important to note that pseudo-random sequences generated using a many-to-one next-state function will not be "maximum period" sequences. If the number of generator states is n then a maximum period generator will produce a sequence of length n, whereas a pseudo-random sequence produced with a "random" next-state function will have period roughly $\sqrt{n}$. The number n of generator states should be chosen so that $\sqrt{n}$ is much larger than the length of the messages being enciphered.

As an example, consider encrypting the word CRYPTOGRAPHY using the next-state function $f(x) = x^2 + 3 \pmod{1009}$, the forwards seed 919, and the backwards seed 21, and the output function $g(x) = (x \bmod 26)$. The encryption computation is shown in Table 1, the numeric ciphertext 11, 12, 4, ..., is formed by adding (mod 26) the numeric message codes with the corresponding elements of the forwards and backwards pseudo-random sequences. Our ciphertext is thus LMEDXEIYLBMI.

Forwards State Sequence	Backwards State Sequence	Forwards Random Sequence	Backwards Random Sequence	Message	Numeric Message	Numeric Ciphertext	Ciphertext
919	130	9	0	C	2	11	L
31	146	5	16	R	17	12	M
964	446	2	4	Y	24	4	E
10	758	10	4	P	15	3	D
103	967	25	5	T	19	23	X
522	248	2	14	O	14	4	E
57	699	5	23	G	6	8	I
225	848	17	16	R	17	24	Y
178	145	22	15	A	0	11	L
408	384	18	20	P	15	1	B
991	444	3	2	H	7	12	M
327	21	15	21	Y	24	8	I

TABLE 1.

Even if a cryptanalyst could deduce that the sixth letter of our plaintext was O and that the states of the pseudo-random number generators for that letter were 522 and 248, he would not be able to determine that the next plain-text letter is G, since it equally well could have been F. (Note that $248 \equiv 699^2 + 3 \equiv 310^2 + 3 \pmod{1009}$, so that the next element of the backwards pseudo-random sequence could have been either 23 ($\equiv 699 \pmod{26}$) or 24 ($\equiv 310 \pmod{26}$).) Similarly the previous plain-text letter could be decrypted as either T (which is correct) or W, corresponding to forward states of 103 or 906. Thus a probable-word attack is of no avail to the cryptanalyst.

While the basic principle should now be clear, we leave it to the reader to investigate which many-to-one functions are most suitable for use in our method. The works mentioned in the bibliography describe other pseudo-random sequence generation techniques, or methods for breaking such techniques.

REFERENCES

1. Gait, J. 1977. A new nonlinear pseudorandom number generator, *IEEE Trans. on Software Engineering*. 3: 359-363.
2. Kahn, D. 1967. *The codebreakers*. New York: Macmillan.
3. Knuth, D. E. 1969. *The art of computer programming, Vol 2: seminumerical algorithms*. Reading: Addison-Wesley.
4. Meyer, C. H. 1974. Enciphering data for secure transmission. *Computer Design*. April: 129-134.
5. Meyer, C. H. and Tuchman, W. 1972. Pseudo-random codes can be cracked. *Electron. Des*. 23: 74-76.
6. Pless, V. 1977. Encryption schemes for computer confidentiality. *IEEE Trans. on Computers*. 26: 1133-1136.
7. Reeds, J. 1977. Cracking a random number generator. *Cryptologia*. 1: 20-26.
8. Reeds, J. 1979. Solution of challenge cipher. *Cryptologia*. 3: 83-95.