Proposal of primitive polynomials for Linux kernel PRNG

David FONTAINE
7 Rue Claude Chappe
35510 Cesson-Sévigné
France
david.fontaine@capgemini.com

Olivier VIVOLO
Orange
30 rue du chêne Germain
35510 Cesson-Sévigné
France
olivier.vivolo@orange.com

July 25, 2017

Abstract

The polynomials defining the LFSRs of the linux Kernel PRNG are irreducible but not primitive. As a result, the space of numbers generated by these LFSRs does not fill all the space. We propose in this paper more optimal polynomials which increase by a factor of 3 the space of the random numbers generated by these LFSRs. The polynomials used in the current implementation of the PRNG and the point presented here, do not conclude a practical attack on the PRNG.

1 Short description of context

The PRNG of the latest version of the Linux kernel at the time of writing this article (v.4.13) is defined by two following polynomials of LFSR ([2])

$$P_1(X) = x^{128} + x^{104} + x^{76} + x^{51} + x^{25} + x + 1,$$

$$P_2(X) = x^{32} + x^{26} + x^{19} + x^{14} + x^7 + x + 1.$$

The first one P_1 is used for the input pool and the second P_2 for output pool see figure 1. For futher information about the Linux kernel PRNG, we recommend the paper [3].

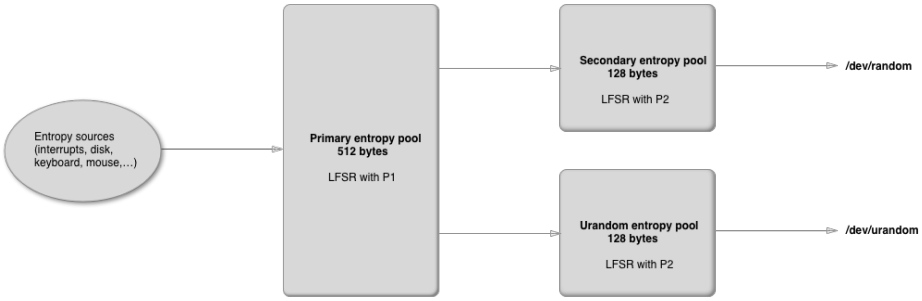

Figure 1: The general structure of the Linux kernel LRNG

After the publication of research paper on the mixing funtions by [3], the Linux kernel team decided to use these polynomials proposed by the authors of [3] to improve the TGFSR. Unfortunately, it is stated in the paper [3] that these polynomials have periods $(2^{(32deg(Pi))} - 1)/3, i = 1, 2$. In fact, the polynomials $Q_1(X) = \alpha^3(P1(X) - 1) + 1$ and $Q_2(X) = \alpha^3(P_2(X) - 1) + 1$ are not primitive over $GF(2^{32})$, where α is a primitive element of $GF(2^{32})$.

2 New possible polynomials

After several calculations and wanting the new polynomials very close to the current ones, we propose here the new polynomials R_1 for input pool and R_2 for the output pool,

$$R_1(x) = x^{128} + x^{106} + x^{79} + x^{51} + x^{25} + x + 1,$$

$$R_2(x) = x^{32} + x^{27} + x^{21} + x^{14} + x^7 + x + 1.$$

The polynomials $S_1(X) = \alpha^4(R_1(X)-1)+1$ and $S_2(X) = \alpha^4(R_2(X)-1)+1$ are primitive on $GF(2^{32})$ and their periods are $2^{32deg(P_i)} - 1, i = 1, 2$, the maximum possible periods. As mentionned in [3], the long period of the Linux kernel LFSR can no longer be guaranteed, and the process is no longer a linear function of the initial state but we consider that to replace the polynomials by the new ones can improve the LFSRs.

These polynomials can be checked easily as primitive (see Annex A). For implementing them in the source code of Linux kernel, we provide a patch see Annex B. Moreover, we informed the Linux kernel team of our proposal with a post to the mailing list [1] one year ago.

References

[1] Fontaine David and Vivolo Olivier. Proposal to modify the lfsr used by linux kernel. *http://www.mail-archive.com/linux-crypto@vger.kernel.org/msg21287.html*, september 2016.

[2] Linux kernel team. The source code of the prng implemented in the linux kernel. *https://github.com/torvalds/linux/blame/master/drivers/char/random.c.*

[3] Lacharme Patrick, Rök Andrea, Strubel Vincent, and Videau Marion. The linux pseudorandom number generator revisited. *http://eprint.iacr.org/2012/251.pdf*, page 23, may 2012.

A How to check that the polynomials are primitive

It is very easy to check that the polynomials R_1, R_2 are primitive with magma and the following code:

```
K0:=GF(2);
P<X> := PolynomialRing(K0);
K1<a> :=
ext<K0|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+1>;K1;
P<t> := PolynomialRing(K1);

R1 := t^128 + t^106 + t^79 + t^51 + t^25 + t + 1;
S1 := a^4*(R1-1)+1;

R2 := t^32 + t^27 + t^21 + t^14 + t^7 + t + 1;
S2 := a^4*(R2-1)+1;

S1test := Evaluate((1/(t^128))*S1,1/t);S1test;
> t^128 + a*t^127 + a*t^100 + a*t^74 + a*t^53 + a*t^25 + a

S1test := t^128 + a^4*t^127 + a^4*t^103 + a^4*t^77 + a^4*t^49 + a^4*t^22
    + a^4;
IsPrimitive(S1test);
> true

S2test := Evaluate((1/(t^32))*S2,1/t);S2test;
> t^32 + a*t^31 + a*t^24 + a*t^16 + a*t^12 + a*t^6 + a

S2test := t^32 + a^4*t^31 + a^4*t^25 + a^4*t^18 + a^4*t^11 + a^4*t^5 +
    a^4;
IsPrimitive(S2test);
> true
```

B Patch of random.c

To use these polynomials, the following changes in the random.c file should be applied:

```
olivier@Zebulon:~/Documents/linux-4.7.4/drivers/char$ diff random.c
    random-new.c
371,372c371,373
<       /* x^128 + x^104 + x^76 + x^51 +x^25 + x + 1 */
<       { S(128),       104,    76,      51,      25,      1 },
---
>       /* was: x^128 + x^104 + x^76 + x^51 +x^25 + x + 1 */
>       /* x^128 + x^106 + x^79 + x^51 +x^25 + x + 1 */
>       { S(128),       106,    79,      51,      25,      1 },
374,375c375,377
<       /* x^32 + x^26 + x^19 + x^14 + x^7 + x + 1 */
<       { S(32),         26,      19,      14,      7,        1 },
---
>       /* was: x^32 + x^26 + x^19 + x^14 + x^7 + x + 1 */
>       /* x^32 + x^27 + x^21 + x^14 + x^7 + x + 1 */
>       { S(32),         27,      21,      14,      7,        1 },
478a481
> /* was:
```

```
481a485,490
> */
> static __u32 const twist_table[16] = {
>       0x00000000, 0x1db71064, 0x3b6e20c8, 0x26d930ac,
>        0x76dc4190, 0x6b6b51f4, 0x4db26158, 0x5005713c,
>        0xedb88320, 0xf00f9344, 0xd6d6a3e8, 0xcb61b38c,
>        0x9b64c2b0, 0x86d3d2d4, 0xa00ae278, 0xbdbdf21c };
525c534,536
<                  r->pool[i] = (w >> 3) ^ twist_table[w & 7];
---
>                  /*
>                 was: r->pool[i] = (w >> 3) ^ twist_table[w & 7];*/
>                  r->pool[i] = (w >> 4) ^ twist_table[w & 15];
```