Using SIMD Instructions to Accelerate AES with Provably Secure Higher-Order Masking

Abdulaziz Miyajan and Zhijie Jerry Shi
Computer Science and Engineering Department, University of Connecticut
Email: {abdulaziz.miyajan, zshi}@engr.uconn.edu

ABSTRACT
As a widely used block cipher, AES has been the target of many attacks, including side-channel attacks. Masking is a countermeasure to mitigate side-channel attacks by hiding the intermediate values used in cryptographic algorithms with random values. However, the masking scheme, especially high-order masking, has large overhead. In this paper we study efficient implementations of the higher-order masking algorithm for AES on processors with SIMD instructions. The SIMD instructions process multiple data items simultaneously, thus reducing the total execution time. The proposed implementation can defeat timing attacks, cache attacks, and power analysis attacks. On Intel processor with SSE3, our second order masking implementation is 6.8 times faster than previously reported results, and our third order masking is 5.6 times faster.

1. INTRODUCTION
Since the National Institute of Standards and Technologies (NIST) in 2001 selected Rijndael block cipher as the Advances Encryption standard (AES) [1] and replaced the old Data Encryption Standard (DES), AES has become a widely and extensively used encryption primitive. It has attracted a lot of attacks. The most successful attacks on AES are side-channel attacks, which utilize side channel information, such as timing and power consumption (e.g., [3] and [4]), to obtain secret keys.

Many countermeasures have been proposed to mitigate side-channel attacks. One common technique is masking, which tries to break the relationship between the leakage information and the value of sensitive data [5]. Secure masking schemes require splitting an intermediate value into a number of shares (called d-th order masking for d + 1 shares), which are then processed independently. To retrieve the original value, one has to combine all the shares. Attacks become more impractical as d increases, which makes higher-order masking a good countermeasure technique.

A provably secure higher order masking scheme was proposed for AES in [7]. The higher order masking can be easily applied to all linear operations in AES, except for the S-Box because it contains non-linear operations. Unfortunately, the higher order masking scheme spends most of the time on computing the non-linear operations. In [7], the authors used log/alog tables to implement multiplications over GF(2^8) and other look-up tables for squaring operations. In other methods such as [6] and [8], the authors generated the tables needed for computation before the S-Box operation, which is time-consuming. In recent work [2], Kim et al. performed the inverse operation over a composite field to reduce the cost of the S-Box. They used isomorphism functions to transform elements in GF(2^8) into a composite field, and then used pre-computed tables to perform the squaring and multiplications. The method reduced the execution time, but may be vulnerable to cache attacks because of table lookups.

It has been shown that SIMD instructions can greatly improve the performance of multimedia applications. As a result, SIMD instructions have been adopted in many popular processors, e.g., Intel and ARM processors. In this paper, we study the acceleration of AES software implementation with a provably secure higher order masking. We leverage SIMD instructions to achieve better performance, especially on the S-Box operation. Although software implementations are slower than dedicated hardware implementations, they have some advantages. For example, the costs can be lower as the existing hardware can be reused. It is also easier to update the software implementations to defeat new attacks.

We implemented AES with high order maskings on Intel processors with SSE3 support. Our implementation is more than five times faster than previous work with high order masking. Although we only focus on Intel processors in this paper, the techniques can be applied on any processors with SIMD support.

2. A FAST AND PROVABLY SECURE HIGHER-ORDER MASKING OF AES
In this section we describe our method to accelerate the provably secure higher order masking of AES [7].

AES places bytes in a block of plaintext into a matrix of 4 x 4, which is called a state, and has four major steps: SubBytes, ShiftRows, MixColumns, and AddRoundKey to update the state. The ShiftRows step does not change byte values. The basic operations in MixColumns and AddRoundKey are multiplication and addition in GF(2^8). The SubBytes is the most complicated step. In implementations that focus on performance, SubBytes is performed with ta-
ble lookups and can be combined with other steps. Without tables, SubBytes can be done with an inversion in $GF(2^8)$ followed by a few affine transformations. In the method described in [7], the inversion is performed with exponentiation because $x^{-1} = x^{254}$ in $GF(2^8)$. As the results, the most important operations in the secure implementation of AES are secure multiplication SecMult and secure addition.

To accelerate the implementation, we worked on both algorithm and implementation. On the algorithm side, we reduced the number of the SecMult operations. On the implementation side, we take the advantage of SIMD instructions to process multiple bytes simultaneously.

The higher order masking scheme of AES spends the most time on SecMult, which is used in the SubBytes and MixColumns steps. The secure version of multiplication and squaring relies on the regular multiplication of elements in $GF(2^8)$. Therefore, due to space limit, we focus on the implementation of multiplication in $GF(2^8)$ with SIMD instructions.

Algorithm 1 shows how to perform a multiplication in $GF(2^8)$. Basically, the algorithm scans all the bits in $b$ and for each non-zero bit, add $a$ to the product. $a$ is doubled and reduced (Lines 6, 7, and 8) in each iteration.

Algorithm 1: GMul

\begin{algorithm}
\begin{algorithmic}
  \State $p \leftarrow 0$
  \For {$i = 0$ to $7$} \Do
    \If {($b \& 0x1$)} \Then
      \State $p \leftarrow (p \oplus a)$
      \State $carry \leftarrow (a \& 0x80)$
      \State $a \leftarrow (a \ll 1)$
      \State $\text{if} (carry != 0)$ \Then
        \State $a \leftarrow (a \oplus 0x1B)$
        \State $b \leftarrow (b \gg 1)$
      \EndIf
    \EndIf
  \EndFor
\end{algorithmic}
\end{algorithm}

When Algorithm 1 is implemented with SIMD instructions in Intel SSE3, each instruction can process 16 bytes in parallel. All the constants, e.g., $0x1B$, need to be loaded in registers in advance. The $\text{paddb}$ (parallel add bytes) instruction is leveraged to perform the parallel shift left operation. We also shift the mask to left (Line 9), instead of shifting $b$ to right because shift right is expensive. The $\text{pcmpeqb}$ instruction compares bytes in parallel, and the results (either $0xFF$ or $0$) are used later as masks to perform the conditional operations (Lines 3 and 7).

3. SECURITY ANALYSIS AND IMPLEMENTATION RESULTS

The security of our implementations can be proven in similar ways as in [7]. The implementation can thwart timing attacks because it takes constant times for all input data. No branches depend on data or key information. The implementation does not use tables and the data locations do not depend on their values. Therefore, the implementation is secure against cache attacks. The high order masking schemes make the implementation secure against power analysis attacks.

We implemented AES-128 with the C language, using inline assembly code. We assume that a pool of random numbers that are independently and uniformly distributed is available as masks. This pool of masks is inaccessible to the adversaries.

We compared our implementations with recent schemes in [7] and [2]. The results are listed in Table 1. Our implementation does not use any tables and achieve a speedup of 6 or 5 times over [2] for second order and third order masking. The numbers of cycles do not include the time for generating random values, as in previous work [7] and [2].

<table>
<thead>
<tr>
<th>Method</th>
<th>Cycles($\times 10^3$cc)</th>
<th>Table size(Bytes)</th>
<th>Times</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original AES (Straightforward AES)</td>
<td>9.0</td>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First Order Masking</td>
<td>129</td>
<td>3153</td>
<td>9.56</td>
<td></td>
</tr>
<tr>
<td>Second Order Masking</td>
<td>2</td>
<td>256</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>13.5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Third Order Masking</td>
<td>2</td>
<td>3845</td>
<td>9.34</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>199.3</td>
<td>816</td>
<td>6.87</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>29</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. CONCLUSIONS

In this paper, we studied the fast implementation of AES with high order masking, using SIMD instructions. The implementation was done on Intel Processors with SSE3 and achieved a speed up of more than five times over the previous results with high order masking.

5. REFERENCES