A Circuit-Based Attack on a Linear Congruential Pseudorandom Number Generator

March 6, 2011

Abstract

In this chapter, I will describe the steps necessary to break the pseudorandom number generator used in conjunction with the AES pipeline. In order to break the generator, an attacker must determine the values of constants in the generating equation. Solving for these constants requires the extended Euclidean algorithm, which is a recursive algorithm requiring many stages of calculation. In the end, I will demonstrate that an optimal circuit for breaking the generator will require a large circuit area that would be difficult to conceal as a backdoor on an AES accelerator.

1 Linear congruential generators

Linear congruential generators are a class of generators that have the form

$$X_n = (aX_{(n-1)} + c) mod\, m$$

where a, c, and m are constants. m is often chosen to be the word size of the computer architecture. When m is a power of 2, a should be chosen so that $a\, mod\, 8 = 5$. c should share no common factor with m[1].

2 Attacking the linear congruential generator

In order to break the PRNG, an attacker must determine the generating algorithm and the constants used in the algorithm, thereby gaining the ability to predict the next number in a string of random numbers. We assume that the attacker knows that the generating algorithm is a linear congruential generator (which is easily determined[2]), and we assume that the attacker may record a history of recently generated numbers.

2.1 Determining constant m

The attacker may be able to determine the value of m via prior knowledge of the computer architecture. Due to the speed requirement on the PRNG, the PRNG designer does not have many choices for the value of m.

2.2 Determining constant a

The value of a can be solved for algebraically once the attacker records three consecutive outputs of the generator, x,y, and z.

From the generating equation of the PRNG,

$$y \equiv ax + c \; (mod\, m) \tag{1}$$

$$z \equiv ay + c \; (mod\, m) \tag{2}$$

Subtracting the two equations we obtain:

$$(z - y) \equiv a(y - x) \; (mod\, m)$$

Determining the value of a requires solving this equation in modular space to isolate a, which would involve finding the modular inverse of $(y - x)$. Finding the modular inverse of a number is more involved than simple division, and we will investigate it calculation method in the next section.

2.3 Determining constant c

Again from equations 1and 2, we obtain an equation to solve for c.

$$(y^2 - zx) \equiv c(y - x) \; (mod\, m)$$

Note that solving for c also requires finding the modular inverse of $(y - x)$.

3 Solving for modular inverses using the extended Euclidean algorithm

The optimal method for determining the modular inverse is via the extended Euclidean algorithm[3]. In the following section, we manipulate the general form of equations that the extended Euclidean algorithm to show that the algorithm may be used to break the PRNG.

3.1 The extended Euclidean algorithm

The extended Euclidean algorithm solves for equations of the form

$$pu + qv = gcd(p, q)$$

where p and q are known constants.

In the special case that p and q are coprime (have no common factors), the equation takes on the special form

$$pu + qv = 1$$

This equation can be written in modular arithmetic form:

$$pu \equiv 1 \ (mod\, q) \tag{3}$$

In this form, v is the number of modular bases, q, required to satisfy the identity. We ignore the significance of v.

3.2 Applying the extended Euclidean algorithm

Our goal is to solve for the modular inverse of $(y-x)$, which, expressed in modular arithmetic form, is to solve for the equation

$$(y-x)(y-x)^{-1} \equiv 1 \ (mod\, m) \tag{4}$$

This equation is identical in form to 3, so long as our two known variables $(y-x)$ and m are coprime. $(y-x)$ will be odd because the outputs of the PRNG alternate between even and odd numbers on each cycle—this is guaranteed by the requirement that c must be odd. m is the word size of the computer architecture, and therefore is a power of 2. With $(y-x)$ odd and m a power of 2, our known variables are coprime; therefore, 4fits the form of3.

References

[1] Knuth, D. E. (1981). Seminumerical Algorithms. In D. E. Knuth, The Art of Computer Programming (Vol. 2, pp. 9-37, 170). Reading, Mass.: Addison-Wesley.

[2] Marsaglia, G. (2003). Random Number Generators. Journal of Modern Applied Statistical Methods , 2 (1), 2-13.

[3] Liu, C.-L., Horng, G., & Liu, H.-Y. (2008). Computing the modular inverses is as simple as computing the GCDs. Finite Fields and Their Applications , 14 (1), 65-75.