High performance ASIC for Virtual Private Network

University of Maryland, Baltimore County (UMBC)

CMSC 611: Project Report, Fall 2008

Team Members:

1. Bhushan Sonawane (bhushan1@umbc.edu)
2. Namita Sapre (namita2@umbc.edu)
3. Vivek Relan (relan1@umbc.edu)
High performance ASIC for Virtual Private Network (VPN)

Abstract:

In today’s world, most of the communication is done using electronic media. Internet provides the global data communication system for it. Moreover, Internet allows users to connect to other computers and information stores easily, whenever they may be across the world. For example, an office worker away from his desk, perhaps on the other side of the world on a business trip or a holiday, can open a remote desktop session into his normal office PC using a secure Virtual Private Network (VPN) connection via the Internet. Data Security plays a vital role in such private communication in public networks. In modern information system, data security is achieved using cryptographic techniques. The widespread adoption of virtual private networks and secure IP (IPSEC) will further emphasize the significance of cryptographic processing among all types of communication. As demand for secure communication bandwidth continues to grow, efficient cryptographic processing will in the forefront of high throughput system design. In this project, we present high performance application specific chip for VPN. Performance enhancement is achieved by pipelined implementation of AES algorithm in counter mode and designing application specific instructions for RSA algorithm.

1. Overview

In last couple of decades, the world has changed a lot. Many businesses now think of global market and logistics. Most of the companies have facilities which are spread out around the world, and all of them looks for a secure and reliable communication wherever their offices are. Virtual Private Network (VPN) [1] is a solution for it.

VPN allows a private communication over the public network. It is achieved by encrypting a plaintext message at a sender site and decrypting the ciphertext at the receiver end. Here, two main aspects of data security are crucial: data confidentiality and data integrity. Using symmetric key algorithm like Advance Encryption Standard (AES), data confidentiality is achieved, while Rivest, Shamir, and Aldeman (RSA) algorithm is used for data integrity by digitally signing a message.

AES [2, 14] is a block cipher which takes 128 bits plaintext block and key size of either 128, 192 or 256 bits and produces a 128-bits cipher text block. The encryption process consists of at most 15 rounds and each round consists of four sub stages: sub bytes transformation, shift rows, mix columns, and add round key. Counter mode (CTR) [3] is widely used to convert block cipher into stream cipher to achieve non-linearity in the ciphertext.

RSA [4] is a public key encryption algorithm which consists of a pair of public key and private key. In VPN, RSA algorithm is used for digitally signing a message to provide data integrity. Sender's private key is used to sign a message, while receiver ensures the integrity of the message by decrypting a message using sender’s public key. The robustness of the algorithm lies in selecting a large key size of 1024, 2048 or 4096 bits wide. The most frequently used operation is modulo exponentiation [(M^E)%N]. It is necessary that such computer arithmetic
operations should be done rapidly on such a large numbers of size 1024, 2048 or 4096 bits wide.

The performance of the VPN application is mostly dependent upon the speed of the encryption and decryption process. Therefore, efficient implementation of AES and RSA algorithm are of prime focus in this project. By pipelined implementation of AES algorithm in CTR mode and designing application specific instructions for RSA algorithm, significant performance gain is obtained.

2. Characteristics of the application

As the performance of VPN application is mostly dependent upon the speed of the encryption and decryption process, the encryption algorithms used in VPN application are studied thoroughly. AES and RSA encryption algorithms are used in VPN.

Software implementation of AES algorithm used in OpenSSL [5] library is studied. After disassembling the AES encryption routine, number of cycles needed to encipher a single 128-bit plaintext block is calculated for the different key sizes. Table 1 summarizes the number of clock cycles needed for encrypting a single 128-bit plaintext block.

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Number of Rounds</th>
<th>Number cycles needed to encipher a single 128-bit plaintext block</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10</td>
<td>1000</td>
</tr>
<tr>
<td>192</td>
<td>12</td>
<td>1176</td>
</tr>
<tr>
<td>256</td>
<td>14</td>
<td>1352</td>
</tr>
</tbody>
</table>

Table 1 Number of clock cycles needed for AES encryption routine

As the AES algorithm is a block cipher, the plaintext is divided into blocks of 128-bits wide. The first plaintext block goes through the number of the rounds N0, N1..., N14 and produces the first ciphertext block and then second block goes the same number of rounds and produces the second cipher text block. This process continues, till the last plaintext block. This process can be visualized as shown in the figure 1.

Data:
R0 to R14 each takes 88 cycles
Fetch takes 24 cycles
Write back takes 8 cycles

Figure 1 Encryption procedure in AES algorithm
To achieve non-linearity in the ciphertext, counter mode (CTR) is most widely used. In counter mode, the series of rounds R0, R1, .., R14 applied on the single plaintext block are dependent upon each other, whereas the round 'N' of the current plaintext block is independent of round 'N' of the previous plaintext block.

In VPN, RSA algorithm is used to sign [6] a message for its integrity. The common operation in RSA algorithm is modulo exponentiation [7] as shown below.

\[ c = m^e \mod n \]
\[ m = c^d \mod n. \]

Here, m is the message, c is ciphertext, e is the encryption key, d is decryption key and n is modulus. As the robustness of the algorithm lies in the size of the m, e, d, and n, the typical size of them is either 1024, 2048 or 4096 bits wide. But, native arithmetic instructions work on 32-bits wide operands. For multi-precision multiplication operation [8], various methods like Comba multiplication method, Montgomery algorithm are used. 'C' implementation of Comba method is presented in Appendix A. In this method, the inner loop performs the unified multiplication and addition operation. Table 2 summarizes the profiling information of RSA encryption routine. It is obtained using gprof [9].

<table>
<thead>
<tr>
<th>% of Time</th>
<th>Number of calls</th>
<th>Name of function</th>
</tr>
</thead>
<tbody>
<tr>
<td>72.45</td>
<td>1300</td>
<td>combaMethod</td>
</tr>
<tr>
<td>27.52</td>
<td>1300</td>
<td>LargeNumberModulo</td>
</tr>
<tr>
<td>0.03</td>
<td>1</td>
<td>moduloExponentiation</td>
</tr>
</tbody>
</table>

Table 2 Profiling of modulo exponentiation routine for 1024 bit wide operand

From table 2, it can be inferred that multi-precision integer multiplication is most commonly used operation in encryption/decryption. As Comba method is used for multi-precision integer multiplication, optimizing it will improve the overall performance of encryption/decryption.

3. Design

3.1 AES

AES algorithm encompass at most 15 rounds. Each round consists of four stages: sub bytes transformation, shift row, mix column and add round key. In OpenSSL, efficient software implementation of AES is achieved by combining these four stages into single stage. This single step is nothing more than a byte substitution and XOR with round encryption key.

In AES, plaintext is divided into number of blocks and each block consist of 128 bits plaintext. Each 128 bit plaintext block goes through entire encryption process and produces ciphertext block. Afterward, second plaintext block goes through same process. As AES is block cipher,
encrypting the same plaintext under the same key always produces same output. Therefore to achieve nonlinearity in the ciphertext, several modes of operation have been invented. Counter Mode (CTR) is one of the most widely used mode of operation to convert AES as block cipher into stream cipher. Figure 2 shows AES encryption process using counter mode. Decryption processes in the counter mode (CTR) is symmetric to encryption process.

In counter mode, encryption/decryption of one block is independent of encryption/decryption of the other block. Inside encryption/decryption process, the first block goes through the number of the rounds N0, N1, ..., N14 and produces the first output block and then second block goes the same number of rounds and produces the second output block. This process continues, till the last plaintext block. This process can be visualized as shown in the figure 1 and 2.

Once we visualize the process of encryption/decryption as series of rounds [R0, R1, ..., R14] and each round as one functional unit, it can be inferred that independent blocks are not processed simultaneously. This is the main bottleneck in the entire encryption/decryption process. When functional unit R14 is active, then rest of the functional units [R0, R1, ..., R13] remains idle. Hence, round level parallelism is possible by issuing next block in the sequence to idle functional units. Figure 3 depicts the block level parallelism in encryption/decryption process of AES algorithm.

The encryption/decryption process is divided into 3 stages:

- Fetch: Fetch 128-bit plaintext block into registers. It takes 24 cycles.
- Execute: It is further divided into 15 sub stages R0, R1, ..., R14. R0 denotes the round 0 in the encryption/decryption operation. Each round takes 88 clock cycles.
- Write Back: In this stage, ciphertext block is written back to the main memory. It takes 8 cycles.

In our proposed pipelined AES, next message block is issued to functional unit when round functional unit is available. Therefore, message block (K+1) is in round (N-1) and message block K is in round (N) execute simultaneously. Therefore, first ciphertext block will available at
clock cycle 1352, second one will be available after 88th clock cycle and so on.

![Pipeline AES Algorithm in Counter Mode](image)

**Figure 3 Pipelined AES algorithm in Counter Mode**

### 3.2 RSA

In VPN, RSA algorithm is used to sign a message. Sender's private key is used to sign a message and receiver confirms the integrity of a message by decrypting a message using public key of the sender. The main operation in the RSA encryption/decryption process is the modulo exponentiation. Modulo exponentiation is performed efficiently using Montgomery reduction [10] procedure.

Native arithmetic instructions on 32-bit CPU work on 32-bit operands. But, in the RSA algorithm, modulo exponentiation is performed on at least 1024-bits wide operand. For multi-precision integer multiplication various methods like Comba method, Montgomery multiplication are used.

According to the principle of computer design, focus on common case in making decision trade-off, favor the frequent case over the infrequent case. The most common operation in multi-precision integer multiplication algorithms is the unified multiplication and addition operation. As the profiling result shows that comba method is time-consuming in the modulo exponentiation operation, improving the inner loop of it will improve the overall performance of the modulo exponentiation operation.

The code snippet of the inner loop of the comba method is obtained by disassembling the RSA code. The inner loop of comba method consumes 18 clock cycles on 32-bit general purpose CPU. Figure 4 shows the disassemble code for it.
MTHI 0                     # Initialize upper half of sum to zero
MTLO 0                     # Initialize lower half of sum to zero
Loop:   Lw     $t0, 0($t8)         # load A[j]
        LW     $t2, 0($t9)         # load B[i-j]
        ADDIU  $t8, $t8, 4         # Increment the counter j
        MFHI  $t3                 # load previous HI (Upper half of sum)
        MFLO  $t4                 # load previous LO (Lower half of sum)
        MUL    $t0, $t2            # (HI/LO) = A[j] * B[i-j]
        MFHI  $t5                 # load result of Mul (HI)
        MFLO  $t6                 # load result of Mul (LO)
        ADD    $t4, $t4, $t6       # ADD: LO
        ADD    $t3, $t3, $t5       # ADD: HI
        MTHI   $t3                 # Upper half of the sum
        MTLO   $t4                 # Lower half of the sum
        BNE    $t8, $t9, loop      # loop condition
        ADDIU  $t9, $t9, -4        #

Figure 4 Assembly code of inner loop of comba method on 32-bit CPU

By adding the unified multiply and add instruction in 32-bit general purpose CPU will improve
the inner loop of the comba method. Table 3 proposes the addition of the new application
specific instructions in general purpose processor instruction set to improve the overall
performance of the modulo exponentiation operation. Here, UH and LH are two special 32-bit
registers refers to upper half and lower half. MUL2ADD instruction is needed when two
operands of the multi-precision multiplication are same. The control logic for MULADD,
MUL2ADD and CADD instructions are presented in appendix B.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULADD R1, R2</td>
<td>(UH/LH) = (UH/LH) + R1*R2</td>
<td>Multiply and add</td>
</tr>
<tr>
<td>MUL2ADD R1, R2</td>
<td>(UH/LH) = (UH/LH) + 2<em>R1</em>R2</td>
<td>Multiply, double and add</td>
</tr>
<tr>
<td>CADD R1, R2</td>
<td>(UH/LH) = (UH/LH) + R1 + R2</td>
<td>Add to accumulator</td>
</tr>
</tbody>
</table>

Table 3 Addition of application specific instruction set to 32-bit processor

Figure 5 shows the assembly code of inner loop of comba method on 32-bit CPU having
enhanced instruction set architecture. The inner loop now takes 10 cycles.

MTHI 0                     # Initialise upper half of sum to zero
MTLO 0                     # Initialise lower half of sum to zero
Loop:   Lw     $t0, 0($t8)         # load A[j]
        LW     $t2, 0($t9)         # load B[i-j]
        ADDIU  $t8, $t8, 4         # Increment the counter j
        MULADD $t0, $t2            # (UH/LH) = (UH/LH) + t0*t2
        BNE    $t8, $t9, loop      # loop condition
        ADDIU  $t9, $t9, -4        #

Figure 5 Assembly code of inner loop of comba method on 32-bit CPU having enhanced ISA
4. Analysis

4.1 AES

Table 4 shows the number of clock cycles needed to encipher a different sizes of the plaintext block on general purpose CPU and on ASIC having proposed pipeline for AES algorithm. Assuming the clock frequency of both general purpose CPU and ASIC is same, speedup can be calculated as T(\text{old})/T(\text{new}).

where, T = Number of clock cycle needed for execution * Clock cycle time

\[
\text{Speedup} = \frac{\text{Number of cycles needed to encipher/decipher on CPU}}{\text{Number of cycles needed to encipher/decipher on ASIC}}
\]

<table>
<thead>
<tr>
<th>Number of Plain text blocks (1 block = 128-bits)</th>
<th>Number of cycles needed to encipher/decipher on CPU</th>
<th>Number of cycles needed to encipher/decipher on ASIC having pipelined AES</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1352</td>
<td>1352</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>13520</td>
<td>2144</td>
<td>6.31</td>
</tr>
<tr>
<td>20</td>
<td>27040</td>
<td>3024</td>
<td>8.94</td>
</tr>
<tr>
<td>30</td>
<td>40560</td>
<td>3904</td>
<td>10.39</td>
</tr>
<tr>
<td>40</td>
<td>54080</td>
<td>4784</td>
<td>11.3</td>
</tr>
<tr>
<td>50</td>
<td>67600</td>
<td>5664</td>
<td>11.94</td>
</tr>
<tr>
<td>60</td>
<td>81120</td>
<td>6544</td>
<td>12.4</td>
</tr>
<tr>
<td>70</td>
<td>94640</td>
<td>7424</td>
<td>12.75</td>
</tr>
<tr>
<td>80</td>
<td>108160</td>
<td>8304</td>
<td>13.03</td>
</tr>
<tr>
<td>90</td>
<td>121680</td>
<td>9184</td>
<td>13.25</td>
</tr>
<tr>
<td>100</td>
<td>135200</td>
<td>10064</td>
<td>13.43</td>
</tr>
<tr>
<td>120</td>
<td>162240</td>
<td>11824</td>
<td>13.72</td>
</tr>
<tr>
<td>140</td>
<td>189280</td>
<td>13584</td>
<td>13.93</td>
</tr>
<tr>
<td>160</td>
<td>216320</td>
<td>15344</td>
<td>14.1</td>
</tr>
<tr>
<td>180</td>
<td>243360</td>
<td>17104</td>
<td>14.23</td>
</tr>
<tr>
<td>200</td>
<td>270400</td>
<td>18864</td>
<td>14.33</td>
</tr>
<tr>
<td>500</td>
<td>676000</td>
<td>45264</td>
<td>14.93</td>
</tr>
<tr>
<td>1000</td>
<td>1352000</td>
<td>89264</td>
<td>15.15</td>
</tr>
</tbody>
</table>

Table 4 Analysis of ASIC having proposed pipeline for AES algorithm in counter mode

The graph 1 shows the speedup enhanced using pipelined AES in counter mode for various sizes of the plaintext block. It can be inferred that when number of plaintext blocks increases the overall speedup also increases. For 1000 blocks of the plaintext, speedup achieved is 15.15. It is observed that maximum speedup achieved using pipelined AES algorithm in counter mode is 15.36.
4.2 RSA

The profiling results shows that the multi-precision integer multiplication takes 72.45% of the time in executing 1024-bit modulo exponentiation on 32-bit general purpose CPU. The inner loop of the comba method takes 18 CPU clock cycles.

*SimpleScalar Instruction and Architecture Tool (SSIAT)* [11] is used to add the new instruction to SimpleScalar [12] simulation tool. All simulation results are obtained using sim-outorder utility present in SimpleScalar. By adding application specific instructions in 32-bit general purpose CPU, the inner loop of the comba method takes 10 CPU cycles.

Speedup enhanced in the inner loop of comba method = 18/10 = 1.8

According to Amdahl's law,

\[
\text{Net speedup} = \frac{1}{(1 - f) + \left(\frac{f}{\text{Speedup enhanced}}\right)}
\]

where \( f \) is fraction enhanced.

Here, fraction enhanced (\( f \)) = 0.7245 and speedup enhanced = 1.8

Therefore, net speedup = \[1/(0.2755 + (0.7245/1.8))\] = 1.475
5. Conclusion

High performance cryptographic processing is crucial for VPN application. In this project, ASIC having pipelined implementation of AES algorithm in counter mode is presented. The results shows that for large number of message block, the speedup enhanced over 32-bit CPU is 15.36. By designing application specific instructions for multi-precision integer multiplication, the overall speedup enhanced for RSA algorithm is 1.475 over 32-bit general purpose CPU. By designing application specific cache can further improves the overall speedup for AES and RSA algorithms.

References:

11. http://ce.et.tudelft.nl/~demid/SSIAT/
Appendix A

```c
struct big_number combaMethod(struct big_number A, struct big_number B)
{
    int i,j;
    unsigned long long sum;
    struct big_number result;

    unsigned int d = ((2*key_size)/(8*sizeof(unsigned long)));
    result.w = (unsigned long *) malloc(d);
    result.d = d;

    big_number_assign(&result, 0);

    sum = 0;
    unsigned long long pow_2_w = pow(2, 32);

    for(i=0; i< A.d; i++)
    {
        for(j=0; j<=i; j++)
        {
            sum = sum + (A.w[j]*B.w[i-j]);
        }
        result.w[i] = sum%pow_2_w;
        sum = sum/pow_2_w;
    }

    for(i=A.d; i< (2*(A.d)); i++)
    {
        for(j= (i - (A.d-1)); j <=(A.d-1); j++)
        {
            sum = sum + (A.w[j]*B.w[i-j]);
        }
        result.w[i] = sum%pow_2_w;
        sum = sum/pow_2_w;
    }

    return result;
}
```
Appendix B

\textbf{\texttt{\textbf{MULADD R1, R2}}}

\hspace{1cm}

\begin{center}
\begin{tikzpicture}
\node [draw] (r2) at (0,0) {R2};
\node [draw] (r1) at (2,0) {R1};
\node [draw] (temp) at (1,-1) {TEMP};
\node [draw] (uh) at (0,-2) {UH};
\node [draw] (lh) at (2,-2) {LH};
\node [draw] (carry) at (1,-3) {CARRY};
\node [draw] (multiply) at (1,0) {\times};

\draw [->] (r2) -- (multiply);
\draw [->] (r1) -- (multiply);
\draw [->] (multiply) -- (temp);
\draw [->] (temp) -- (uh);
\draw [->] (temp) -- (lh);
\draw [->] (uh) -- (carry);
\draw [->] (lh) -- (carry);
\end{tikzpicture}
\end{center}
MUL2ADD R1, R2

R2

R1

31 0 31 0

31

63

* 31

SHIFT LEFT BY 1 BIT

63 31 0

63

UH

LH

CARRY

+ +