Hardware and Software Implementations of RSA Encryption Using Fast Montgomery Modular Multiplication

Prof. Kris Gaj
George Mason University
Mike Koontz, Ryon Sumner

I. Introduction
With the heightened emphasis on security in realm of computers and computer networks, the RSA encryption algorithm has seen world-wide use as an effective method to encrypt and protect data. This key-based algorithm relies heavily on integer multiplication to perform the data encryption or decryption, with the speed of the multiplication algorithm contributing heavily to the throughput performance of the RSA encryption algorithm.

According to the paper by C. McIvor et al., the Montgomery multiplication algorithm is well suited to hardware implementations of RSA\(^1\) and was the algorithm of choice for implementing hardware RSA encryption. This project will further investigate the throughput performance of the RSA encryption algorithm using Montgomery modular multiplication in both hardware and software using VHDL and the C programming language.

II. Implemented Arithmetic Unit
The main unit of the project will provide an interface to load public/private keys and the data for performing RSA encryption or decryption:

---

Performing RSA encryption/decryption requires implementing the following algorithms:

\[ C = M^e \mod N \]
\[ M = C^d \mod N \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Cipher Code (Encrypted Result)</td>
</tr>
<tr>
<td>M</td>
<td>Decrypted Result</td>
</tr>
<tr>
<td>e</td>
<td>RSA Public Key</td>
</tr>
<tr>
<td>d</td>
<td>RSA Private Key</td>
</tr>
<tr>
<td>N</td>
<td>Product of two prime numbers used to generate RSA public and private keys</td>
</tr>
</tbody>
</table>

This will be implemented in hardware using Montgomery Modular multiplication as the basis for all multiplication required by the RSA algorithm, as shown in the below functional diagram:
The functional diagram implements the RSA algorithm as follows:

**Initialization:**

For encryption:

\[ S = e \]
\[ Y = M \]

For decryption:

\[ S = d \]
\[ Y = C \]

**RSA Algorithm:**

\[ S \rightarrow S' \quad // \text{convert } S \text{ into Montgomery domain} \]
\[ Y \rightarrow Y' \quad // \text{convert } Y \text{ into Montgomery domain} \]

for \( i = 0 \) to \( t - 1 \) \( // \text{where } t \text{ is number of bits in exponent} \)

\[ \begin{align*}
   &\text{if ( } E[i] == 1 \text{ )} \\
   &\quad Y' = MP(Y', S', N) \\
   &\quad S' = MP(S', S', N)
\end{align*} \]

\[ Y \leftarrow Y' \quad // \text{convert } Y \text{ back from Montgomery domain} \]

The resulting RSA algorithm after incorporating the 5-2 Montgomery Modular multiplication is:

\[ K = 2^{2k} \text{ (mod n)}; \]
\[ P1[0], P2[0] = 5to2\_MontMult(K, 0, C, 0, n); \]
\[ R1[0], R2[0] = 5to2\_MontMult(K, 0, 1, 0, n); \]

for \( i \) in 0 to \( d \) loop

\[ \begin{align*}
   &P1[i+1], P2[i+1] = \\
   &5to2\_MontMult(P1[i], P2[i], P1[i], P2[i], n); \\
   &\text{if } d[i] = 1 \text{ then} \\
   &R1[i+1], R2[i+1] = 5to2\_MontMult(R1[i], R2[i], P1[i], P2[i], n)
\end{align*} \]

end if;
end loop;

\[ M1, M2 = 5to2\_MontMult(1, 0, R1[k], R2[k], n); \]
\[ M = M1 + M2; \]

return \( M \);

In addition, the Montgomery multiplication will be performed using a 5-to-2 reduction Montgomery Multiplier, which is built using carry-save adders:
The 5-to-2 Montgomery Multiplier algorithm to be implemented is:

\[
\begin{align*}
S1[0] &= 0; \\
S2[0] &= 0; \\
\text{for } i \text{ in } 0 \text{ to } k-1 \text{ loop} & \quad q_i = (S1[i]_0 + S2[i]_0) + (A_i \cdot (B1_0 + B2_0)) \mod 2 \\
& \quad S1[i+1], S2[i+1] = \text{CSR} (S1[i] + S2[i] + A_i \cdot (B1 + B2) + q_i \cdot n) \div 2; \\
\text{end loop;}
\end{align*}
\]

In addition, the possibility of using Lucas chains to enhance the Montgomery Modular multiplication will also be explored.

A Lucas chain is an addition chain of the form

\[
a_0, a_1, a_2, a_3, \ldots
\]

*Where* \(a_0=1\) *and each successive entry in the chain can be created by adding 2 previous entries in the chain together. For example:*

\[
1, 1, 2, 3, 4, 6, 9, \ldots \text{etc}
\]

The use of Lucas chains to perform fast exponentiation will be investigated. By breaking down large exponents into a series of multiplications of smaller exponentiations, it will be possible to perform exponentiation on very large numbers efficiently.
III. Real-Life Application of Given Arithmetic Unit

The RSA algorithm is used throughout the word for a wide variety of security applications. This includes RSA SecurID Authentication, which is used to protect access to internal computer networks, VPNs, wireless networks and servers. RSA is also used to protect secure email and other network based transactions from unauthorized users.

A wide variety of devices incorporate the RSA algorithm to protect communications, such as cell phones, PDA (personal data assistants), ID card readers, RFID readers, as well as both laptop and desktop computers. These devices may all have different requirements for hardware RSA encryption area, latency and throughput. But overall, RSA encryption throughput will improve the performance of any of the previously listed devices, as the user observed speed is a critical factor in the performance of the device.

IV. Optimization Criteria

The RSA algorithm will be implemented with throughput as the main optimization goal. Circuit area, latency, and power will be considered as secondary goals. Optimization strategies will be used to increase throughput, which may include pipelining (if applicable), algorithmic parallelization, using the high-speed XtremeDSP slices built into the Virtex 4 architecture.

V. Interface

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>in</td>
<td>1 bit</td>
</tr>
<tr>
<td>reset</td>
<td>in</td>
<td>1 bit</td>
</tr>
<tr>
<td>key_available</td>
<td>in</td>
<td>1 bit</td>
</tr>
<tr>
<td>key_in</td>
<td>in</td>
<td>64-bits</td>
</tr>
<tr>
<td>key_read</td>
<td>out</td>
<td>1 bit</td>
</tr>
<tr>
<td>data_available</td>
<td>in</td>
<td>1 bit</td>
</tr>
<tr>
<td>data_in</td>
<td>in</td>
<td>64-bits</td>
</tr>
<tr>
<td>data_read</td>
<td>out</td>
<td>1 bit</td>
</tr>
<tr>
<td>full</td>
<td>in</td>
<td>1 bit</td>
</tr>
<tr>
<td>write</td>
<td>out</td>
<td>1 bit</td>
</tr>
<tr>
<td>data_out</td>
<td>out</td>
<td>64-bits</td>
</tr>
</tbody>
</table>

VI. Software Implementation and Test Vector Generation

The software implementation of the RSA encryption algorithm using Montgomery modular multiplication will be either freely available public domain software or a completely independent implementation of the RSA algorithm using Montgomery modular multiplication.

Test vectors will be generated using a predefined set of RSA public and private keys. These vectors will be chosen to simulate a real application of the RSA encryption algorithm, most notably the password authentication process for accessing a VPN network protected with RSA.

VII. Test Plan

The testing of the Montgomery modular multiplication RSA encryption algorithm will validate that the encryption process is working properly. In addition, the throughput of the RSA circuit

---

will be tested to ensure the highest rate achievable. Test vectors will be generated to test the overall RSA circuit and to trigger the worst-case critical path (or paths), which will affect overall circuit throughput.
VIII. Language, Platform and Tools

The hardware implementation of the RSA encryption algorithm with Montgomery modular multiplication will include:

- All hardware coding will be performed in VHDL
- Embedded platforms:
  - FPGA: VIRTEX 4-SX
  - ASIC: 90nm TCBN90G TSMC
- Hardware tools:
  - FPGA:
    - Aldec Active-HDL
    - Synplicity Synplify Pro
    - Xilinx ISE
  - ASIC:
    - ModelSim
    - Synposys Design Analyzer
    - Synopsys Prime Time
- All software coding will be written in C/C++ using Microsoft Visual Studio 6 or Microsoft Visual Studio .NET

IX. References


http://www.microlab.ece.ntua.gr/docs/nikos_paper1.pdf

http://ce.aut.ac.ir/~kmanochehri/Fast_montgomery.pdf