Improved cryptanalysis of Py

Paul Crowley, paul@ciphergoth.org

LShift Ltd, www.lshift.net

Abstract We improve on the best known cryptanalysis of the stream cipher Py by using a hidden Markov model for the carry bits in addition operations where a certain distinguishing event takes place, and constructing from it an "optimal distinguisher" for the bias in the output bits which makes more use of the information available. We provide a general means to efficiently measure the efficacy of such a hidden Markov model based distinguisher, and show that our attack improves on the previous distinguisher by a factor of 2^{16} in the number of samples needed. Given 2^{72} bytes of output we can distinguish Py from random with advantage greater than $\frac{1}{2}$, or given only a single stream of 2^{64} bytes we have advantage 0.03.

Keywords: Py, symmetric cryptanalysis, hidden Markov model

1 Introduction

Py [2] is a candidate in the eSTREAM project to identify new stream ciphers that might be suitable for widespread adoption. It is a synchronous stream cipher with a 1300-byte internal state, and at each step produces eight bytes of output, organised as two four-byte words. Py is one of the fastest eSTREAM candidates in software.

[6] presents a distinguisher against this cipher. It defines an event L in the internal state of the cipher which occurs with probability roughly $2^{-41.91}$. When this event occurs, two output values can be guaranteed to be equal. This results in a very small linear bias in the output of Py, which can be detected with on the order of $\Pr[L]^{-2}$ samples.

Specifically, when the event occurs, two output words $O_{1,1}$ and $O_{2,3}$ are generated from three words of the internal state S, A, and B as follows:

$$O_{1,1} = (S \oplus B) + A$$
$$O_{2,3} = (S \oplus A) + B$$

This implies that the least significant bits of $O_{1,1}$ and $O_{2,3}$ are equal. [6] goes on to observe that there will also be biases in the more significant bits of $O_{1,1} \oplus O_{2,3}$.

In this paper, we show that a more effective distinguisher can be built using the same model of the cipher as the above by making use of all of the bits of $O_{1,1}$ and $O_{2,3}$ in concert rather than considering them separately. We use a hidden Markov model to trace the propogation of the unknown carry bits from least to most significant bit to calculate the exact probability that a given $O_{1,1}, O_{2,3}$ pair will be seen given that the event L takes place, and from this construct a distinguisher optimal for this model with the method described in [1]. We show that this results in a reduction in the number of samples needed by a factor of approximately 60552.

2 Description of Py

An understanding of the exact workings of Py is not needed to follow how our work builds on the work of [6], but we describe the round function here for completeness. Py operates on 32-bit words (treated as members of $\mathbb{Z}/2^{32}\mathbb{Z}$) and (8-bit) bytes. Its internal state in round i comprises

Algorithm 1 Py round function

$$\begin{aligned}
O_{1,i} &= (\mathrm{ROTL32}(s_i, 25) \oplus Y_i[256]) + Y_i[P_i[26]] \\
O_{2,i} &= (s_i \oplus Y_i[-1]) + Y_i[P_i[208]] \\
Y_{i+1} &= Y_i[-2 \ldots 256] \,\|\, ((\mathrm{ROTL32}(s_i, 14) \oplus Y_i[-3]) + Y_i[P_i[153]]) \\
P_{i+1} &= \begin{cases} P_i[1 \ldots k-1] \,\|\, P_i[0] \,\|\, P_i[k+1 \ldots 255] \,\|\, P_i[k] & k \neq 0 \\ P_i[1 \ldots 255] \,\|\, P_i[0] & k = 0 \end{cases} \\
&\quad \text{where} \quad k = Y_{i+1}[185] \bmod 256 \\
s_{i+1} &= \mathrm{ROTL32}(s_i + Y_{i+1}[P_{i+1}[72]] - Y_{i+1}[P_{i+1}[239]], (P_{i+1}[116] + 18) \bmod 32)
\end{aligned}$$

"$\|$" represents array concatenation.

- a 260-word array Y_i, indexed from -3 to 256
- a 256-byte array P_i indexed from 0 to 255 which always contains a permutation, and
- a word s_i.

The specification of Py in [2] describes the round function as two state-update functions with an output function inbetween. To simplify cryptanalysis, we mark the boundaries between rounds differently, so that we can consider the round function to be an output function followed by a state-update function combining both parts. This is consistent with the conventions of [6]. Algorithm 1 defines the output and state update functions; it produces two 32-bit output words $O_{1,i}, O_{2,i}$ in round i.

We do not specify the key/IV setup; like [6], for all of our results we model P_1, Y_1 and s_1 as independent and uniformly distributed, with P_1 uniformly distributed over permutations of bytes.

3 Sekar et al attack

[6] presents a distinguisher against Py that requires 8 bytes of output from each of $2^{83.82}$ distinct keystreams. The authors define an event L which is the combination of the following six conditions:

- $P_2[116] \equiv -18 \pmod{32}$
- $P_3[116] \equiv 7 \pmod{32}$
- $P_2[72] = P_3[239] + 1$
- $P_2[239] = P_3[72] + 1$
- $P_1[26] = 1$
- $P_3[208] = 254$

They show that $\Pr[L] \approx 2^{-41.91}$ (with the initial state is modelled as random as always). Defining

$$\begin{aligned}
A &= Y_1[1] \\
B &= Y_1[256] \\
S &= \mathrm{ROTL32}(s_1, 25)
\end{aligned}$$

they show that where L occurs,

$$\begin{aligned}
O_{1,1} &= (S \oplus B) + A \\
O_{2,3} &= (S \oplus A) + B
\end{aligned}$$

In particular, where $[A]_0$ is the low bit of A, this implies that $[O_{1,1} \oplus O_{2,3}]_0 = ([S]_0 \oplus [B]_0 \oplus [A]_0) \oplus ([S]_0 \oplus [A]_0 \oplus [B]_0) = 0$. The authors show that $\Pr[[O_{1,1} \oplus O_{2,3}]_0 = 0 | \neg L] = \frac{1}{2}$, and thus that

$$\begin{aligned}\Pr[[O_{1,1} \oplus O_{2,3}]_0 = 0] &= \Pr[[O_{1,1} \oplus O_{2,3}]_0 = 0 | L] \Pr[L] + \\ &\quad \Pr[[O_{1,1} \oplus O_{2,3}]_0 = 0 | \neg L] \Pr[\neg L] \\ &= \Pr[L] + \frac{1}{2}(1 - \Pr[L]) \\ &= \frac{1}{2}(1 + \Pr[L])\end{aligned}$$

The authors go on to estimate that this bias can be used to construct an effective distinguisher given roughly $\Pr[L]^{-2} \approx 2^{83.82}$ samples.

[6] defines a second event with the same probability which we term L', which is identical to L except that $P_2[72] = P_3[72] + 1$ and $P_2[239] = P_3[239] + 1$. The authors assert [4] that where L' occurs, $O_{1,1} = (\text{ROTL32}(S, 25) \oplus B) + A$ and $O_{2,3} = (\text{ROTL32}(S + 2K, 25) \oplus A) + B$ where S, K, A and B are all independent and uniformly random under the assumption of independent uniform randomness in the initial state. Where neither L nor L' occur, $O_{1,1}$ and $O_{2,3}$ are independent and uniformly random.

We now measure the exact efficacy of this distinguisher using [1] and show how to improve on it with a hidden Markov model.

4 Optimal distinguishers

[1] describes a general means to construct an efficient distinguisher between distributions D_0 and D_1 over a shared alphabet $\mathcal{Z}$, given n independent and identically distributed samples drawn from the unknown distribution D. $\Pr_{\mathsf{D}_j}[X]$ is shorthand for $\Pr[X | \mathsf{D} = \mathsf{D}_j]$ and $P_j(z) = \Pr_{\mathsf{D}_j}[D = z]$ where D is a random variable drawn from D. We consider only the case where $P_0(z) > 0$ and $P_1(z) > 0$ for all $z \in \mathcal{Z}$. Where $Z = z_1 \ldots z_n$ is the vector of samples, the efficacy of a distinguisher $\mathcal{A}$ is measured by its "advantage":

$$\text{Adv}(\mathcal{A}) = \Pr_{\mathsf{D}_1}[\mathcal{A}(Z) = 1] - \Pr_{\mathsf{D}_0}[\mathcal{A}(Z) = 1]$$

and [1] shows that the distinguisher $\mathcal{A}_{opt}$ defined here maximizes advantage given the information available:

$$\mathcal{A}_{opt}(Z) = \begin{cases} 1 & \text{where } P_1(Z) > P_0(Z) \\ 0 & \text{otherwise} \end{cases}$$

If we define the *log-likelihood ratio* function LLR below then (since each z_i is independent) $\mathcal{A}_{opt}$ can be expressed in a different way:

$$\text{LLR}(z) = \log\left(\frac{P_1(z)}{P_0(z)}\right)$$

$$\mathcal{A}_{opt}(Z) = \begin{cases} 1 & \text{where } \sum_i \text{LLR}(z_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Appealing to the central limit theorem, the authors show that where n is large, $\Pr_{\mathsf{D}_j}[\mathcal{A}_{opt}(Z) = 1] \approx \Phi\left(\frac{\sqrt{n}\mu_j}{\sigma_j}\right)$ where $\mu_j = \text{E}[\text{LLR}(\mathsf{D}_j)]$ and $\sigma_j^2 = \text{Var}[\text{LLR}(\mathsf{D}_j)]$. Next they define for every $z \in \mathcal{Z}$:

$$\epsilon_z = P_1(z) - P_0(z)$$

Where D_0 and D_1 are close (ie where $\epsilon_z \ll P_0(z)$ for all $z \in \mathcal{Z}$), they state that

$$-\mu_0 \approx \mu_1 \approx \frac{\beta}{2}, \quad \sigma_0^2 \approx \sigma_1^2 \approx \beta \quad \text{where} \quad \beta = \sum_{z \in \mathcal{Z}} \frac{\epsilon_z^2}{P_0(z)}$$

and thus that

$$\begin{aligned}
\mathrm{Adv}(\mathcal{A}_{opt}) &= \Pr_{\mathsf{D}_1}[\mathcal{A}_{opt}(Z) = 1] - \Pr_{\mathsf{D}_0}[\mathcal{A}_{opt}(Z) = 1] \\
&\approx \Phi\left(\frac{\sqrt{n}\mu_1}{\sigma_1}\right) - \Phi\left(\frac{\sqrt{n}\mu_0}{\sigma_0}\right) \\
&\approx \Phi\left(\frac{\sqrt{n}\beta}{2\sqrt{\beta}}\right) - \Phi\left(-\frac{\sqrt{n}\beta}{2\sqrt{\beta}}\right) \\
&= 1 - 2\Phi\left(-\frac{\sqrt{n\beta}}{2}\right)
\end{aligned}$$

For the distinguisher of [6] we have that

$P_j(z)$	$j = 0$	$j = 1$	ϵ_z
$z = 0$	$\frac{1}{2}$	$\frac{1}{2}(1 + \Pr[L])$	$\frac{1}{2}\Pr[L]$
$z = 1$	$\frac{1}{2}$	$\frac{1}{2}(1 - \Pr[L])$	$-\frac{1}{2}\Pr[L]$

where D_j is the distribution of $[O_{1,1}]_0 \oplus [O_{2,3}]_0$, from which we can deduce that in this instance $\beta = \Pr[L]^2$. Thus where $n = \Pr[L]^{-2}$ the advantage is approximately $1 - 2\Phi\left(-\frac{1}{2}\right) \approx 0.3829$; for an advantage greater than $\frac{1}{2}$, around 2^{85} samples (or 2^{88} bytes) are required. The presence of event L' makes no difference to the efficacy of this distinguisher.

5 Hidden Markov models

We can construct a more efficient distinguisher for Py by using a hidden Markov model [7,5]. We briefly reprise the theory of hidden Markov models here.

Consider a sequence of $n + 1$ random variables $Q_0 \ldots Q_n$ drawn from an alphabet of states $\Psi = \{S_1 \ldots S_N\}$. We say this sequence is generated by a first-order Markov process if the probability that Q_{i+1} is in state S_k depends only on the previous state Q_i, or in other words, if for all $0 \leq i < n$ and for all $q_0 \ldots q_{i+1} \in \Phi^{i+1}$

$$\Pr[Q_{i+1} = q_{i+1} | Q_0 \ldots Q_i = q_0 \ldots q_i] = \Pr[Q_{i+1} = q_{i+1} | Q_i = q_i]$$

We define the initial state vector π such that $\pi_i = \Pr[Q_0 = S_i]$, and the transition matrix M_i such that $(M_i)_{jk} = \Pr[Q_{i+1} = S_k | Q_i = S_j]$. The entries of π must sum to 1, as must each column of each M_i. For all the processes we consider here, each M_i will be the same and we drop the subscript i. π and M completely characterize the Markov process.

In a hidden Markov model, we also consider each transition[1] to also generate an output Y_i from an output alphabet $\mathcal{Y}$. We therefore define a transition matrix M_y for each possible output symbol $y \in \mathcal{Y}$ such that $(M_y)_{jk} = \Pr[Y_i = y \wedge Q_{i+1} = S_k | Q_i = S_j]$. For each state the probabilities of each output/next-state pair must sum to 1 as before, so each column of $\sum_{y \in \mathcal{Y}} M_y$ must sum to 1.

Given this matrix representation, if we define the vector $\mathbf{x} = M_{y_{n-1}} \ldots M_{y_0}\pi$ then $\mathbf{x}_i = \Pr[(Y_0 \ldots Y_{n-1}) = (y_0 \ldots y_{n-1}) \wedge Q_n = S_i]$ and thus the sum of the elements of $\mathbf{x}$ gives the probability of the output sequence $y_0 \ldots y_{n-1}$. This is known as the "forward algorithm".

[1] Following the practice described in section IV.C of [5], we specify outputs as produced on transitions, not from states

6 Applying the hidden Markov model to Py

In order to build a more efficient distinguisher using this method, we now consider the problem of calculating $\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3)|L]$. A naive algorithm for this, based on the observation that $\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3)|L] = \Pr[(S \oplus B) + A = o_1 \wedge (S \oplus A) + B = o_2] = |\{a, b, s \in (\mathbb{Z}/2^{32}\mathbb{Z})^3 | (s \oplus b) + a = o_1 \wedge (s \oplus a) + b = o_3\}|/2^{96}$, will take approximately 2^{96} operations. We use a hidden Markov model to calculate this exactly and efficiently.

Define

$$\mathrm{carry}(x, y) = (x + y) \oplus x \oplus y$$

it is well known (see eg [3]) that if $z = \mathrm{carry}(x, y)$ then $[z]_0 = 0$ and $[z]_{i+1} = \mathrm{maj}([x]_i, [y]_i, [z]_i)$ for $i \in 0 \ldots 30$ where maj is the binary majority function.

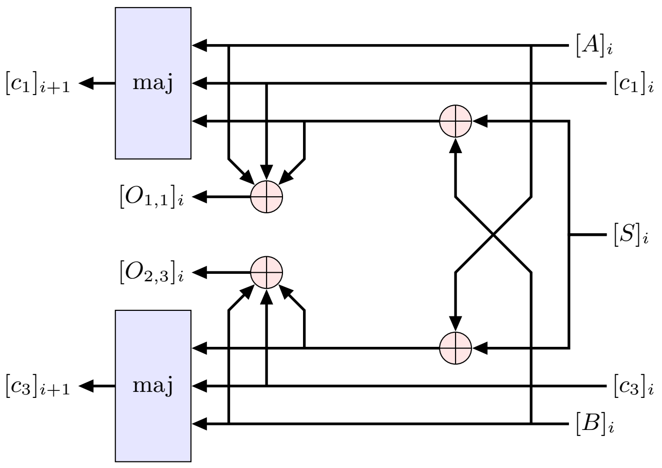

Figure 1. Calculating $[O_{1,1}]_i, [O_{2,3}]_i$

Following [6] we define $c_1 = \mathrm{carry}(S \oplus B, A)$ and $c_3 = \mathrm{carry}(S \oplus A, B)$. Our sequence of hidden states is the sequence of pairs of carry bits $([c_1]_i, [c_3]_i)$ for each bit i; the initial state is $([c_1]_0, [c_3]_0) = (0, 0)$ with probability 1, and the hidden Markov model tracks the propogation of these carry bits from least to most signficant bit in parallel across the two addition operations. Our outputs are pairs of bits $[O_{1,1}]_i, [O_{2,3}]_i$. Both the states and the outputs are drawn from the alphabet $\Psi = \mathcal{Y} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. A transition is represented in figure 1.

Each transition depends on the three independent uniform random bits $[A]_i$, $[B]_i$ and $[S]_i$. This gives us enough information to exactly specify the probability that a particular output and next state will result from a given state; it is determined by the number of $([A]_i, [B]_i, [S]_i)$ triples of bits that can produce this output/next state from that state. Given this model, the forward algorithm [5,7] can straightforwardly be used to exactly calculate $\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3)|L]$ for any (o_1, o_3) pair.

We determine the transition matrices below.

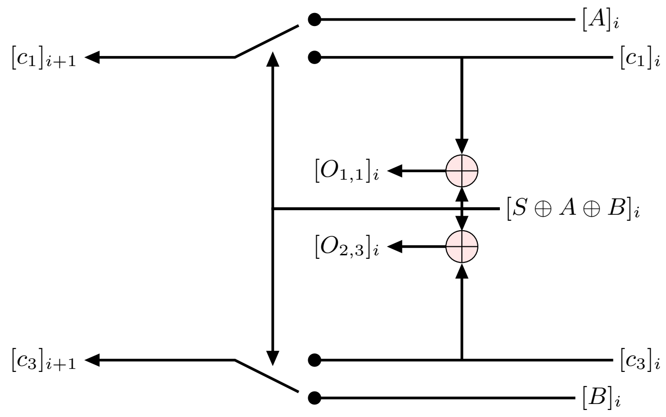

Figure 2. Simplification of figure 1

$$
\begin{aligned}
&\Pr\left[\begin{array}{c}[O_{1,1}]_i,[O_{2,3}]_i = w_1,w_3 \\ \wedge\ [c_1]_{i+1},[c_3]_{i+1} = v_1,v_3\end{array}\middle| L \wedge [c_1]_i,[c_3]_i = u_1,u_3\right] \\
&= \frac{\left|\left\{a,b,s\in\{0,1\}^3 \,\middle|\, \begin{array}{c} w_1 = s\oplus b\oplus a\oplus u_1 \wedge w_3 = s\oplus a\oplus b\oplus u_3 \\ \wedge\ v_1 = \mathrm{maj}(s\oplus b,a,u_1)\wedge v_1 = \mathrm{maj}(s\oplus a,b,u_3)\end{array}\right\}\right|}{8} \\
&= \frac{\left|\left\{a,b,s'\in\{0,1\}^3 \,\middle|\, \begin{array}{c} w_1 = s'\oplus u_1 \wedge w_3 = s'\oplus u_3 \\ \wedge\ v_1 = \mathrm{maj}(s'\oplus a,a,u_1)\wedge v_1 = \mathrm{maj}(s'\oplus b,b,u_3)\end{array}\right\}\right|}{8} \\
&= \frac{\left|\left\{a,b,s'\in\{0,1\}^3 \,\middle|\, \begin{array}{c} w_1 = s'\oplus u_1 \wedge w_3 = s'\oplus u_3 \\ \wedge\ v_1 = \mathrm{IF}(s',a,u_1)\wedge v_1 = \mathrm{IF}(s',b,u_3)\end{array}\right\}\right|}{8} \\
&= \begin{cases} \frac{1}{2} & \text{if } (u_1,u_3) = (v_1,v_3) = (\neg w_1,\neg w_3) \\ \frac{1}{8} & \text{if } (u_1,u_3) = (w_1,w_3) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

where

$$\mathrm{IF}(a,b,c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{if } a = 1 \end{cases}$$

This simplification (illustrated in figure 2) is achieved by defining $s' = a\oplus b\oplus s$. This yields the following transition matrices:

$$M_{(0,0)} = \begin{pmatrix} \frac{1}{8} & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & \frac{1}{2} \end{pmatrix},\ M_{(0,1)} = \begin{pmatrix} 0 & \frac{1}{8} & 0 & 0 \\ 0 & \frac{1}{8} & 0 & 0 \\ 0 & \frac{1}{8} & \frac{1}{2} & 0 \\ 0 & \frac{1}{8} & 0 & 0 \end{pmatrix},$$

$$M_{(1,0)} = \begin{pmatrix} 0 & 0 & \frac{1}{8} & 0 \\ 0 & \frac{1}{2} & \frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{8} & 0 \\ 0 & 0 & \frac{1}{8} & 0 \end{pmatrix}, \; M_{(1,1)} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{8} \\ 0 & 0 & 0 & \frac{1}{8} \\ 0 & 0 & 0 & \frac{1}{8} \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix}$$

Finally, we apply the forward algorithm described above, to yield the formula

$$\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3)|L] = (1\ 1\ 1\ 1) M_{([o_1]_{31},[o_3]_{31})} \ldots M_{([o_1]_0,[o_3]_0)} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

This more sophisticated model of $O_{1,1}, O_{2,3}$ yields some surprising results. For example if $O_{1,1}$ ends with the suffix 01^k for any k, then $O_{2,3}$ must end with the same suffix.

7 The Markov distinguisher

Now that we can efficiently calculate $\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3)|L]$, we can use the techniques from [1] presented in section 4 to directly construct a distinguisher from the probability model.

We examine n streams from n distinct key/IV pairs, and from each stream i we take a sample $z_i = O_{1,1}, O_{2,3}$, so our alphabet $\mathcal{Z}$ consists of all pairs of 32-bit words[2]. As above, we define $\mathrm{LLR}(z) = \log\left(\frac{P_1(z)}{P_0(z)}\right)$ and our distinguisher returns 1 iff $\sum_i \mathrm{LLR}(z_i) > 0$.

We do not yet take account of event L'; where L does not occur, we model $O_{1,2}, O_{2,3}$ as independent and uniformly random. This introduces a small error; we believe that the distinguisher will nevertheless be roughly as effective as advertised, but it is likely that a very slightly more effective distinguisher could be built by taking L' into account. Instead, we approximate $P_1(z)$ as $\Pr[(O_{1,1}, O_{2,3}) = z|L] \Pr[L] + P_0(z) \Pr[\neg L]$.

To find β for this distinguisher and thus discover the number of samples required for a given advantage, we proceed as follows:

$$\begin{aligned}
\beta &= \sum_{z \in \mathcal{Z}} \frac{(P_1(z) - P_0(z))^2}{P_0(z)} \\
&= |\mathcal{Z}| \sum_{z \in \mathcal{Z}} (P_1(z) - \frac{1}{|\mathcal{Z}|})^2 \\
&= |\mathcal{Z}| \sum_{z \in \mathcal{Z}} \begin{pmatrix} \Pr_{\mathsf{D}_1}[(O_{1,1}, O_{2,3}) = z|L]\ \Pr[L] \\ + \Pr_{\mathsf{D}_1}[(O_{1,1}, O_{2,3}) = z|\neg L]\ \Pr[\neg L] \\ - \frac{1}{|\mathcal{Z}|} \end{pmatrix}^2 \\
&= |\mathcal{Z}| \sum_{z \in \mathcal{Z}} \begin{pmatrix} \Pr_{\mathsf{D}_1}[(O_{1,1}, O_{2,3}) = z|L]\ \Pr[L] \\ + \frac{1}{|\mathcal{Z}|}(1 - \Pr[L]) \\ - \frac{1}{|\mathcal{Z}|} \end{pmatrix}^2 \\
&= |\mathcal{Z}| \Pr[L]^2 \sum_{z \in \mathcal{Z}} \left(\Pr_{\mathsf{D}_1}[(O_{1,1}, O_{2,3}) = z|L] - \frac{1}{|\mathcal{Z}|} \right)^2
\end{aligned}$$

[2] Two alphabets are at work in this distinguisher. The hidden Markov model works over an alphabet of pairs of bits $\mathcal{Y} = \{0,1\}^2$ to find the probability of a given pair of words; the optimal distinguisher constructed from it works on an alphabet of pairs of words $\mathcal{Z} = (\mathbb{Z}/2^{32}\mathbb{Z})^2$. Note that $|\mathcal{Z}| = |\mathcal{Y}|^{32}$.

We cannot directly compute this sum in reasonable time because $\mathcal{Z}$ has 2^{64} elements. However, we can define the following function family:

$$f_k(\mathbf{x}) = \sum_{\mathbf{y}\in\mathcal{Y}^k} ((1\ 1\ 1\ 1)M_{y_0}M_{y_1}\dots M_{y_{k-1}}\mathbf{x} - \frac{1}{|\mathcal{Z}|})^2$$

and from our formula for $\Pr[(O_{1,1}, O_{2,3}) = z|L]$ we see that

$$\beta = |\mathcal{Z}|\Pr[L]^2 f_{32}\left(\begin{pmatrix}1\\0\\0\\0\end{pmatrix}\right)$$

Furthermore, we can define f recursively:

$$f_0(\mathbf{x}) = ((1\ 1\ 1\ 1)\mathbf{x} - \frac{1}{|\mathcal{Z}|})^2$$
$$f_{k+1}(\mathbf{x}) = \sum_{y\in\mathcal{Y}} f_k(M_y\mathbf{x})$$

This by itself does not yield an efficient algorithm for finding β, since each evaluation of f_{k+1} requires four evaluations of f_k. However, we now show by induction that there exists a series of matrices $A_0 \dots A_{32}$ such that $f_k(\mathbf{x}) = \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}^T A_k \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}$. A_0 is simply a 5x5 matrix whose every entry is 1, and

$$\begin{aligned}
f_{k+1}(\mathbf{x}) &= \sum_{y\in\mathcal{Y}} f_k(M_y\mathbf{x})\\
&= \sum_{y\in\mathcal{Y}} \begin{pmatrix}M_y\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}^T A_k \begin{pmatrix}M_y\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}\\
&= \sum_{y\in\mathcal{Y}} \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}^T \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix}^T A_k \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix} \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}\\
&= \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}^T \left(\sum_{y\in\mathcal{Y}} \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix}^T A_k \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix}\right) \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}\\
&= \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}^T A_{k+1} \begin{pmatrix}\mathbf{x}\\-\frac{1}{|\mathcal{Z}|}\end{pmatrix}
\end{aligned}$$

where

$$A_{k+1} = \sum_{y\in\mathcal{Y}} \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix}^T A_k \begin{pmatrix}M_y & 0\\0 & 1\end{pmatrix}$$

We can therefore use this algorithm to find A_{32} recursively, from which we find that $\beta \approx 60552\Pr[L]^2$. For a distinguisher with the same advantage as that of [6], we therefore need only $n = \left\lceil\frac{1}{\beta}\right\rceil \approx \frac{1}{60552}\Pr[L]^{-2}$ samples.

8 Conclusions and further work

We have shown that Py can be distinguished from a random function given roughly a factor of 2^{16} fewer samples than the previous best attack in [6]. We prefer to state the number of samples needed to gain advantage greater than $\frac{1}{2}$; with 2^{69} samples—ie 2^{72} bytes—the attack has an advantage of around 0.53. Like that attack, this attack is not restricted to using words at the start of the stream to build the distinguisher; it may use nearly the entire stream. This means that there will be correlations between samples, but those correlations are unlikely to affect the efficacy of the attack. Py is limited to producing 2^{64} bytes from a single key/IV pair, which is equivalent to just under 2^{61} samples, so we gain advantage greater than $\frac{1}{2}$ once the complete streams from roughly 2^8 different key/IV pairs are used. Surprisingly, this attack is disallowed by the security goals set out in [2], which limit the attacker to at most 2^{64} bytes of keystream total. Against a single complete stream, our attack offers advantage 0.03, which is low but perhaps not negligible.

We did not take account of event L' defined in section 3. We anticipate that if we did so, we would need fewer samples still. Extending the hidden Markov model to find $\Pr[(O_{1,1}, O_{2,3}) = (o_1, o_3) | L \vee L']$ is not hard—a single bit may be added to the state indicating which of L or L' took place—but we have not yet done the work of estimating β for this extended model.

9 Acknowledgements

Thanks to Souradyuti Paul for invaluable clarification of [6], to Shahram Khazaei for the suggestion of considering $O_{1,1}$ and $O_{2,3}$ separately, and to Matthias Radestock for useful commentary on my drafts.

References

1. Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450. Springer, 2004.
2. Eli Biham and Jennifer Seberry. Py (Roo) : A fast and secure stream cipher using rolling arrays. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023, 2005.
3. Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.
4. Souradyuti Paul. Re: Improved cryptanalysis of Py. Personal emails, December 2006.
5. Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. pages 267–296, 1990.
6. Gautham Sekar, Souradyuti Paul, and Bart Preneel. Distinguishing attacks on the stream cipher Py. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/081, 2005.
7. Wikipedia. Hidden Markov model — Wikipedia, the free encyclopedia, 2006. [Online; accessed 19-January-2006].

http://www.ciphergoth.org/crypto/py/