A Simple Power Analysis Attack on the Serpent Key Schedule

Kevin J. Compton* Brian Timm† Joel VanLaven‡
Computer Science and Engineering Division
University of Michigan - Ann Arbor
Ann Arbor, MI 48109-2212, USA

September 24, 2009

Abstract

We describe an SPA attack on an 8-bit smart card implementation of the Serpent block cipher. Our attack uses measurements taken during an on-the-fly key expansion together with linearity in the cipher's key schedule algorithm to drastically reduce the search time for an initial key. An implementation finds 256-bit keys in 3.736 ms on average. Our work shows that linearity in key schedule design and other cryptographic applications should be carefully evaluated for susceptibility to side-channel attacks and that search algorithm design can greatly speed up side-channel attacks.

Keywords: Serpent, SPA, Power Attack, Linearity, Block Cipher.

1 Introduction

In 1997, the National Institute of Standards and Technology issued a call for proposals for the Advanced Encryption Standard (AES), a block cipher that would replace the Data Encryption Standard (DES). Serpent [1] was selected as one of five finalists, and although another cipher, Rijndael, ultimately triumphed, Serpent was a strong contender in the final round and is still available for use. The principles used in Serpent's design provide insights for future cryptosystem design, so it is instructive to assess its vulnerability to certain types of cryptanalysis. We show that the Serpent key schedule algorithm is susceptible to a side-channel attack related to its use of a modified linear feedback shift register (LFSR). Since LFSRs are very common in block cipher design and other cryptographic applications (*cf.* Schneier [14]), our results suggest that cryptographic applications employing LFSRs and modified LFSRs should be carefully evaluated when side-channel attacks are a concern. Our work also shows that search algorithm design, in this case based on standard techniques from linear algebra, can greatly lower the expected time for a side-channel attack.

We consider side-channel attacks known as a *power analysis attacks*, which are performed by measuring the power utilization of a processor or ASIC as it performs cryptographic

*kjc@umich.edu
†timm@umich.edu
‡jvanlav@umich.edu

operations. The first investigation of power analysis attacks was due to Kocher *et al.* [4]. They observed that the power consumed by a CMOS chip varies depending on the values operated on, so attackers may obtain information about data manipulated by the low-level processor. Since microprocessors perform operations on fixed-sized blocks of data, these variations actually reveal the sum of the bits, or the Hamming weight, of each data block. We base our attack on empirical studies which have shown the feasibility of determining Hamming weights of byte-length register loads executed by smart cards [8, 10, 7, 12].

Kocher *et al.* [4] identify two types of power attacks: differential power analysis (DPA) attacks and simple power analysis (SPA) attacks. A DPA attack correlates particular plaintext or key bits with variations in the power profiles from a large number of encryptions made with the same key. An SPA attack may, in theory, require only a single power profile, though in practice measurement error often necessitates multiple executions. It is generally more difficult to determine whether a particular cipher is susceptible to an SPA attack because this form of attack usually depends on particular vulnerabilities in the cipher design (*e.g.*, in the case of Serpent, linearity in the key schedule algorithm).

Validation and testing of block ciphers often focus on their encryption algorithms [15], but security of key expansion algorithms is also important, particularly when considering side-channel attacks. In 1999 Biham and Shamir [2] gave a very preliminary appraisal of power attack susceptibility of various AES candidate ciphers. They observed that the Serpent key schedule applies a linear feedback shift register to the initial key to generate the entire prekey (followed by an application of non-linear S-boxes to create the final round key), but speculated that the Hamming weights of the intermediate key values will most likely not yield any useful information about the key. The results here refute this speculation, illustrating that SPA attacks are not always easy to discern.

Previous work has produced power analysis attacks on DES [10] and AES (Rijndael). Those on AES include both DPA attacks [11] and SPA attacks on the key schedule algorithm [6, 16]. The work here is, to the best of our knowledge, the first power analysis attack on Serpent and the first that makes extensive use of the properties of LFSRs. Several researchers have looked at techniques for thwarting power analysis attacks [9, 7], but we do not consider their applicability here.

2 The Serpent Block Cipher

Serpent was designed and submitted as an AES candidate proposal by Ross Anderson, Eli Biham, and Lars Knudsen [1]. We will follow their notation and terminology; in particular, all values are represented in little-endian notation. In this section we present a brief description of Serpent's encryption algorithm and key schedule algorithm.

2.1 The Serpent Encryption Algorithm

Serpent is a 32-round substitution-permutation cipher with 128-bit block size and a 256-bit initial key. Both the encryption algorithm and the key schedule algorithm use 8 S-boxes $S_0, S_1, S_2, \ldots, S_7$, each of which maps a 4-bit input to a 4-bit output. The encryption algorithm first applies an initial permutation IP to a 128-bit block plaintext. Each of the following 31 rounds, numbered $0, 1, \ldots, 30$, follows the same pattern. In round j, the 128-bit input block is XOR-ed with a round key $\hat{K}_j$, then an S-box substitution is effected by replicating $S_{(j \bmod 8)}$ 32 times and applying it in parallel, and finally a linear transformation is applied. The last round (round 31) differs only in that the linear transformation is replaced an additional XOR with a round key $\hat{K}_{32}$. After the last round, the algorithm applies the permutation IP^{-1}.

2.2 The Serpent Key Expansion Algorithm

Encryption with Serpent requires a 256-bit initial key, but the input key supplied by the user can be any length up to 256 bits. To any initial key of length less than 256, the key schedule algorithm appends (on the MSB end) a 1 followed by enough 0's to increase the total key length to exactly 256 bits.

Throughout this paper, w_i will denote a 32-bit word, b_i will denote a single bit with $w_i = b_{32i} b_{32i+1} \cdots b_{32i+31}$. The *initial key* (following the indexing scheme used in [1]) is

$$w_{-8} w_{-7} w_{-6} \cdots w_{-1} = b_{-256} b_{-255} b_{-254} \cdots b_{-1},$$

from which the *prekey*

$$w_0 w_1 w_2 \cdots w_{131} = b_0 b_1 b_2 \cdots b_{4223}$$

is derived using the recurrence

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11. \tag{1}$$

Here $\oplus$ is bitwise-XOR, i is the 32-bit binary representation of the integer i, ϕ is the first 32 bits of the binary representation the fractional part of the Golden Ratio (in hexadecimal this is 9e3779b9), and $\lll 11$ indicates a left rotational shift by 11 bits.

Next, words $k_0, k_1, k_2 \cdots k_{131}$ are derived using the eight S-boxes as follows. S-box $S_{((3-j) \bmod 8)}$ is replicated 32 times and applied in parallel to $w_{4j} w_{4j+1} w_{4j+2} w_{4j+3}$ to obtain $k_{4j} k_{4j+1} k_{4j+2} k_{4j+3}$. The round keys used by Serpent's encryption algorithm are $\hat{K}_j = IP(k_{4j} k_{4j+1} k_{4j+2} k_{4j+3})$, for $j = 0, 1, 2, \ldots, 32$.

2.3 Serpent Key Expansion Observations

The designers of Serpent refer to (1) as an "affine recurrence," as both XOR and left rotational shifts are linear operations. Due to the 11-bit left rotation, this is not the type of recurrence occurring in the standard definition of an LFSR [14] (or in the more general definition of an LFSR over a finite field [5]). However, we may use a *modified* LFSR that stores 8 consecutive 32-bit words of the output stream. The next word w_i in the output stream is computed as a linear function of the register contents. Following this is a 32-bit register shift (rather than a 1-bit shift, as in a true LFSR) and the insertion of w_i in the register.

Our attack exploits the linear relationship between bits of Serpent's prekey. While the Serpent key schedule does include a set of S-boxes to eliminate this linearity in the actual round keys used for encryption, they are not introduced until after the entire prekey has been generated. As a result, every prekey bit $b_0, b_1, b_2, \ldots, b_{4223}$ can be computed using some subset of bits from the 256-bit initial key $b_{-256}b_{-255}\cdots b_{-1}$ along with one bit to represent the aggregated constants from i and ϕ. We provide the details in the next section.

The 11-bit left rotational shift could be used to obtain additional Hamming weight measurements and reduce the search space. Typical smart card architectures, including those used in smart card implementations of Serpent published during the AES evaluation period [3, 13], are equipped with only single-bit rotate and shift operations. However, rather than assuming that each rotation will require distinct single-bit shift operations, we assume that power traces are measured only immediately after the 11-bit shift. While the pre-shift measurements are not used in the present version of our attack, the additional information they provide would likely be valuable for error correction and other attack optimizations.

Our attack is restricted to smart card implementations where the key expansion is performed on the fly. It would still be possible to obtain power measurements if the entire set of round keys were pre-loaded into memory, but we would only obtain the Hamming weights of bytes after the S-box transformation. The linearity that we exploit among bits of the intermediate prekey is not present in the final round keys due to the non-linear design of the S-boxes. However, for many smart cards where memory usage and size are critical factors, pre-loading and storing 33 128-bit round keys would be prohibitively expensive, particularly since many protocols call for frequent key replacement. Additionally, many documents from the AES evaluation process reported key expansion times and praised the low RAM usage made possible by Serpent's on-the-fly key schedule, implying that on-the-fly key expansion is a reasonable expectation in most cases [13, 15].

3 Key Schedule Power Analysis Attack

We begin by rewriting recurrence (1) to express the relation between bits rather than words. Regard these expressions as linear equations over $\mathbf{GF}(2)$, the field of order 2, with $+$ and $\cdot$ denoting addition and multiplication modulo 2. To simplify notation, let ϕ_i be the bit in position i in the binary representation of the fractional part of ϕ, $\mathrm{bin}_i(j)$ be the bit in position i in the 32-bit binary representation of j, and

$$c_i = \phi_{((i+11) \bmod 32)} + \mathrm{bin}_{((i+11) \bmod 32)}(i).$$

We require two cases to handle wraparound from the 11-bit rotational shift. For $0 \leq i \leq 4223$,

$$b_i = b_{i-21} + b_{i-85} + b_{i-149} + b_{i-245} + c_i, \qquad \text{if } 0 \leq (i \bmod 32) \leq 20; \tag{2}$$

$$b_i = b_{i-53} + b_{i-117} + b_{i-181} + b_{i-277} + c_i, \qquad \text{if } 21 \leq (i \bmod 32) \leq 31. \tag{3}$$

This recurrence uses two different linear combinations of previous bits to produce an output bit, whereas a true LFSR uses only one.[1]

[1] If allowed to proceed indefinitely, the Serpent key expansion would produce an ultimately periodic sequence, so it could, in principle, be implemented by an LFSR. However, the register length would be prohibitively large.

By iterating (2) and (3) we can derive a expression for each bit b_i, $0 \le i \le 4223$, of the prekey as a linear combination of bits b_i, $-256 \le i \le -1$, of the initial plus a constant:

$$b_i = \sum_{j=0}^{255} a_{i,j} \cdot b_{j-256} + d_i, \quad \text{for } i = 0, \ldots, 4223.$$

Taking

$$\begin{aligned} \mathbf{b} &= (b_0, b_1, \ldots, b_{4223})^t, \\ \mathbf{d} &= (d_0, d_1, \ldots, d_{4223})^t, \\ \mathbf{x} &= (b_{-256}, b_{-255}, \ldots, b_{-1})^t, \end{aligned}$$

(where $\mathbf{v}^t$ is the transpose of vector $\mathbf{v}$) and

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,255} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,255} \\ \vdots & \vdots & \ddots & \vdots \\ a_{4223,0} & a_{4223,1} & \cdots & a_{4223,255} \end{pmatrix},$$

we can rewrite the system of equations as

$$\mathbf{b} = \mathbf{A}\,\mathbf{x} + \mathbf{d}. \tag{4}$$

Note that if we run the key schedule algorithm on an initial key $\mathbf{x} = 0$, the resulting prekey will be $d_0 d_1 \cdots d_{4223}$, so we can compute $\mathbf{d}$. If we run the key schedule algorithm on the initial key with bit b_{j-256} equal to 1 and all other bits 0, the resulting prekey will be $a_{0,j} a_{1,j} \cdots a_{4223,j} \oplus d_0 d_1 \cdots d_{4223}$, so we can compute the columns of $\mathbf{A}$. From equation (4), we can solve for $\mathbf{x}$ if we know $\mathbf{b}$. Unfortunately, power analysis gives only the Hamming weights of the bytes that comprise $\mathbf{b}$, not $\mathbf{b}$ itself. It is difficult to work with the Hamming weights in $\mathbf{GF}(2)$, but the *parity* of the Hamming weight of a byte is just the sum of its bits in $\mathbf{GF}(2)$. Thus, denoting the parity of the i-th byte by h_i for $i = 0, 1, \ldots, 527$, we have the following system of linear equations in $\mathbf{GF}(2)$:

$$h_i = \sum_{j=0}^{7} b_{8i+j}, \quad \text{for } i = 0, 1, \ldots, 527.$$

Again, rewrite this system in matrix notation, taking $\mathbf{b}' = (h_0, h_1, \ldots, h_{527})^t$ and $\mathbf{T}$ to be a 528×256 matrix such that $\mathbf{b}' = \mathbf{T}\,\mathbf{b}$. Apply $\mathbf{T}$ to both sides of (4) and set $\mathbf{A}' = \mathbf{T}\,\mathbf{A}$, $\mathbf{d}' = \mathbf{T}\,\mathbf{d}$. The result is

$$\mathbf{b}' = \mathbf{A}'\,\mathbf{x} + \mathbf{d}',$$

a system of 528 linear equations in 256 unknowns. We know $\mathbf{b}'$, $\mathbf{A}'$, and $\mathbf{d}'$, so we solve for $\mathbf{x}$ by standard methods.

Set $\mathbf{e}' = \mathbf{b}' + \mathbf{d}'$ and use Gauss-Jordan elimination to reduce the augmented matrix $[\mathbf{A}'|\mathbf{e}']$ to reduced row echelon form $[\mathbf{A}''|\mathbf{e}'']$ (we used the computer algebra program Mathematica to do this); $\mathbf{A}''$ turns out to be of the form

$$\mathbf{A}'' = \begin{pmatrix} \mathbf{D} & & & & \\ & \mathbf{D} & & 0 & \\ & & \mathbf{D} & & \\ & & & \ddots & \\ & & & & \mathbf{D} \\ & 0 & & & \end{pmatrix}$$

where $\mathbf{D}$ is the 25×32 matrix in Figure 1. More precisely, $\mathbf{A}''$ is formed as follows. Replace the main diagonal entries of an 8×8 matrix with D and all entries off the main diagonal with the 25×32 0-matrix (this is the tensor product $\mathbf{I} \otimes \mathbf{D}$ where $\mathbf{I}$ is the 8×8 identity matrix); then add an additional 328 rows of 0's to form a 528×256 matrix.

$$\mathbf{D} = \left(\begin{array}{cccccccccccccccccccccccccccccccc}
1 & - & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
- & 1 & - & 1 & - & - & - & - & - & - \\
- & - & 1 & - & 1 & - & - & - & - & - \\
- & - & - & 1 & - & 1 & - & - & - & - \\
- & - & - & - & 1 & - & 1 & - & - & - \\
- & - & - & - & - & 1 & - & 1 & - & - \\
- & - & - & - & - & - & 1 & - & 1 & - \\
- & - & - & - & - & - & - & 1 & - & 1 \\
- & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
- & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & 1 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & 1 & - & - & - & - \\
- & 1 & - & - & - & - & - & - & - & 1 & - & - & - \\
- & 1 & - & - & - & - & - & - & - & 1 & - & - \\
- & 1 & - & - & - & - & - & - & - & 1 & - \\
- & 1 & - & - & - & - & - & - & - & 1 \\
- & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right)$$

Figure 1: The 25×32 matrix $\mathbf{D}$. For readability, 0's are replaced with hyphens.

Thus, the rank of $\mathbf{A}'$ is 200 so the parities of the prekey Hamming weights contain 200 bits of information about the 256-bit initial key. Therefore, among the 256 bits of the initial key there are 56 bits (whose positions can be determined in advance) that will determine the other 200 bits of the initial key. That is, the search space is of size 2^{56}, already a significant improvement over a brute-force algorithm. Examination of $\mathbf{A}''$ shows an even more dramatic reduction in search space size because the linear system $\mathbf{A}''\,\mathbf{x} = \mathbf{e}''$ can be decomposed into 8 linear systems of the form

$$\mathbf{D}\,\mathbf{y} = \mathbf{c} \tag{5}$$

For example, to solve for w_{-8} we take $\mathbf{c} = (e''_0, e''_1, \ldots, e''_{24})^t$. The linear system (5) has 25 equations and 32 unknowns, hence 2^7 solutions; they are given by vectors of the form $\mathbf{c}' + \mathbf{v}$, where $\mathbf{c}'$ is the 32-dimensional vector formed by appending seven 0's to $\mathbf{c}$, and $\mathbf{v}$ is a vector in the null space of $\mathbf{D}$. Since $\mathbf{D}$ is in reduced row echelon form, the last seven columns of $\mathbf{D}$, suitably extended, determine a basis for the null space of $\mathbf{D}$ (see Figure 2). By taking all possible linear combinations of these basis vectors, we generate the 2^7 vectors $\mathbf{v}$ needed for a solution.

$$
\begin{aligned}
\mathbf{v}_1 &= (1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-})^t,\\
\mathbf{v}_2 &= (1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-})^t,\\
\mathbf{v}_3 &= (1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-})^t,\\
\mathbf{v}_4 &= (1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-},1,\text{-},\text{-},\text{-})^t,\\
\mathbf{v}_5 &= (1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-},1,\text{-},\text{-},\text{-},\text{-},1,\text{-},\text{-})^t,\\
\mathbf{v}_6 &= (1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-},1,\text{-},\text{-},\text{-},\text{-},\text{-},1,\text{-})^t,\\
\mathbf{v}_7 &= (1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1,1,\text{-},\text{-},\text{-},\text{-},\text{-},\text{-},1)^t.
\end{aligned}
$$

Figure 2: A basis of the null space of $\mathbf{D}$. For readability, 0's are replaced with hyphens.

The next step is to check, for each w_i, $i = -8, -7, \ldots, -1$, each of the 2^7 solutions of the appropriate instantiation of (5) against the actual Hamming weights of the bytes of w_i (rather than their parities). This generates a set of 32-bit *candidate words* for w_i; as we shall see, the number of candidate words is much smaller than 2^7. By taking all combinations of candidate words for $w_{-8}, w_{-7}, \ldots, w_{-1}$ we generate a set of potential 256-bit initial keys. For each of these we execute the key expansion algorithm and compare the resulting Hamming weight values to the recorded measurements. We halt a key expansion and eliminate a potential initial key as soon as we find a discrepancy in the Hamming weights. Any potential initial key that matches all 528 measured Hamming weights is a solution.

4 Theoretical and Experimental Results

In this section we will compute the expected number of key expansions for the attack described in the last section, then present runtime data for an implementation of the attack.

For the expected number of key expansions, first compute the expected number of candidate words for a given w_i. Let X_i be the random variable giving the number of candidate words for w_i. Arguments of X_i are 32-dimensional column vectors representing w_i, so $X_i(\mathbf{y}) = |\{\mathbf{y}' \mid \mathbf{y}' \text{ is a candidate word for } \mathbf{y}\}|$. By definition, $\mathbf{y}'$ is a candidate word for $\mathbf{y}$ if and only if $\mathbf{y}$ and $\mathbf{y}'$ produce the same value in the linear system (5) (*i.e.*, $\mathbf{D}\,\mathbf{y} = \mathbf{D}\,\mathbf{y}'$) and the four bytes of $\mathbf{y}$ have the same Hamming weights as their counterparts in $\mathbf{y}'$. We wrote a C++ program to enumerate $|\{X_i = k\}|$ (where $\{X_i = k\} = \{\mathbf{y} : X_i(\mathbf{y}) = k\}$) for each k and then computed $\Pr\{X_i = k\} = |\{X_i = k\}|/2^{32}$. There are just 26 values of k for which $|\{X_i = k\}|$ is nonzero; they are given in the table in Figure 3 together with the values for $|\{X_i = k\}|$ and $\Pr\{X_i = k\}$. We summarize this information in the bar graph of Figure 4 with the values in the column labeled $\Pr\{X_i = k\}$ given as percentages and values for $k \geq 13$

aggregated. The expected number of candidate words is $E[X_i] = \sum_k k \cdot \Pr\{X_i = k\} \approx 4.407$. The variance is $E[X_i^2] - E[X_i]^2 \approx 9.208$, so the standard deviation is approximately 3.035.

k	$\lvert\{X_i = k\}\rvert$	$\Pr\{X_i = k\}$
1	584780016	0.136
2	774870656	0.180
3	675478272	0.157
4	607470080	0.141
5	250100480	0.0582
6	631576512	0.147
7	149599744	0.0348
8	169612928	0.0395
9	210228480	0.0489
10	72110080	0.0168
12	117933312	0.0275
13	6289920	0.00146
14	3386880	0.000789

k	$\lvert\{X_i = k\}\rvert$	$\Pr\{X_i = k\}$
15	9152640	0.00213
16	8592640	0.00200
18	15543360	0.00362
20	5653760	0.00132
21	37632	0.00000876
24	1881600	0.000438
28	448	0.000000104
30	470400	0.000110
35	62720	0.0000146
36	70560	0.0000164
40	62720	0.0000146
56	896	0.000000209
70	560	0.000000130

Figure 3: Frequencies of w_i's with k candidate words and the corresponding probabilities.

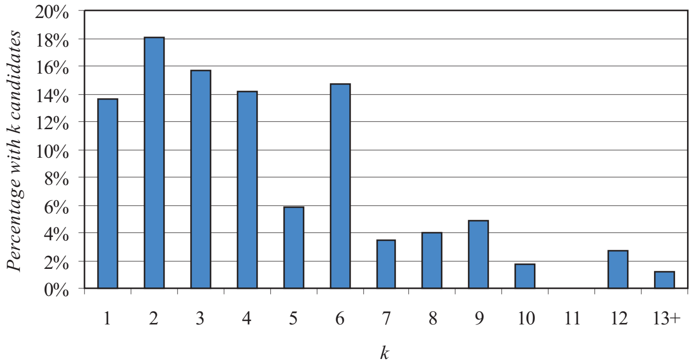

Figure 4: Bar graph of the data from the table of Figure 3.

The number of key expansions is $\prod_{-8 \le i \le -1} X_i$. The random variables X_i are independent so the expected number of key expansions is $E[X_i]^8 \approx 142253.43 \approx 2^{17.12}$. That is, the Hamming weights of the initial key bytes together with Hamming weight parities of the prekey determine, on average, all but about 17 bits of the 256-bit initial key.

We obtained experimental results by running multiple simulations on an Intel(R) Core(TM)2 CPU 6400 machine running at 2.13 GHz. The operating system was 64-bit Red Hat Enterprise Linux 4. We restricted our testing to 256-bit initial keys, since a viable attack on 256-bit inputs will be capable of computing a 128 or 192-bit key as well. As noted in the introduction, this study did not investigate the effects of power measurement error in the Hamming weights.

In our data a trial is the time required to execute the attack and retrieve the set of initial keys that generate a prekey matching all measured Hamming weight values. The results for 10^6 randomly generated initial keys were as follows.

- The attack found a unique and correct result in 100% of the trials.

- Total user time for all trials was 62 minutes 16.019 seconds (plus 1.621 seconds of system time), so the average time to find one initial key was 3.736 milliseconds.
- The average number of key expansions per attack was 142184.54, within 0.05% of the predicted number derived above.

In the (exceedingly rare) worst case, finding the initial key would require 70^8 key expansions and take our attack about half a year. However, the prekey generated by the modified LFSR can be derived from any 8-byte substring. A small change would enable our algorithm to find eight consecutive prekey bytes at a considerable time savings.

5 Conclusion and Future Work

The results here show that Hamming weight measurements from 8-bit smart card implementations of Serpent's key schedule reveal enough side channel information to uniquely determine a 256-bit initial key in a few milliseconds. More generally, we have shown that LFSRs may be very susceptible to SPA attacks and that algorithm design can greatly accelerate side channel attacks.

How does measurement error affect this type of attack? VanLaven *et al.* [16] presented an error-robust SPA attack on the AES key schedule. We suspect that the attack presented here can be made error-robust, as well, since we use only the first 200 rows of the reduced row echelon matrix $[\mathbf{A}''|\mathbf{e}'']$. The remaining 328 rows could be used for error correction. Also, pre-shift Hamming weights could be used.

The Rijndael and Serpent key schedule algorithms, we now know, are susceptible to SPA attacks. Are the key schedules of the other three AES finalists, Twofish, RC6, and Mars, also vulnerable?

A final question concerns the linear algebra of side channel attacks on LFSRs. We have seen that Hamming weight parities dramatically reduce search time for a Serpent initial key. Is this true in general for LFSRs? How much speedup should one expect?

References

[1] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the Advanced Encryption Standard. In *First Advanced Encryption Standard (AES) Candidate Conference*. National Institute of Standards and Technology (NIST), 1998.

[2] Eli Biham and Adi Shamir. Power analysis of the key scheduling of the AES candidates. In *Second Advanced Encryption Standard (AES) Candidate Conference*. National Institute of Standards and Technology (NIST), 1999.

[3] Geoffrey Keating. Performance analysis of AES candidates on the 6805 CPU core. In *Second Advanced Encryption Standard (AES) Candidate Conference*, pages 109–114. National Institute of Standards and Technology (NIST), 1999.

[4] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology (CRYPTO 1999)*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[5] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications.* Cambridge University Press, Cambridge, second edition, 1994.

[6] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Fifth International Conference on Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2003.

[7] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer, 2007.

[8] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In *Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2000.

[9] Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In *Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.

[10] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51(4):541–552, 2002.

[11] Siddika Berna Örs, Frank Gürkaynak, Elisabeth Oswald, and Bart Preneel. Power-analysis attack on an ASIC AES implementation. In *Information Technology: Coding and Computing (ITCC 2004)*, volume 2, pages 546–552. IEEE Computer Society, 2004.

[12] Thomas Popp, Stefan Mangard, and Elisabeth Oswald. Power analysis attacks and countermeasures. *IEEE Design and Test of Computers*, 24(6):535–543, 2007.

[13] Fumihiko Sano, Masanobu Koike, Shinichi Kawamura, and Masue Shiba. Performance evaluation of AES finalists on the high-end smart card. In *Third Advanced Encryption Standard (AES) Candidate Conference*, pages 82–93. National Institute of Standards and Technology, 2000.

[14] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Wiley, second edition, 1996.

[15] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, and Chris Hall. Performance comparison of the AES submissions. In *Second Advanced Encryption Standard (AES) Candidate Conference*, pages 15–34. National Institute of Standards and Technology (NIST), 1999.

[16] Joel VanLaven, Mark Brehob, and Kevin J. Compton. A computationally feasible SPA attack on AES via optimized search. In *Security and Privacy in the Age of Ubiquitous Computing: 20th International Information Security Conference (SEC 2005)*, pages 577–588, 2005.