Cryptanalysis of ORYX

D. Wagner[1], L. Simpson[2], E. Dawson[2], J. Kelsey[3], W. Millan[2], and B. Schneier[3]

[1] University of California, Berkeley
daw@cs.berkeley.edu
[2] Information Security Research Centre,
Queensland University of Technology
GPO Box 2434, Brisbane Q 4001, Australia
{simpson,dawson,millan}@fit.qut.edu.au
[3] Counterpane Systems,
101 E Minnehaha Parkway, Minneapolis, MN 55419
{schneier,kelsey}@counterpane.com

Abstract. We present an attack on the ORYX stream cipher that requires only 25–27 bytes of known plaintext and has time complexity of 2^{16}. This attack directly recovers the full 96 bit internal state of ORYX, regardless of the key schedule. We also extend these techniques to show how to break ORYX even under a ciphertext-only model. As the ORYX cipher is used to encrypt the data transmissions in the North American Cellular system, these results are further evidence that many of the encryption algorithms used in second generation mobile communications offer a low level of security.

1 Introduction

The demand for mobile communications systems has increased dramatically in the last few years. Since cellular communications are sent over a radio link, it is easy to eavesdrop on such systems without detection. To protect privacy and prevent fraud, cryptographic algorithms have been employed to provide a more secure mobile communications environment. First generation mobile communications devices were analog. Analog cellphones rarely use encryption, and in any case analog encryption devices offered a very low level of security [2]. Over the last five years digital mobile communications systems have emerged, such as the Global Systems Mobile (GSM) standard developed in Europe and several Telecommunications Industry Association (TIA) standards developed in North America [6]. For these digital systems, a much higher level of security, using modern encryption algorithms, is possible. Unfortunately, algorithms which offer a high level of security have not been used in mobile telecommunications to date.

In the case of GSM telephony, it is shown in [4] that it may be possible to conduct a known plaintext attack against the voice privacy algorithm used in GSM telephones, the A5 cipher. More recently it was shown in [3] that it is possible to clone GSM telephones by conducting a chosen-challenge attack on the COMP128 authentication algorithm.

The North American digital cellular standards designed by the TIA, including time division multiple access (TDMA) and code division multiple access (CDMA) both use roughly the same security architecture. The four cryptographic primitives used in these systems and described in the TIA standard [6] are:

- CAVE, for challenge-response authentication protocols and key generation.
- ORYX, a LFSR-based stream cipher for wireless data services.
- CMEA, a simple block cipher used to encrypt message data on the traffic channel.
- For voice privacy, TDMA systems use an XOR mask, or CDMA systems use keyed spread spectrum techniques combined with an LFSR mask.

The voice privacy algorithm in TDMA systems is especially weak since it is based on a repeated XOR mask. Such a system can be easily attacked using ciphertext alone [1]. The CMEA algorithm is susceptible to a known plaintext attack [7]. In this paper the security of the ORYX algorithm is examined.

ORYX is a simple stream cipher based on binary linear feedback shift registers (LFSRs) that has been proposed for use in North American digital cellular systems to protect cellular data transmissions [6]. The cipher ORYX is used as a keystream generator. The output of the generator is a random-looking sequence of bytes. Encryption is performed by XORing the keystream bytes with the data bytes to form ciphertext. Decryption is performed by XORing the keystream bytes with the ciphertext to recover the plaintext. Hence known plaintext-ciphertext pairs can be used to recover segments of the keystream. In this paper, the security of ORYX is examined with respect to a known plaintext attack conducted under the assumption that the cryptanalyst knows the complete structure of the cipher and the secret key is only the initial states of the component LFSRs.

For this attack, we assume that the complete structure of the cipher, including the LFSR feedback functions, is known to the cryptanalyst. The key is only the initial states of the three 32 bit LFSRs: a total keysize of 96 bits. There is a complicated key schedule which decreases the total keyspace to something easily searchable using brute-force techniques; this reduces the key size to 32 bits for export. However, ORYX is apparently intended to be a strong algorithm when used with a better key schedule that provides a full 96 bits of entropy. The attack proposed in this paper makes no use of the key schedule and is applicable to ORYX whichever key schedule is used.

2 The ORYX Cipher

The cipher ORYX has four components: three 32-bit LFSRs which we denote LFSR_A, LFSR_B and LFSR_K, and an S-box containing a known permutation L

of the integer values 0 to 255, inclusive. The feedback function for LFSR_K is

$$x^{32} + x^{28} + x^{19} + x^{18} + x^{16} + x^{14} + x^{11} + x^{10} + x^9 + x^6 + x^5 + x + 1.$$

The feedback functions for LFSR_A are

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

and

$$x^{32}+x^{27}+x^{26}+x^{25}+x^{24}+x^{23}+x^{22}+x^{17}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^7+x^2+x+1.$$

The feedback function for LFSR_B is

$$x^{32} + x^{31} + x^{21} + x^{20} + x^{16} + x^{15} + x^6 + x^3 + x + 1.$$

The permutation L is fixed for the duration of a call, and is formed from a known algorithm, initialised with a value which is transmitted in the clear during call setup. Each keystream byte is generated as follows:

1. LFSR_K is stepped once.
2. LFSR_A is stepped once, with one of two different feedback polynomials depending on the content of a stage of LFSR_K.
3. LFSR_B is stepped either once or twice, depending on the content of another stage in LFSR_K.
4. The high bytes of the current states of LFSR_K, LFSR_A, and LFSR_B are combined to form a keystream byte using the combining function:

$$Keystream = \{High8_K + L[High8_A] + L[High8_B]\} \bmod 256$$

3 Attack Procedure

Since ORYX has a 96-bit keyspace, it is not feasible to simply guess the whole generator initial state and check if the guess is correct. However, if the generator initial state can be divided into smaller parts, and it is possible to guess one small part of the generator initial state, and incrementally check whether that guess is correct, the generator can be attacked. The attack presented in this paper uses this divide and conquer approach, and is a refinement of a method originally proposed in [8]. A feature of ORYX which contributes to the efficiency of the attack outlined in this paper is that the two stages of LFSR_K whose contents control the selection of the feedback polynomial for LFSR_A and the number of times LFSR_B is stepped are both within the high eight stages of LFSR_K. Since the keystream bytes are formed from the contents of the high eight stages of each of the three LFSR states, we divide the keyspace and focus our attack on these 24 bits.

3.1 Attack Algorithm

Denote the high eight bits of the three LFSRs at the time the i^{th} byte of keystream is produced by $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$. The initial contents are $High8_A(0)$, $High8_B(0)$ and $High8_K(0)$, and all registers are stepped before the first byte of keystream, denoted $Z(1)$, is produced. To produce a keystream byte $Z(i+1)$ at time instant $i+1$, LFSR_K is stepped once, then LFSR_A is stepped once, then LFSR_B is stepped either once or twice. The contents of $High8_A(i+1)$, $High8_B(i+1)$ and $High8_K(i+1)$ are then combined to form the keystream byte $Z(i+1)$. Therefore, there is no need to guess all 24 bits: if we guess the contents of $High8_A(1)$ and $High8_B(1)$ we can use the first byte of the known keystream $Z(1)$ and the combining function to calculate the corresponding contents of $High8_K(1)$. Thus the attack requires exhaustive search of only a 16 bit subkey: the contents of $High8_A(1)$ and $High8_B(1)$.

For a particular 16-bit guess of $High8_A(1)$ and $High8_B(1)$, we use $Z(1)$ and calculate the corresponding contents of $High8_K(1)$. After this calculation, the attack proceeds iteratively as we construct a path of guesses of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ which are consistent with the known keystream. In each iteration a set of predictions for the next keystream byte is formed, and the guess evaluated by comparing the known keystream byte with the predicted values.

In the ith iteration, we exploit the fact that after stepping the three LFSRs to produce the next output byte, $High8_K(i+1)$ and $High8_A(i+1)$ effectively have one unknown bit shifting into them and, depending on $High8_K(i+1)$, $High8_B(i+1)$ has either one or two unknown bits, for each byte of output. We try all possible combinations of these new input bits, a total of 12 combinations, and compute the output byte for each case. At most, there will be 12 distinct output bytes which are consistent with the guess of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$. We compare the known keystream byte $Z(i+1)$ with the predicted output bytes.

If $Z(i+1)$ is the same as one of the predicted output bytes, for the case where there are 12 distinct outputs, then a single possible set of values exists for $High8_K(i+1)$, $High8_A(i+1)$ and $High8_B(i+1)$. We use these values in the next iteration of the attack.

Occasionally, where there are less than 12 distinct outputs, and the keystream byte is the same as the predicted output byte for more than one combination of new input bits, we must consider more than one possible set of values for $High8_K(i+1)$, $High8_A(i+1)$, and $High8_B(i+1)$. That is, the path of consistent guesses we are following may branch. In this situation we conduct a depth-first search.

If the keystream byte is not the same as any of the predicted output bytes, then the guessed contents of $High8_A(i)$ and $High8_B(i)$ were obviously incorrect. We go back along the path to the last branching point and start to trace out another path. If we search each possible path without finding a path of consistent guesses of length equal to the number of bytes of known keystream, then the guessed contents of $High8_A(1)$ and $High8_B(1)$ were obviously incorrect, and we make a new 16-bit guess and repeat the procedure.

When we find a sufficiently long path of consistent guesses, we assume that the values for $High8_A(1)$, $High8_B(1)$ and $High8_K(1)$ were correct. This provides knowledge of the contents of the high eight stages of each of the three LFSRs at the time that the first byte of keystream was produced. For the 24 consecutive guesses $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ for $2 \leq i \leq 25$, each set of values: $High8_K(i)$ and $High8_A(i)$ gives another bit in the state of LFSR_K and LFSR_A, respectively, and $High8_B(i)$ gives either one or two bits in the state of LFSR_B. Once we reconstruct the 32-bit state of each LFSR, at the time the first keystream byte was produced the LFSR states can then be stepped back to recover the initial states of the three LFSRs: the secret key of the ORYX generator. Thus we recover the entire key using a minimum of 25 bytes of keystream, and at most 2^{16} guesses. We use the recovered initial states to produce a candidate keystream and compare to the known keystream. If the candidate keystream is the same as the known keystream, the attack ends, otherwise we make a new 16 bit guess and repeat the procedure.

In practice, we may occasionally need a few more than 25 keystream bytes to resolve ambiguities in the final few bits of the LFSR states. That is, we may need to chase down a few more false trails to convince ourselves we got the last few bits right.

3.2 Testing Procedure

The performance of the attack was experimentally analysed to find the proportion of performed attacks for which the initial states can be successfully recovered, for various keystream lengths. The experiments use the following procedure: Nonzero initial states are generated for LFSR_A,LFSR_B and LFSR_K. A keystream segment of length N, $\{Z(i)\}_{i=1}^{N}$ is produced using the ORYX cipher as outlined in Section 2. The attack, as described in Section 3.1, is launched on the produced segment of the keystream. An outline of the testing procedure follows:

- *Input:* The length of the observed keystream sequence, N.
- *Initialisation:* $i = 1$, where i is the current attack index, Also define i_{max},the maximum number of attack trials to be conducted.
  LFSR_A initial state seed index, j is the current LFSR_B initial state seed index and k is the current LFSR_K initial state seed index.
- *Stopping Criterion:* The testing procedure stops when the number of attacks conducted reaches i_{max}.
- *Step 1:* Generate pseudorandom initial state seeds ASEED_i, BSEED_i and KSEED_i for LFSR_A, LFSR_B and LFSR_K, respectively. (pseudorandom number routine *drand*48, see [5] is used).
- *Step 2:* Generate pseudorandom LFSR initial states using ASEED_i, BSEED_i and KSEED_i for LFSR_A, LFSR_B and LFSR_K, respectively.
- *Step 3:* Generate the keystream sequence of bytes $\{Z(i)\}_{i=1}^{N}$.

- *Step 4:* Apply the attack to $\{Z(i)\}_{i=1}^{N}$ to obtain the reconstructions of the initial states of the three LFSRs.
- *Step 5:* If $i \leq i_{max}$, increment i and go to Step 1.
- *Step 6:* Stop the procedure.
- *Output:* Reconstructed initial states of LFSR_A, LFSR_B and LFSR_K.

4 Implementation Issues for the Attack

The attack procedure described in section 3.1 involves assuming that a particular guess of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ is correct, using this guess to form predictions for the next keystream byte, and then comparing a known keystream byte with the predictions: if the keystream byte contradicts all predictions, we conclude that the guess was wrong. However, it is possible that the keystream byte $Z(i+1)$ will be the same as one of the predicted output bytes, although the values for $\text{High8}_A(i)$ and $\text{High8}_B(i)$ are incorrect. We refer to such a situation as a false alarm.

4.1 The Probability of a False Alarm

For the attack to be effective, the probability of a false alarm occurring must be small. Therefore, we require a high probability that an incorrect guess will be detected (through comparison of the predicted outputs with the corresponding keystream byte). That is, we require the probability that no predicted output byte matches the actual keystream byte to be significantly greater than one half, given that the guessed contents of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ are incorrect.

Consider the formation of the predicted output values. Each predicted output is formed from an 8-bit possible value for $High8_A(i+1)$, an 8-bit possible value for $High8_B(i+1)$ and an 8-bit possible value for $High8_K(i+1)$. So there are a total of 2^{24} different input combinations. The predicted output has 8 bits. Therefore, given a particular output value, there exist multiple input combinations which result in this output. As the inputs are all non-negative integers less than 256 and the combining function is the modulo 256 sum of the three inputs, all output values are equally likely if the input combinations are equally likely. Thus each output value is produced by 2^{16} different input combinations. If one of these input combinations is the correct combination, then there are $2^{16}-1$ other combinations which, although incorrect, produce the same value. The probability that a single incorrect input combination produces the same output as the correct combination is $\frac{2^{16}-1}{2^{24}-1} \approx 0.0039$. The probability that a single incorrect input combination produces a value different to the output of the correct combination is the complement of this; approximately 0.9961. We select a set of twelve input combinations. The probability that an incorrect guess will be detected (through comparison of the predicted outputs with the corresponding keystream byte) is the probability that none of the predicted outputs is the same as the known keystream value, given that all of the twelve possible input combinations are incorrect. The situation can be approximated by the binomial distribution. Therefore,

$$P(\text{incorrect guess detected}) \approx (.9961)^{12} = 0.9541$$

Since the probability that no predicted output byte matches the actual keystream byte, given that the guessed contents of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ are incorrect, is 0.9541, the probability that at least one predicted output byte matches the actual keystream byte, given that the guessed contents of $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ are incorrect, is 0.0459. The probability of a false alarm is less than five percent, and the nice attribute of a false alarm is that once we are on the wrong track, we have a 0.9541 probability of detecting this at each step.

Using the binomial distribution to calculate approximate probabilities, given the guessed bits are incorrect,

$$P(\text{keystream byte matches 1 prediction}) \approx \binom{12}{1}(0.0039)^1(0.9961)^{11} = 0.0448$$

$$P(\text{keystream byte matches 2 predictions}) \approx \binom{12}{2}(0.0039)^2(0.9961)^{10} = 0.0010$$

$$P(\text{keystream byte matches } \geq 2 \text{ predictions}) \approx 0.0001$$

From this, we conclude that most of the time, we will generate very few false trails—typically just one or two. Thus, we perform a depth-first search of the possible states, but we seldom spend much time on a false trail.

Note that if we have the correct states for $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ we never mistakenly think we have the wrong state. Once we identify the correct $\mathrm{High8}_A(1)$, $\mathrm{High8}_B(1)$ and $\mathrm{High8}_K(1)$, we can quickly find the correct states for $\mathrm{High8}_A(i)$, $\mathrm{High8}_B(i)$ and $\mathrm{High8}_K(i)$, for $2 \leq i \leq n$ for some $n \geq 25$. From these we can reconstruct the initial states of the three LFSRs.

4.2 Effect of Length of Known Keystream

The minimum length of keystream required for this attack to be successful is 25 bytes; one byte to obtain the required eight bit value for $High8_K(1)$, giving a known eight bits in each of the three 32-bit LFSR initial states, and then one byte to recover each of the other 24 bits in the three LFSR initial states. The more keystream available, the more certain we are of successful reconstruction. However, if we have less than 25 bytes of known keystream, the attack can still be performed as outlined above to give a likely reconstruction of most of the LFSR states, and we use exhaustive search over the contents of the last few stages.

N	25	26	27
% Success	99.7	99.9	100.0

Table 1. Success rate (%) versus N.

5 Experimental Results

The performance of the attack was experimentally analysed to find the the proportion of performed attacks for which the initial states can be successfully recovered, for the keystream lengths $N = 25$, 26, and 27. For each keystream length, the attack was performed one thousand times, using pseudorandomly generated LFSR initial states. The attack was considered successful if the reconstructed LFSR initial states were the same as the actual LFSR initial states. Table 1 shows as the success rate the proportion of attacks conducted which were successful, for each value of N.

From Table 1, we observe that even for the minimum keystream length, $N = 25$, the attack is usually successful. In a small number of cases, there exist multiple sets of LFSR initial states which produce the required keystream segment and the attack cannot identify the actual states used. However, as noted in Section 3.1 only a small increase in keystream length is required to eliminate these additional candidates.

6 Ciphertext-only Attacks

In many cases, the known-plaintext attack on ORYX can be extended to a ciphertext-only attack if some knowledge of the plaintext statistics is assumed. For example, when the plaintext is English text or other stereotyped data, ciphertext-only attacks are likely to be feasible with just hundreds or thousands of bytes of ciphertext.

To perform a ciphertext-only attack we start by identifying a probable string of at least seven characters; a word or phrase which is likely to appear in the plaintext. Examples of probable strings include "`login:␣`" and "`.␣␣The␣`". We then slide the probable string along each ciphertext position, hoping to find a "match" with the correct cleartext message.

If we align a probable plaintext string correctly, then we obtain a segment of the keystream with length equal to the length of the probable plaintext string. The known-plaintext attack described above can be performed on this keystream segment. If every path of guesses is ruled out by the end of the $N = 7$ bytes of known text, then we know the probable string does not match the cleartext at this position. Otherwise, we conclude that we have found a valid match; this may sound optimistic, but we show next that the probability of error is acceptably low.

With this procedure, false matches should be rare, because false paths of guesses are eliminated very quickly. After analyzing the first byte, 2^{16} possibilities for the high bytes of each register remain. From Section 4.1, only 0.0459 of the wrong possibilities remain undetected after the second byte; of those, the proportion which remain undetected after the third byte is 0.0459; and so on. This means that only $2^{16} \cdot (0.0459)^6 = 0.00061 \approx 2^{-10.7}$ wrong possibilities are expected to survive the tests after $N = 7$ bytes of known text are considered, on average. The probability of a false match being accepted is at most $0.00061 \approx 2^{-10.7}$. Therefore, with less than a thousand bytes of ciphertext, we expect to see no false matches, for probable strings of length $N = 7$. Using a slightly longer probable word will further reduce the probability of error.

The search for ciphertext locations which yield a valid match with the probable word can be performed quite efficiently. It should take about 2^{16} work, on average, to check each ciphertext position for a possible match. With less than a thousand bytes of ciphertext, the total computational effort to test a probable word is less than 2^{26}, and thus even a search with a dictionary of probable words is easily within reach.

Next, we describe how to use matches with the probable word to recover the ORYX key material. Each valid match provides $8 + (N - 1) = 14$ bits of information on the initial states of LFSR_A and LFSR_K, and $8+1.5\cdot(N-1) = 17$ bits of information on the initial state of LFSR_B. Therefore, with three probable word matches, we will have accumulated about 42 bits of information on each of the 32 bit keys for LFSR_A and LFSR_K, and 51 bits of information on the 32 bit key for LFSR_B. The key can be easily recovered by solving the respective linear equations over $GF(2)$. Alternatively, with two matches, we have 28 bits of information for LFSR_A and LFSR_K, and 34 bits of information for LFSR_B. An exhaustive search over the remaining eight unknown bits should suffice to find the entire key with about 2^8 trials. For each key trial, we can decrypt the ciphertext, and check whether the result looks like plausible plaintext by using simple frequency statistics or more sophisticated techniques.

As long as we have enough ciphertext and can identify some set of probable words, it should be easy to find two or three matches and thus recover the entire ORYX key. In other words, it appears that even ciphertext-only attacks against ORYX have relatively low complexity, when some knowledge of the plaintext statistics is available. The computational workload and the amount of ciphertext required are modest, and these attacks are likely to be quite practical.

7 Summary and Conclusions

ORYX is a simple stream cipher proposed for use as a keystream generator to protect cellular data transmissions. The known plaintext attack on ORYX presented in this paper is conducted under the assumption that the cryptanalyst knows the complete structure of the cipher and the 96-bit secret key is only the initial states of the component LFSRs. The attack requires exhaustive search over 16 bits, and has over 99 percent probability of success if the cryptanalyst knows 25 bytes of the keystream. The probability of success is increased if the cryptanalyst has access to more than 25 bytes of the keystream. In our trials, a keystream length of 27 bytes was sufficient for the attack to correctly recover the key in every trial. Furthermore, we have shown how to extend this to a ciphertext-only attack which is likely to be successful with only hundreds or thousands of bytes of known ciphertext.

These results indicate that the ORYX algorithm offers a very low level of security. The results further illustrate the low level of security offered in most second generation mobile telephone devices. The authors are of the opinion that, in most cases, this is due to the lack of public scrutiny of the cryptographic algorithms prior to their adoption for widespread use. It is to be hoped that the past reliance on security through obscurity will not be repeated in the cryptographic algorithms to be used in the third generation of mobile communications systems, due for use early in the twenty-first century.

References

1. E. Dawson and L. Nielsen. Automated cryptanalysis of XOR plaintext strings. *Cryptologia*, volume XX Number 2, pages 165–181. April 1996.
2. B. Goldburg, E. Dawson and S. Sridharan. The automated cryptanalysis of analog speech scramblers. *Advances in Cryptology - EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 422–430. Springer-Verlag, 1991.
3. M. Briceno, I. Goldberg and D. Wagner. GSM cloning. 20 April, 1998. `http://www.isaac.cs.berkeley.edu/isaac/gsm.htm`
4. J. Dj. Golić. Cryptanalysis of alleged A5 stream cipher. *Advances in Cryptology - EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.
5. H. Schildt. *C the Complete Reference* Osborne McGraw-Hill, Berkeley, CA, 1990.
6. TIA TR45.0.A, *Common Cryptographic Algorithms* June 1995, Rev B.
7. D. Wagner, B. Schneier and J. Kelsey. Cryptanalysis of the cellular message encryption algorithm. *Advances in Cryptology - CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 526–537. Springer-Verlag, 1997.
8. D. Wagner, B. Schneier and J.Kelsey. *Cryptanalysis of ORYX.* unpublished manuscript, 4 May 1997.