Cryptanalysis of a modern rotor machine in a multicast setting

Shane Kepley, David Russo, and Rainer Steinwandt*

Florida Atlantic University
{skepley,drusso2012,rsteinwa}@fau.edu

Abstract. At FSE '93, Anderson presented a modern byte-oriented rotor machine that is suitable for fast software implementation. Building on a combination of chosen ciphertexts and chosen plaintexts, we show that in a setting with multiple recipients the recovery of an (equivalent) secret key can be feasible within minutes in a standard computer algebra system.

Keywords: cryptanalysis, rotor machine, stream cipher

1 Introduction

The most common mode of operation for stream ciphers is the generation of a pseudo-random sequence which is then added to a plaintext stream. Other modes of operation can be considered, however, and at FSE '93, Anderson proposed an elegant construction which combines a basic linear feedback shift register (LFSR) with three random permutations (rotors) on $\mathbb{Z}_{256}$ [1]. This results in an efficient byte-oriented (stream) cipher that is suitable for software implementations. To the best of our knowledge, no cryptanalytic weaknesses in this scheme have been published so far.

Anderson's design is minimalistic, and our attack shows that at least some form of strengthening is required to establish an acceptable security margin. In a standard computer algebra system, we can perform full recoveries of an (equivalent) secret key within minutes. Conceptually, it may be worth noting that our attack recovers the rotor *positions* before recovering the rotor "wiring".

2 Anderson's rotor cipher from FSE '93

At the core of the explored cipher we have an LFSR of length $n \geq 80$ over the binary field $\mathbb{F}_2$. The feedback polynomial $f(x)$ is assumed to be primitive and according to the specification *has at least 20 well-distributed nonzero coefficients.*

* RS was supported by AFRL/RIKF Award No. FA8750-15-2-0047.

Key space. While the feedback polynomial $f(x)$ is known, the LFSR's initial state $(\kappa_0^{(0)}, \ldots, \kappa_{n-1}^{(0)}) \in \mathbb{F}_2^n$ is part of the secret key. The remaining parts of the secret key are three uniformly and independently chosen permutations $\pi_1, \pi_2, \pi_3 \in S_{256}$, where S_{256} denotes the group of permutations on $\mathbb{Z}_{256}$. This results in a formidable key space of size $2^n \cdot (2^8!)^3 > 2^{5051+n}$. However, as we show in Section 3.2, the *effective* key space is significantly smaller than this.

Encryption. The encryption processes a stream of plaintexts one byte at a time. Shifting the LFSR once per clock cycle, we can encrypt a new byte every 8 clock cycles. To encrypt a new input byte $x \in \mathbb{Z}_{256}$ in clock cycle $8t$, we use the LFSR's internal state $(\kappa_0^{(8t)}, \ldots, \kappa_{n-1}^{(8t)}) \in \mathbb{F}_2^n$ at that time and read off the least significant three bytes

$$k_i^{(8t)} = \sum_{\nu=0}^{7} \kappa_{8i+\nu}^{(8t)} \cdot 2^\nu \in \mathbb{Z}_{256} \quad (i = 1, 2, 3).$$

Then we perform 8 shifts (one byte) of the LFSR to update the state, and output

$$y = \pi_3\left(k_3^{(8t)} + \pi_2\left(k_2^{(8t)} + \pi_1\left(k_1^{(8t)} + x\right) - k_1^{(8t)}\right) - k_2^{(8t)}\right) - k_3^{(8t)} \in \mathbb{Z}_{256}. \quad (1)$$

The intuition here is that π_1, π_2, π_3 represent connected rotors, and the k_i-values represent offsets of these rotors.

Decryption. The recipient knows the secret key and keeps its LFSR synchronized with the sender. To decrypt a received ciphertext byte y with the current state, one inverts each step of the encryption process by computing

$$x = \pi_1^{-1}\left(k_1^{(8t)} + \pi_2^{-1}\left(k_2^{(8t)} + \pi_3^{-1}\left(k_3^{(8t)} + y\right) - k_3^{(8t)}\right) - k_2^{(8t)}\right) - k_1^{(8t)} \in \mathbb{Z}_{256}.$$

3 Description of the attack

The goal of our attack is a *full secret key recovery.* As explained in Section 3.2, however, for each secret key there are several other secret keys which are equivalent, i. e., encryptions and decryptions are identical for all inputs. Consequently, we will restrict to recovering one secret key of this equivalence class. For our attack, we assume that the adversary controls the communication network as a middle-person and can perform the following actions:

- obtain encryptions of *chosen plaintext* bytes from the sender
- obtain decryptions of *chosen ciphertext* bytes from any recipient

Acting as middle-person, our adversary will replace some observed ciphertext bytes with ciphertexts of their own choice. Observing the received ciphertexts and the decryptions of the replaced bytes, linear equations for the initial state of the LFSR can be derived. We will try to use these equations to identify the initial state $(\kappa_0^{(0)}, \ldots, \kappa_{n-1}^{(0)})$ and then use this information to recover three secret permutations which are equivalent to the three actual secret permutations.

3.1 Recovering the initial state

By means of the companion matrix [3] of the underlying LFSR, we can express the state $(\kappa_0^{(8t)}, \ldots, \kappa_{n-1}^{(8t)})$, which determines the encryption of the $(t+1)^{\text{st}}$ plaintext byte, in terms of the initial secret state. More specifically, we have

$$(\kappa_0^{(8t)}, \ldots, \kappa_{n-1}^{(8t)}) = (\kappa_0^{(0)}, \ldots, \kappa_{n-1}^{(0)}) \cdot A^{8t}, \tag{2}$$

where A is the LFSR's companion matrix and the 8 in the exponent reflects that the register is clocked 8 times for each plaintext byte. For each of the initial state bits $\kappa_i^{(0)}$, we introduce an indeterminate K_i $(i = 0, \ldots, n-1)$, and we set up a linear system of equations over $\mathbb{F}_2$ in these n indeterminates.

There are only 2^{24} possible values for the least significant three bytes of the LFSR, so we can expect repetitions/collisions of these entries with some frequency. Suppose for the moment we are given an oracle that informs us that such a collision occurs at particular clock cycles, say

$$(\kappa_{n-1}^{(8t_0)}, \ldots, \kappa_{n-24}^{(8t_0)}) = (\kappa_{n-1}^{(8t_1)}, \ldots, \kappa_{n-24}^{(8t_1)}).$$

Then Equation (2) yields 24 linear equations over $\mathbb{F}_2$—we can equate the entries in the 24 least significant positions of $(K_0, \ldots, K_{n-1}) \cdot A^{8t_0}$ with those at the 24 least significant positions of $(K_0, \ldots, K_{n-1}) \cdot A^{8t_1}$. More generally, from a *multi-collision*

$$(\kappa_{n-1}^{(8t_0)}, \ldots, \kappa_{n-24}^{(8t_0)}) = (\kappa_{n-1}^{(8t_1)}, \ldots, \kappa_{n-24}^{(8t_1)}) = \cdots = (\kappa_{n-1}^{(8t_s)}, \ldots, \kappa_{n-24}^{(8t_s)})$$

we obtain $24 \cdot s$ linear equations over $\mathbb{F}_2$—we can equate the entries in the 24 least significant positions of $(K_0, \ldots, K_{n-1}) \cdot A^{8t_0}$ with those of $(K_0, \ldots, K_{n-1}) \cdot A^{8t_i}$ for each $i = 1, \ldots, s$. One may hope that for $24 \cdot s \geq n$ we can solve for the initial secret state.

Remark 1. This system of linear equations is homogeneous, but as we are over $\mathbb{F}_2$, a one-dimensional solution space yields a unique solution different from 0^n.

For $n = 80$ this means that we hope to see $s + 1 \geq 5$ repetitions of a value of the least significant LFSR bits. These 24 bits of the LFSR are not distributed independently and uniformly at random when encrypting plaintext bytes one by one. Still, to get an idea if an attack requiring 5 repetitions of a value of these 24 bits might be practical, applying Suzuki et al.'s analysis of the *birthday paradox for multi-collisions* [4] seems worthwhile. Based on their work, we can hope for a success probability 1/2 with about $\sqrt[5]{5!} \cdot (2^{24})^{4/5} \approx 2^{20.6}$ encrypted plaintext bytes. Section 3.3 shows that this estimate is reasonably in line with experimental reality.

Realizing the oracle. Suppose we encrypt a fixed plaintext byte x repeatedly, say at clock cycles $8t_i$ $(i = 0, 1, \ldots)$ and record the corresponding ciphertexts $y_0^{(8t_0)}, y_1^{(8t_1)}, \ldots$. A necessary condition for the 24 least significant bits of the

LFSR to repeat is that we observe a ciphertext that we have recorded earlier—this happens after no more than 2^8+1 encryptions of x. However, we must expect that there are multiple choices of the 24 least significant bits of the LFSR that explain a single plaintext-ciphertext pair$(x, y_i^{(8t_i)})$.

To eliminate false positives (same ciphertext without the 24 least significant bits of the LFSR being identical), we derive additional plaintext-ciphertext pairs through chosen ciphertexts. Namely, the adversary will for each clock cycle $8t_i$ transmit the ciphertext $y_i^{(8t_i)} + \ell \in \mathbb{Z}_{256}$ to the ℓ^{th} recipient and record the resulting decryption $\tilde{x}_{i,\ell}^{(8t_i)}$. In this way, we can collect ℓ (≤ 255) additional plaintext-ciphertext pairs $(\tilde{x}_{i,\ell}^{(8t_i)}, y_i^{(8t_i)} + \ell)$ for clock cycle $8t_i$ that are combined to increase confidence that the least significant 24 bits of the LFSR have repeated at that point in time. Specifically, we will guess that equality of these bits holds at clock cycle $8t_i$ and $8t_j$ if and only if all of the following hold:

- $y_i^{(8t_i)} = y_j^{(8t_j)}$,
- $\tilde{x}_{i,\ell}^{(8t_i)} = \tilde{x}_{j,\ell}^{(8t_j)}$ for all ℓ.

With a single recipient, we can only choose $\ell = 1$ and have a total of 2 plaintext-ciphertext pairs available, which still has room for false positives. In a multicast setting with L recipients we have $L+1$ plaintext-ciphertext pairs available, and for sufficiently large L we can meaningfully expect that they determine the least significant 24 bits of the LFSR uniquely—thereby avoiding false positives. In the experiment documented in the next section, we show that the attack works reliably and efficiently with $L = 4$.

3.2 Reconstructing equivalent rotors

For this section, we will assume that the attack described in Section 3.1 has succeeded, thus the adversary can calculate the LFSR state for any given point in time. Before trying to recover the remaining part of the secret key—the permutations π_1, π_2, and π_3—we observe that in the cipher under discussion different secret keys are equivalent. For $\Delta \in \mathbb{Z}_{256}$, denote by $\tau_\Delta \in S_{256}$ the cyclic shift

$$\begin{aligned} \tau_\Delta : \mathbb{Z}_{256} &\longrightarrow \mathbb{Z}_{256} \\ a &\longmapsto a + \Delta \end{aligned} .$$

Then we can rewrite Equation (1) as

$$y = \tau_{-k_3^{(8t)}} \circ \pi_3 \circ \tau_{k_3^{(8t)} - k_2^{(8t)}} \circ \pi_2 \circ \tau_{k_2^{(8t)} - k_1^{(8t)}} \circ \pi_1 \circ \tau_{k_1^{(8t)}}(x),$$

where $\circ$ denotes functional composition. As the cyclic shifts form an Abelian subgroup of S_{256}, we observe that Equation (1) remains valid for all plaintext-ciphertext pairs and rotor positions, if we replace (π_1, π_2, π_3) with

$$(\tau_\Delta \circ \pi_1, \tau_{-\Delta'} \circ \pi_2 \circ \tau_{-\Delta}, \pi_3 \circ \tau_{\Delta'})$$

for any choice of $(\Delta,\, \Delta') \in \mathbb{Z}_{256}^2$. This implies that the key space is redundant and its effective size is smaller than a naïve computation suggests.

For our purposes, we select $(\Delta, \Delta') = (-\pi_1(0), \pi_3^{-1}(0))$ to obtain permutations π_1', π_2', π_3' that behave exactly like π_1, π_2, π_3 and satisy $\pi_1'(0) = 0$ and $\pi_3'(0) = 0$. Our attack aims without loss of generality at recovering such a "normalized" triple of permutations, i. e., the inner and outer rotor stabilize 0.

Recovering π_2'. Having successfully recovered the initial state, the adversary will now always choose the plaintext bytes equal to $-k_1^{(8t)} \in \mathbb{Z}_{256}$, so that the result of applying π_1' within the encryption process is always known to be 0. Whenever a ciphertext y is observed that is equal to $-k_3^{(8t)} \in \mathbb{Z}_{256}$, then we conclude from our normalization condition on π_3' that the argument of π_3' must have been 0. In other words, from Equation (1) we obtain the equation

$$k_3^{(8t)} + \pi_2'\Big(k_2^{(8t)} + \underbrace{\pi_1'\left(k_1^{(8t)} - k_1^{(8t)}\right)}_{=0} - k_1^{(8t)}\Big) - k_2^{(8t)} = 0, \tag{3}$$

revealing that $\pi_2'(k_2^{(8t)} - k_1^{(8t)})$ is $k_2^{(8t)} - k_3^{(8t)} \in \mathbb{Z}_{256}$. The adversary keeps choosing plaintexts of the form $-k_1^{(8t)} \in \mathbb{Z}_{256}$ until the values $k_2^{(8t)} - k_1^{(8t)} \in \mathbb{Z}_{256}$ covered 255 different values in $\mathbb{Z}_{256}$ with an observed ciphertext $-k_3^{(8t)}$.[1] This yields the value of π_2' at all but one input. As π_2' is a permutation on $\mathbb{Z}_{256}$, this last missing value can then be filled in easily.

Recovering π_3'. Again, we restrict to plaintexts of the form $-k_1^{(8t)}$, so that π_1' evaluates to 0. Having π_2' available and knowing that π_1' evaluates to 0, we can evaluate the argument of π_3'—which is just the left-hand side of Equation (3). Because of Equation (1), adding $k_3^{(8t)}$ to the observed ciphertext yields the value of π_3' for this argument. Once the left-hand side of Equation (3) has taken 255 different values in $\mathbb{Z}_{256}$, we can complete the last missing value by exploiting that π_3' is a permutation.

Recovering π_1'. From the imposed normalization condition, we know already that $\pi_1(0) = 0$. Now, with π_2' and π_3' being known, from encryptions of $x - k_1^{(8t)} \in \mathbb{Z}_{256}$ for $x = 1, \ldots, 254$, we directly deduce the values $\pi'(1), \ldots, \pi'(254)$ from Equation (1). The only remaining value $\pi_1'(255)$ then follows from π_1' being a permutation.

3.3 Experimental validation

To test the practicality of the above attack, we implemented it in Magma [2] on a Dell PowerEdge T420. With `SetSeed(1234567890)`, we ran 20 experiments,

[1] It is actually sufficient to induce a chosen ciphertext only at those times when $k_2^{(8t)} - k_1^{(8t)}$ takes a value in $\mathbb{Z}_{256}$ at which the evaluation of π_2' is not known yet.

each time choosing the feedback polynomial $f(x)$ at random of degree $n = 80$ subject to the condition that it is primitive and the total number of terms is 21. In six runs the linear system of equations did not have a unique solution, and we counted these as failures, but even in these cases the solution space was only two-dimensional, i. e., only three non-zero candidates for the initial state were left. In the other 14 cases, the initial state was each time recovered successfully. From here, normalized rotors were each time found. None of these attacks used more than $2,150,000$ chosen plaintext bytes, and the majority of these were needed for the state recovery of the LFSR. Once this state was found, determining the rotor positions never required more than an additional $530,000$ chosen plaintext bytes. The total CPU time for finding a secret key was in each case less than 200 seconds.

4 Conclusion

The analysis above shows that the modern rotor machine in [1] is vulnerable to a practical attack. Our approach exploits that the underlying stepping mechanism for the rotors is a basic LFSR whose internal state can ultimately be recovered with linear algebra. To collect the necessary linear equations, our approach relies on the feasibility of recognizing multi-collisions of a certain portion of the internal state of the LFSR. To avoid false positives, we assume a multicast scenario, which allows us to collect multiple plaintext-ciphertext pairs.

At this point, we have not explored the security of Anderson's construction when using a more involved stepping mechanism, which defeats the attack as described here. So while our cryptanalysis shows that the basic form of Anderson's proposal from FSE'93 is vulnerable, tweaks of this elegant design still deserve further exploration.

References

1. Ross Anderson. A Modern Rotor Machine. In Ross Anderson, editor, *Fast Software Encrytion – FSE '93*, volume 809 of *Lecture Notes in Computer Science*, pages 47–50. Springer, 1994.
2. Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 24:235–265, 1997.
3. Todd Rowland. Companion Matrix. From *MathWorld*–A Wolfram Web Resource, created by Eric W. Weisstein.
4. Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyoya. Birthday Paradox for Multi-collisions. In Min Surp Rhee and Byoungcheon Lee, editors, *Information Security and Cryptology – ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 29–40. Springer, 2006.