Fish: A Fast Software Stream Cipher

Uwe Blöcher and Markus Dichtl

Siemens AG, ZFE ST SN 3, D-81730 München, Germany,
E-Mail:bloecher@zfe.siemens.de or dichtl@zfe.siemens.de

Abstract. This paper describes a fast software stream cipher called *Fish* based on the shrinking principle applied to the lagged Fibonacci generator (*Fish* - Fibonacci shrinking). It is designed to make full use of the 32 bit word length of popular processors. On an Intel 486 clocked with 33 MHz a data rate of 15 Mbit/s is achieved with a C implementation.

1 Introduction

Coppersmith, Krawczyk, and Mansour ([CKM93]) presented at Crypto '93 a promising stream cipher, the shrinking generator. It is based on linear shift registers with linear feedback. The output bits of one shift register decide which of the output bits of the other shift registers are used and which are discarded. The design is well suited for hardware implementation. In software shift registers are not very efficient because each machine instruction operates on a single bit only. The remaining bits in the registers of the processor are unused.

In this paper we suggest an algorithm called *Fish*. We apply the shrinking principle to a stream cipher based on the lagged Fibonacci generator ([Knu81]) (*Fish* - Fibonacci shrinking). We use the full 32 bit wordlength of popular processors in order to achieve a high data rate.

2 The Principle of Shrinking Generators

In this section we describe a slight generalization of the principle of the generator suggested originally ([CKM93]). We consider two pseudo random generators A and S. A produces a sequence $a_0, a_1, \ldots$ of elements of $\mathrm{GF}(2)^{n_A}$. S produces a sequence $s_0, s_1, \ldots$ of elements of $\mathrm{GF}(2)^{n_S}$.

We apply a mapping $d : \mathrm{GF}(2)^{n_S} \to \mathrm{GF}(2)$ to the elements of $s_0, s_1, \ldots$ to decide which elements are accepted and which are discarded. In the original shrinking generator only elements genererated by A are accepted or discarded, in our generalization the results of S are treated the same. Another difference of our scheme is that the accepted elements are not yet the final result, another stage of processing is needed. We define the shrinking procedure as follows: If $d(s_i) = 1$ then a_i and s_i are accepted, otherwise they are discarded. That is, we define a sequence $i_1, i_2, \ldots, i_k, \ldots$ where i_k is the k-th position in $s_0, s_1, \ldots$ with $d(s_i) = 1$. We have $d(s_{i_k}) = 1$ and $\#\{j \in 0 \ldots i_k - 1 \mid d(s_j) = 1\} = k - 1$.

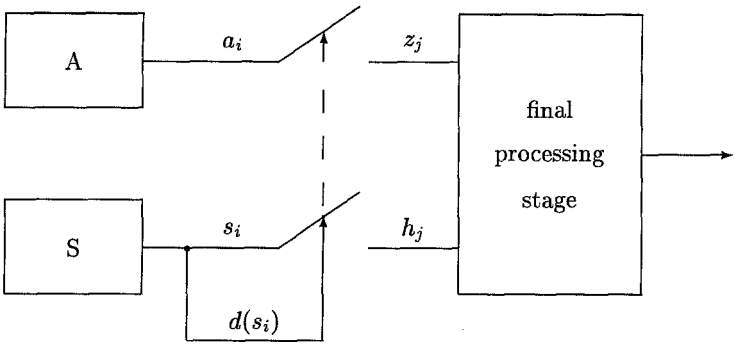

Fig. 1. Principle of the generalized shrinking generator.

We consider the shrunk sequences $z_0, z_1, \ldots$ which is $a_{i_1}, a_{i_2}, \ldots$ and $h_0, h_1, \ldots$ which is $s_{i_1}, s_{i_2}, \ldots$. For all elements h_j $d(h_j) = 1$ holds. The principle of the generalized shrinking generator is illustrated in Fig. 1.

In the original shrinking generator there was $n_A = 1$ and $n_S = 1$. The mapping $d(\,)$ was the identity. $z_0, z_1, \ldots$ were used as the output bits of the generator.

3 Specification of the Fast Software Algorithm Fish

In order to make full use of the 32 bit wordlength of most popular processors, we choose $n_A = 32$ and $n_S = 32$.

For both A and S we use the fastest software pseudo random number generator we know, namely the additive generator ([Knu81]) which is also called the lagged Fibonacci generator. We define

$$a_i = a_{i-55} + a_{i-24} \bmod 2^{32}$$

and

$$s_i = s_{i-52} + s_{i-19} \bmod 2^{32}$$

where $+$ stands for the arithmetical addition operation with carry, and the binary vectors are interpreted as unsigned numbers in the usual way. The values $a_{-55}, a_{-54}, \ldots, a_{-1}$ and $s_{-52}, s_{-51}, \ldots, s_{-1}$ are initial values of the generators and must be derived from the key. The sequence of the least significant bits of a lagged Fibonacci generator is generated by a linear feedback shift register (LFSR) where the feedback polynomial is a trinomial.

The mapping $d : \mathrm{GF}(2)^{32} \to \mathrm{GF}(2)$ maps a 32 bit vector to its least significant bit, $d((b_{31}, b_{30}, \ldots, b_0)) = b_0$.

It would be unsecure to use the shrunk sequence $z_0, z_1, \ldots$ as the result like in the original shrinking generator, since the underlying linear structure could

be detected. With probability 1/8 a triple of elements a_i, a_{i-55}, and a_{i-24} is accepted as elements of $z_0, z_1, \ldots$. An attacker could try to identify such triples by adding elements of $z_0, z_1, \ldots$ with a suitable distance and checking whether the sum turns up some elements later. Therefore we have to hide the linear structure of $z_0, z_1, \ldots$.

We split the sequences $z_0, z_1, \ldots$ and $h_0, h_1, \ldots$ up into pairs (z_{2i}, z_{2i+1}) and (h_{2i}, h_{2i+1}) and derive the two 32 bit output words r_{2i} and r_{2i+1} from these. We define

$$c_{2i} = z_{2i} \oplus (h_{2i} \wedge h_{2i+1})$$

$$d_{2i} = h_{2i+1} \wedge (c_{2i} \oplus z_{2i+1})$$

$$r_{2i} = c_{2i} \oplus d_{2i}$$

$$r_{2i+1} = z_{2i+1} \oplus d_{2i}$$

where $\oplus$ stands for the bitwise logical XOR operation and $\wedge$ for the bitwise logical AND. The last three equations achieve an exchange of those bits of c_{2i} and z_{2i+1} which are 1 in h_{2i+1}. The operations are visualized in Fig. 2.

The least significant bits of h_{2i} and h_{2i+1} are 1 because of our choice of the function d. Therefore it is possible to reconstruct the least significant bits of z_{2i} and z_{2i+1} from r_{2i} and r_{2i+1}, and vice versa the least significant bits of r_{2i} and r_{2i+1} follow from z_{2i} and z_{2i+1}. This implies that the least significant bits of the output words of *Fish* are the bits of the underlying LFSR shrinking generator which has a feedback trinomial.

4 Implementation Considerations

For the implementation a security aspect must be considered. It would be fatal for the security of the implementation if a potential attacker could find out from the time behaviour whether results of the additive generators were discarded or not. In applications where this could be possible it can be prevented by buffering.

5 Results for the Suggested Algorithm

On a PC with an Intel 486 clocked at 33MHz, using the Metaware High C compiler and the Pharlap DOS-Extender a data rate of 15Mbit/s for a C implementation of the suggested algorithm *Fish* is achieved.

Several statistical tests were applied to output sequences of the *Fish* algorithm: collision test, correlation test, coupon collectors test, frequency test, gap test, linear complexity test, Maurer test, overlapping m-tuple test, nonlinear complexity test, poker test, rank test, run test, spectral test, Ziv Lempel complexity test. None of those tests could detect a deviation from the behaviour of a random sequence.

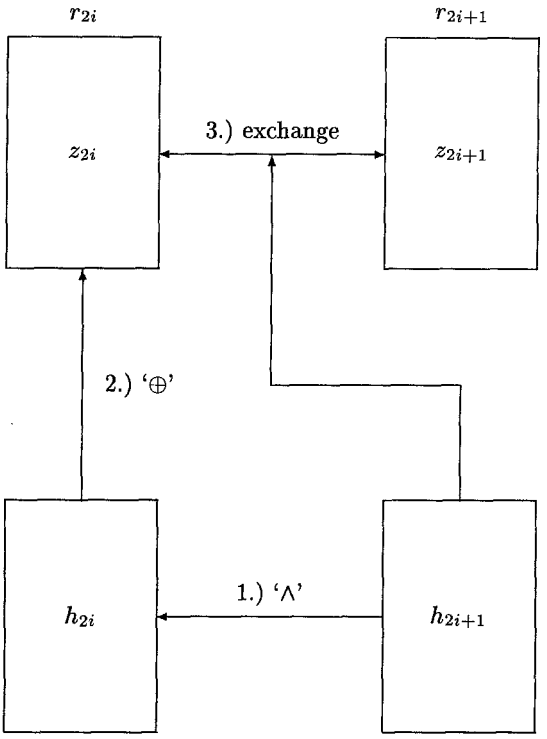

Fig. 2. Final processing stage: The output words r_{2i} and r_{2i+1} are derived by executing the indicated operations.

Acknowledgement

We thank Johan Mordhorst for speeding up the C-implementation of *Fish*.

References

[CKM93] D. Coppersmith, H. Krawczyk, Y. Mansour, 'The Shrinking Generator', Pre-Proceedings of CRYPTO '93.

[Knu81] D. E. Knuth, The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, 2nd Edition, Addison-Wesley, Reading, Mass., 1981.