Practical Attacks on one Version of DICING

Gilles Piret

Ecole Normale Supérieure, Département d'Informatique,
45, Rue d'Ulm, 75230 Paris cedex 05, France
`http://www.di.ens.fr/~piret/`
`Gilles.Piret@ens.fr`

Abstract

DICING is a synchronous stream cipher submitted to the ECRYPT stream cipher project. Two versions of the cipher actually exist: the first one can be found in the proceedings of the SKEW conference, while the second is available from the web site. In this paper we describe practical distinguishing and key recovery attacks against the first version. These attacks do not apply as such to the web site version of DICING.

1 Introduction

The ECRYPT stream cipher project [1] aims at identifying new stream ciphers that might become suitable for widespread adoption. For this purpose, a public call for primitives has been made in November 2004. In May 2005, it resulted in 34 stream cipher submissions.

DICING is one of them. It is based on four Galois-style LFSRs, two of which are used to clock the other two. While such irregular clocking is a good means of obtaining non-linearity at a low cost, the security of primitives based on this principle is often difficult to analyze.

It happens to be two versions of DICING. The first one [2] can be found in the proceedings of the SKEW conference, that took place in Åarhus, Denmark, on May 26-27, 2005. The second [3] is available from the ECRYPT web site; it differs from [2] by several changes to the output function. In this paper, we are concerned with the security of the first version. We show that the way variable clocking is applied in it leads to very serious weaknesses.

2 Notations

Throughout this paper, we use the following notations:

- $\mathbb{F}_{2^n}$ is the Galois field with 2^n elements.
- $\oplus$ denotes exclusive or, that is bitwise addition.

- & denotes bitwise AND.
- $\sim X$ denotes the bit by bit complement of X.
- $X \gg a$ denotes the right shift of X by a bits.
- $X[a, b]$ denotes the substring of binary string X, going from bit position a to bit position b.

3 Description of DICING

3.1 State Update Function

The DICING stream cipher is based on four Galois-style LFSRs Γ_1, Γ_2, Γ_3, Γ_4. Let $\alpha_t \in \mathbb{F}_{2^{127}}, \beta_t \in \mathbb{F}_{2^{126}}, \omega_t \in \mathbb{F}_{2^{128}}, \tau_t \in \mathbb{F}_{2^{128}}$ denote the state of LFSR $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ respectively. We can represent the elements of a Galois field of characteristic 2 as polynomials in $\mathbb{F}_2[x]/p(x)$, where $p(x)$ is an irreducible polynomial over $\mathbb{F}_2$. As an example, α_t will be denoted as $\alpha_{t,126} \cdot x^{126} \oplus \alpha_{t,125} \cdot x^{125} \oplus ... \oplus \alpha_{t,0}$. If the polynomial taken corresponds to the feedback polynomial of the LFSR, then shifting the LFSR is equivalent to multiplication by x in $\mathbb{F}_2[x]/p(x)$. We do not give the feedback polynomials $p_i(x)$ of LFSRs $\Gamma_i (i = 1...4)$ here as they are not relevant for our attacks. Moreover, we often omit the modulo in our equations, as it is obvious from the context.

LFSRs Γ_1 and Γ_2 are shifted 8 bits per clock cycle, and are used to clock the other two LFSRs. More precisely, the state update process is the following:

1. The last eight bits of α_t and β_t are stored in *dices* D'_t and D''_t:

$$\begin{aligned} D'_t &= (\alpha_{t,126}, ..., \alpha_{t,119}) \in \mathbb{F}_2^8 \\ D''_t &= (\beta_{t,125}, ..., \beta_{t,118}) \in \mathbb{F}_2^8 \end{aligned} \tag{1}$$

   Then Γ_1 and Γ_2 are updated:

$$\begin{aligned} \alpha_{t+1} &= x^8 \cdot \alpha_t \mod p_1(x) \\ \beta_{t+1} &= x^8 \cdot \beta_t \mod p_2(x) \end{aligned} \tag{2}$$

2. $D_t = D'_t \oplus D''_t, \quad a_t = D_t \& 15 \in \mathbb{F}_2^4, \quad b_t = D_t \gg 4 \in \mathbb{F}_2^4$
3. Two memories $u_t, v_t \in \mathbb{F}_{2^{128}}$ are updated by XORing the states ω_t and τ_t to them:

$$\begin{aligned} u_t &= u_{t-1} \oplus \omega_t \\ v_t &= v_{t-1} \oplus \tau_t \end{aligned} \tag{3}$$

4. Γ_3 and Γ_4 are updated by shifting them from 0 to 15 bits depending on the value of a_t and b_t:

$$\begin{aligned} \omega_{t+1} &= x^{a_t} \cdot \omega_t \mod p_3(x) \\ \tau_{t+1} &= x^{b_t} \cdot \tau_t \mod p_4(x) \end{aligned} \tag{4}$$

3.2 Output Function

At each clock cycle, the output function produces a 128-bit value z_t, depending on values u_t, v_t, D'_t, D''_t. The function used depends on how D'_t and D''_t compares:

$$z_t = \begin{cases} C_0(u_t) \oplus v_t & \text{if } D'_t > D''_t \\ C_0(v_t) \oplus u_t & \text{if } D'_t < D''_t \\ u_t \oplus v_t & \text{if } D'_t = D''_t \end{cases} \tag{5}$$

where C_0 is a non-linear and key-dependent function.

3.3 Initialization

The initialization of the generator is done in four phases:

1. The key and IV material are used to compute the initial states $\alpha_{-64}, \beta_{-64}, \omega_{-64}, \tau_{-64}$ of the four LFSRs.
2. The state update function is applied to them 32 times without output.
3. The resulting state is used to construct the function C_0 used in the output function (see [2] for more details).
4. The state update function is applied another 32 times. We obtain $\alpha_0, \beta_0, \omega_0, \tau_0$, the initial states of $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ before keystream generation.

The key and IV loading proceeds as follows:

1. $K_I = K \oplus IV$
2. $K' = \begin{cases} K_I & \text{if } K \text{ has length 256} \\ K_I | (\sim K_I) & \text{if } K \text{ has length 128} \end{cases}$
3. $K_{ICS} = S_0(K' \oplus c)$, where S_0 denotes the parallel application of a fixed S-box $S_0 : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ and c is a constant.
4. $\alpha_{-64} = K_{ICS}[0, 126] \qquad \beta_{-64} = K_{ICS}[128, 253]$
5. $s = \bigoplus_{0 \le i < 32} K_{ICS}[i]_{\text{byte}} \in \mathbb{F}_2^8 \qquad \sigma = (s, s, ..., s) \in \mathbb{F}_2^{256}$
6. $K_{II} = S_0(K_{ICS} \oplus \sigma \oplus (\sim c))$
7. $\omega_{-64} = K_{II}[0, 127] \qquad \tau_{-64} = K_{II}[128, 255]$

It is remarkable that knowledge of ω_{-64} and τ_{-64} is enough to retrieve K_I.

4 A Practical Distinguisher

Assume that during cycle t both dices D'_t and D''_t have the same value. Due to statistical properties of the LFSRs this event happens exactly with probability 1/256. Then $(a_t, b_t) = D_t = D'_t \oplus D''_t = 0$. Therefore states ω_t and τ_t do not change during this cycle: $\omega_{t+1} = \omega_t$ and $\tau_{t+1} = \tau_t$. It implies $u_{t+1} = u_t \oplus \omega_{t+1} = u_{t-1} \oplus \omega_t \oplus \omega_{t+1} = u_{t-1}$ and similarly

$v_{t+1} = v_{t-1}$. Finally if the output function used is the same for cycles $t-1$ and $t+1$, we have $z_{t+1} = z_{t-1}$. It will happen whenever ($D'_{t-1} < D''_{t-1}$ and $D'_{t+1} < D''_{t+1}$), or ($D'_{t-1} = D''_{t-1}$ and $D'_{t+1} = D''_{t+1}$), or ($D'_{t-1} > D''_{t-1}$ and $D'_{t+1} > D''_{t+1}$), thus with probability $2 \cdot \left(\frac{2^8-1}{2^9}\right)^2 + \frac{1}{2^{16}} \simeq \frac{1}{2}$.
The conclusion is that two 128-bit output words produced at cycles $t-1$ and $t+1$ are equal with probability $\simeq \frac{1}{512}$ (instead of 2^{-128} for a truly random sequence). So the amount of keystream necessary for our distinguisher to work is about $512 \cdot 128$ bits $= 64$ Ko. The processing time is negligible.

5 A Key Recovery Attack

Instead of assuming $D'_t = D''_t$, suppose that D'_t and D''_t agree on their 4 right-most (or left-most) bits only. Then (assuming the first case, the second is similar)

$$\begin{cases} \omega_{t+1} = \omega_t \\ \tau_{t+1} = x^{b_t} \cdot \tau_t \neq \tau_t \end{cases}, \tag{6}$$

which implies

$$\begin{cases} u_{t+1} = u_{t-1} \\ v_{t+1} \neq v_{t-1} \end{cases}. \tag{7}$$

If $D'_{t-1} > D''_{t-1}$ (resp. $D'_{t+1} > D''_{t+1}$) then $z_{t-1} = C_0(u_{t-1}) \oplus v_{t-1}$ (resp. $z_{t+1} = C_0(u_{t+1}) \oplus v_{t+1}$). As both events occur with probability $\simeq 1/2$, and $a_t = 0$ with probability $1/16$, we conclude that

$$z_{t-1} \oplus z_{t+1} = v_{t-1} \oplus v_{t+1} = \tau_t \oplus \tau_{t+1} = \tau_t \cdot (1 \oplus x^{b_t}) \tag{8}$$

is satisfied with probability $\simeq 1/64$. Similarly (considering the case $b_t = 0$ instead of $a_t = 0$),

$$z_{t-1} \oplus z_{t+1} = u_{t-1} \oplus u_{t+1} = \omega_t \oplus \omega_{t+1} = \omega_t \cdot (1 \oplus x^{a_t}) \tag{9}$$

is satisfied with probability $\simeq 1/64$ as well.

Assume the attacker has got a long enough keystream sequence $(z_t)_{t \geq 0}$. The idea of the attack is the following: each time $a_t = 0$ (resp. $b_t = 0$) and the conditions on $D'_{t-1}, D''_{t-1}, D'_{t+1}, D''_{t+1}$ are satisfied, by guessing correctly the value of b_t (resp. a_t), we can obtain the actual value of τ_t (resp. ω_t) by using equation (8) (resp. equation (9)). From this value we can compute the sequence of the past states of the LFSR (with two consecutive elements of the sequence differing by one bit shift of the LFSR). As equations (8) and (9) are satisfied relatively often, by considering enough positions t we will observe several similar sequences (provided we "align" them correctly). On the other hand, when equation (8) (resp. (9)) is not satisfied, or we do not guess correctly, the value τ_t (resp. ω_t) we deduce can be considered as random (see Appendix A). So the observed similar sequences highly probably correspond to the right one. The best way to identify them is to use two hash tables (or sorted lists) Ω and $\mathbf{T}$.

Once the actual past history of Γ_3 and Γ_4 has been identified, it remains to identify the actual initial states ω_{-64} and τ_{-64} of these registers. Knowing the state of one LFSR at cycle t, the number of bit shifts separating it from the initial state could roughly go from 0 to $15 \cdot (t + 64)$ depending on how the LFSR has been clocked, with an expectancy of $7.5 \cdot (t + 64)$. So we guess this distance for both LFSRs, beginning with values close to the expectancy (which are the most probable ones, as the distance is a random variable with roughly a normal distribution). The computed initial states allow us to retrieve the key from which we can compute a keystream. Comparison with the actual keystream is used to accept or reject the guess on the distances.

More precisely, the attack goes as follows:

1. $\Omega, \mathbf{T} = \varnothing$. For $t = 1, 2, ...$:
   (a) Compute $z_{t-1} \oplus z_{t+1}$.
   (b) Assume $a_t = 0$. For $b_t = 1, 2, ..., 15$:
   Deduce the supposed value $\tau_t^{(b_t)}$ using (8). Use it to compute the history of Γ_4: more precisely, we compute a sequence of values $(\tau_{t,s}^{(b_t)})_{-15t+15 \leq s \leq 0}$, such that $\tau_{t,0}^{(b_t)} = \tau_t^{(b_t)}$ and $\tau_{t,s}^{(b_t)} = x \cdot \tau_{t,s-1}^{(b_t)}$. The bound on s is chosen such that, assuming the guesses on a_t and b_t were right, all values in the actual sequence $(\tau_t)_{t \geq 1}$ are also in $(\tau_{t,s}^{(b_t)})_{-15t+15 \leq s \leq 0}$. The difference between both sequences is that the latter is regularly clocked (shifts of one bit at a time), while the former results from variable clocking. For $-15t + 15 \leq s \leq 0$, check whether $\tau_{t,s}^{(b_t)}$ is in the hash table $\mathbf{T}$.
      - If yes, let $(\tau_{t^*}^{(b_{t^*})}, t^*) \in \mathbf{T}$ be this element. It is probably part of the true history of Γ_4. From $(\tau_{t^*}^{(b_{t^*})}, t^*)$ reconstruct the history $(\tau_{t^*,s}^{(b_{t^*})})_{-15 \cdot (t^*+64) \leq s \leq 0}$.
      - Else store $(\tau_t^{(b_t)}, t)$ in $\mathbf{T}$.
   (c) Similarly, we can assume $b_t = 0$ and compute a supposed value $\omega_t^{(a_t)}$ for each $a_t \in \{1, 2, ..., 15\}$ using (9). We deduce candidate sequences $(\omega_{t,s}^{(a_t)})_{-15t+15 \leq s \leq 0}$, and use another hash table Ω to identify similar sequences. The only difference with step (b) is that computations are performed modulo $p_3(x)$, instead of modulo $p_4(x)$.
   (d) Stop the loop as soon as the true history of both Γ_3 and Γ_4 has been found.
2. Let $(\tau_s^*)_s$ and $(\omega_s^*)_s$ be the two sequences we computed at step 1, and t', t'' be the respective corresponding indexes of the clock cycle corresponding to $s = 0$. Let $\overline{s}' := \lfloor -7.5 \cdot (t' + 64) \rfloor$ and $\overline{s}'' := \lfloor -7.5 \cdot (t'' + 64) \rfloor$. Let us denote by `Try`$(s_1, s_2)$ the following computation:
   - Assume that s_1 and s_2 are the indexes of the initial state of Γ_4 and Γ_3 in $(\tau_s^*)_s$ and $(\omega_s^*)_s$ respectively.

- Deduce the initial key.
- Check it by generating keystream from it and comparing it to the actual keystream.

First we perform $\texttt{Try}(\overline{s}', \overline{s}'')$. Then we set $i = 1$ and repeat:

(a) For $j = \overline{s}'' - i$ to $j = \overline{s}'' + i$, $\texttt{Try}(\overline{s}' - i, j)$ and $\texttt{Try}(\overline{s}' + i, j)$.

(b) For $j = \overline{s}' - i + 1$ to $j = \overline{s}' + i - 1$,
$\texttt{Try}(j, \overline{s}'' - i)$ and $\texttt{Try}(j, \overline{s}'' + i)$.

(c) i++

We stop as soon as Try gives a positive answer.

Remark that step 1 is another distinguisher for DICING: as a matter of fact, performing this computation on a truly random sequence is very unlikely to lead to discovery of a collision in Ω or $\mathbf{T}$. As we will see, this distinguisher requires less data than the previous one, but more computation.

We now look at the complexity of the attack. For it to succeed, we need equations (8) and (9) to be satisfied in two distinct positions t. As these equations are satisfied with probability 1/64, a keystream of about 128 words= 16 Ko is necessary.

Regarding the time complexity of the first phase of the attack (and hence of the distinguisher), about $128 \cdot 15$ sequences $(\tau_{t,s}^{(b_t)})_{-15 \cdot t \leq s \leq 0}$ and $128 \cdot 15$ sequences $(\omega_{t,s}^{(a_t)})_{-15 \cdot t \leq s \leq 0}$ need to be computed. Their average length is $15 \cdot 64$, so the total time complexity of this phase is about $2^7 \cdot 15 \cdot 2 \cdot 15 \cdot 64 \simeq 2^{22}$ LFSR shifts and 2^{22} hash table lookups (which are assumed to be feasible in constant time).

As for the second phase, assuming the first occurrence of the actual history roughly took place for $t = 64$, the number of pairs of initial states we have to test is at most $(15 \cdot 128)^2 \simeq 2^{22}$, which is still practical (each test requires computation of the initialization of the stream cipher, and of a few words of keystream). Note that most of the time less than 2^{16} pairs will be tested before finding the right combination of indexes, and hence the key (as the right index is a random normal variable).

6 The Other Version of DICING

The second version of DICING, available from the ECRYPT web site [3], differs from the description made in section 3 in its output function, which becomes

$$z_t = \begin{cases} C(u_t, v_t) & \text{if } D'_{t-1} > D''_{t-1} \\ C(v_t, u_t) & \text{if } D'_{t-1} < D''_{t-1} \\ \varnothing & \text{if } D'_{t-1} = D''_{t-1} \end{cases} \qquad (10)$$

We remark three changes:

- The choice of the output function no longer depends on the comparison of the current dices, but rather of the previous ones. Note that although it does not formally change the state update function, its computation order is modified. As a matter of fact, this change amounts to updating LFSRs Γ_3 and Γ_4 *before* updating memories u_t and v_t.

- If both dices are equal, the output function outputs nothing (instead of $u_t \oplus v_t$).
- The new output function C used whenever both dices are different is no longer linear in any of its component; it is still key-dependent.

The first two changes prevent use of the distinguisher described in section 4. As a matter of fact, we still have $D'_t = D''_t \Rightarrow (u_{t-1} = u_{t+1}$ and $v_{t-1} = v_{t+1})$. But as $D'_t = D''_t$, there is no output corresponding to u_{t+1} and v_{t+1}. Otherwise said, one of the two repeating values does not appear anymore.

Our second attack (in section 5) obviously exploits partial linearity in the output function. As this linearity has been removed, it no longer works.

7 Conclusion

In this paper we have shown that one of the two versions of DICING is so weak that practical attacks can be mounted against it. The second version, which obviously appears more secure, is not vulnerable to these attacks as such.

References

[1] ECRYPT Stream Cipher Project. Part of the ECRYPT Network of Excellence in Cryptology, European Commission project IST-2002-507932. `http://www.ecrypt.eu.org/stream/`.

[2] Li An-Ping. A New Stream Cipher: DICING. In *Proceedings of the Symmetric Key Encryption Workshop.* Åarhus, Denmark, May 2005.

[3] Li An-Ping. A New Stream Cipher: DICING. Available at `http://www.ecrypt.eu.org/stream/dicing.html`.

A About Possible False Alarms

In this appendix, we consider the case where the attacker falsely assumes $a_t = 0$ (resp. $b_t = 0$), or falsely guesses the value of b_t (resp. a_t) when the first assumption is correct.

In the first case, i.e. if neither a_t nor b_t equals 0, $z_{t-1} \oplus z_{t+1}$ is nonlinear in either ω_t or τ_t, which makes very unlikely that the computed candidates for τ_t and ω_t have anything to do with their actual values. They can be considered as random.

The case where the assumption $a_t = 0$ (resp. $b_t = 0$) is correct and equation (8) (resp. (9)) is satisfied, but the guess on b_t (resp. a_t) is wrong, is more interesting. Consider two cycles t and t' such that (8) is satisfied. For the actual values b_t and $b_{t'}$ we have

$$\begin{cases} \tau_t = (z_{t-1} \oplus z_{t+1}) \cdot (1 \oplus x^{b_t})^{-1} \\ \tau_{t'} = (z_{t'-1} \oplus z_{t'+1}) \cdot (1 \oplus x^{b_{t'}})^{-1} \end{cases} \tag{11}$$

with

$$\tau_t = x^n \cdot \tau_{t'} \tag{12}$$

for some n. For false guesses b_t^* and $b_{t'}^*$, the attacker computes wrong values τ_t^* and $\tau_{t'}^*$:

$$\begin{cases} \tau_t^* = (z_{t-1} \oplus z_{t+1}) \cdot (1 \oplus x^{b_t^*})^{-1} \\ \tau_{t'}^* = (z_{t'-1} \oplus z_{t'+1}) \cdot (1 \oplus x^{b_{t'}^*})^{-1} \end{cases} \tag{13}$$

Putting equations (11), (12), (13) together, we obtain:

$$\tau_t^* \cdot \frac{1 \oplus x^{b_t^*}}{1 \oplus x^{b_t}} = x^n \cdot \tau_{t'}^* \cdot \frac{1 \oplus x^{b_{t'}^*}}{1 \oplus x^{b_{t'}}} \tag{14}$$

So when $b_t = b_{t'}$, there are 14 wrong guesses on the history of Γ_4: those corresponding to $b_t^* = b_{t'}^* \neq b_t$. However it happens with probability 1/15 only, and this problem can be solved by neglecting cycle t', and finding another clock cycle t'' such that (8) is satisfied, with $b_t \neq b_{t''}$.