A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter

Josh Jaffe

Cryptography Research, Inc.
575 Market Street, suite 2150, San Francisco, CA 94105, USA.
josh@cryptography.com

Abstract. Previous first-order differential power analysis (DPA) attacks have depended on knowledge of the target algorithm's input or output. This paper describes a first-order DPA attack against AES in counter mode, in which the initial counter and output values are all unknown.

Keywords: power analysis, SPA, DPA, HO-DPA, AES, counter mode.

1 Introduction

Previous first-order differential power analysis (DPA) attacks have depended on knowledge of the target algorithm's input or output [1][2]. This paper describes a first-order DPA attack against the Advanced Encryption Standard (AES) [3] in counter mode, in which the initial counter, input values, and output values are all unknown.

The attack proceeds as follows. Suppose the input data to an algorithm is unknown, but can be expressed as single secret constant summed with known, variable data. The known, variable part of the data is used to mount a DPA attack, and the secret constant is treated as part of the key to be recovered. The "key" recovered by the DPA attack is then a function of the actual key and the secret constant. The known input values are then combined with the recovered "key" to compute the actual intermediate values produced by the algorithm. The recovered intermediates are then used to carry the attack forward into later rounds, enabling additional DPA attacks to recover the real key.

The attack also addresses the challenges to DPA presented by block ciphers used in counter mode [4]. DPA attacks target secrets when they are mixed with known *variable* quantities. In counter mode only the low-order bits of the input change with each encryption. Hence there are few variable intermediates to target in the first round of a typical block cipher. We demonstrate a method for propagating the attack into later rounds in which more known, variable data is available.

Although counter mode presents additional challenges to DPA attacks, in certain respects it also makes the attack easier. Unlike most first-order DPA attacks, the sequential nature of the counter enables the attack to succeed with

only knowledge of the power measurements. Knowledge of input, output, and initial counter values are not required to implement the attack.

1.1 Related Work

Simple power analysis (SPA) attacks have been used to extract portions of keys directly from power traces without requiring knowledge of input messages. Fahn and Pearson used inferential power analysis (IPA), an attack that exploits binary SPA leaks [5]. Mayer-Sommer presented attacks exploiting SPA leaks in high-amplitude power variations [6]. Mangard presented an SPA attack against the AES key expansion step [7]. Messerges et al described SPA attacks on Hamming weight and transition count leaks [8].

Side channel collision attacks were introduced by Dobbertin, and have traditionally targeted SPA leaks using chosen ciphertext [9] [10] [11]. Side channel collision attacks can be adapted to the case in which inputs are known to be successive values of a counter.

High-order differential power analysis (HO-DPA) [12] attacks target a hypothesized key-dependent relationship between data parameters in a computation. Previous work has noted that HO-DPA attacks can be applied to situations in which cipher input values are not known [13].

Fouque and Valette presented the "doubling attack" [14] which exploits the relationship between inputs in successive RSA decryptions to recover the exponent. The attack succeeds despite the fact that the input to the modular exponentiation step is masked by a blinding factor. Messerges presented a second-order DPA attack [15] that defeated a data whitening scheme.

Chari et al [16] and Akkar et al [17] also presented DPA attacks on block ciphers with a "whitening" step.

2 Preliminaries

2.1 Notation

Suppose X and Y are used to denote input and output data of a transformation. (Letters other than X or Y will also be used.) If the transformation is implemented as a sequence of rounds, the input and output of the i^{th} round are denoted by X_i and Y_i.

Within a round, data may be partitioned into bytes for processing. $X_{i,j}$ and $Y_{i,j}$ denote the j^{th} bytes of round data X_i and V_i.

K is used to denote input keys, K_i denotes the i^{th} round key derived from K, and $K_{i,j}$ denotes the j^{th} byte of round key K_i.

Symbols

The symbol '$\oplus$' denotes the bitwise XOR of two n-bit vectors.
The symbol '$+$' denotes the ordinary addition of two numbers.
The symbol '$\circ$' denotes multiplication between two elements of $GF(2^8)$.
The symbol '||' denotes the concatenation of two vectors.

2.2 Description of AES

Although most readers are no doubt familiar with AES, this section gives a brief review of its design. The round transformations are grouped differently than in the AES standard to facilitate presentation of the attack, but the algorithm described here is equivalent to AES. The review will also familiarize the reader with the notation and concepts used in this paper.

AES is a block cipher that operates on 16-byte blocks of data. It is designed as a sequence of 10, 12, or 14 rounds, depending on whether the key K is 16, 24, or 32 bytes in length. The key is expanded by the AES key schedule into 16-byte round keys K_i.

The round structure of AES encryption. The following transformations are performed during each round of an AES encryption:

1. AddRoundKey
2. SubBytes
3. ShiftRows
4. MixColumns[1]

These operations are described below, using the following notation for intermediate round states:

X_i denotes the input to round i and the AddRoundKey transformation.
Y_i denotes the output of the AddRoundKey transformation and the input to the SubBytes transformation.
Z_i denotes the output of the SubBytes transformation and the input to the ShiftRows transformation.
U_i denotes the output of the ShiftRows transformation and the input to the MixColumns transformation.
V_i denotes the output of the MixColumns transformation and the input to the next round: $V_i = X_{i+1}$.

AddRoundKey Each byte of $Y_{i,j}$ is produced by computing the exclusive or (XOR) of a byte of incoming data $X_{i,j}$ with the corresponding byte of round key $K_{i,j}$:

$$Y_{i,j} = X_{i,j} \oplus K_{i,j} \ . \tag{1}$$

SubBytes Each byte of input data is transformed via an invertible non-linear 8-bit lookup table S:

$$Z_{i,j} = S[Y_{i,j}] = S[X_{i,j} \oplus K_{i,j}] \ . \tag{2}$$

[1] The MixColumns operation is not performed in the final round, and an additional AddRoundKey operation is performed after the final round.