DSP APPLICATION IN E-COMMERCE SECURITY

Jiankun Hu [*], Ziping Xi, Andrew Jennings, H.Y. J. Lee and D. Wahyudi

∗Software & Networks Discipline, School of Electrical and Computer Systems Engineering, Royal Melbourne Institute of Technology (RMIT University), Melbourne 3000, Australia. Fax: +61 3 9925 5340; E-mail: Jiankun.Hu@rmit.edu.au

ABSTRACT

This is a case study on using DSP board to construct an encryption/decryption module embedded in a E-Commerce web server. The idea of using DSP is to push beyond the key length limits of encryption/decryption algorithms and computational power in software environment while avoiding the heavy investment in dedicated hardware encryptor/encryptor. The low cost, high computational power, high flexibility of DSP and the ubiquitous availability of PC PCI (peripheral component interconnect) slot for DSP can provide any web browser or web server an excellent cost-effective option to improve the security level of Internet applications. The paper provides a step-by-step procedure and reveals every detail of a successful implementation of DSP RSA encryptor/decryptor for a E-commerce web server by using the latest TMS320C6000 ^{TM} Evaluation Module (EVM) DSP hardware. A strong prime concept and Garner algorithm are introduced to generate more secure keys and compute encryption/decryption more efficiently than that of recent publications. Experiments show that the performance of using DSP hardware encryption can be 300 times faster than that in software environment. Building a DSP application embedded in a E-Commerce server needs to consider more issues than doing a pure open DSP application. However, since this conference falls exclusively in DSP field, other irrelevant DSP issues like RSA Key Database, server design etc. are not discussed here.

1. INTRODUCTION

With the growing number of electronic commerce applications, secure transmission of information is essential. Data that is sent across the network could be intercepted, read and modified by unauthorised persons. Therefore, a strong encryption system is required in order to protect the data at a maximum level. Unfortunately, a strong data encryption algorithm such as public-key cryptosystem usually leads to a degradation of the encryption performance because of the complexity of the algorithm. A direct consequence is the compromised key lengths being used in popular web browsers like Netscape or Internet Explorer and other E-commerce systems with limited computational resources. Strong public-key cryptographic techniques can not afford to process high volume of message. Instead, they are normally used to transmit secret keys and consequently this relative weaker secret key system is used to transmit real data. Hardware encryption is a preferred way to overcome the performance degradation problem. However, dedicated VLSI chip costs a fortune in investment and normal microcontroller is not suitable for handling sophisticated tasks. Digital Signal Processing (DSP) is an idea choice for such applications. It has very high computational power, high speed,

.

and high flexibility. Moreover, it is cost-effective due to batch production. Another amazing advantage is that nearly every PC has a PCI interface that can support DSP applications which reduces greatly any overhead or extra effort in constructing a functional DSP application.

RSA algorithm is the most popular public-key system created by Rivest, Shanir and Adelman [1]. There are some reports for software implementation of RSA algorithm [2][3]. However, report on DSP hardware implementation of RSA is very rare. In [4], a design and implementation of RSA cryptosystem using multiple TMS320E15 DSP chips is reported. The system developed consists of a stand-alone unit containing the DSP hardware and a high-level PC user-interface. The system allows for additional DSP chips to be inserted in allocated slots to improve its performance. The system was found to be 70 times faster than the same RSA algorithm implemented using C-language at PC level. However, no detailed report has been given, to the best of our knowledge, to address the DSP hardware implementation of RSA in PC PCI slot environment using latest commercial development tools, which should be a far more interesting issue as this PC PCI environment is available everywhere without any extra cost. This paper reports a successful DSP hardware implementation of RSA in a PC PCI board using the latest commercial TMS320C6201 fix-point DSP EVM and latest TI Code Composer Studio DSP software development tools. A strong prime concept is used in key generation which is more secure than conventional key generation as used in [4]. Experiments and testing show that the performance of using DSP hardware encryption can be as high as 300 times faster than that in software environment.

2. PROJECT IMPLEMENTATION

Details of the RSA algorithm are not listed here. Interested readers are referred to references [1][2][3][4][5] of this paper and references therein.

2.2.Hardware

The host is a common Pentium PC with 400MHz machine frequency. A standard PCI expansion slot on the computer's motherboard is available for plug-in DSP board. The TMS320C6x EVM Block Diagram is shown in Fig. 2.1 [6][7].The C6x EVM is built around the C6201 or C6701 DSP, which operates up to 1600 MIPS with a CPU clock rate of 200 MHz. It also provides a PCI Local BUS Revision 2.1-compliant interface that enables host access to the onboard JTAG controller, DSP host port interface (HPI), and board control/status registers.

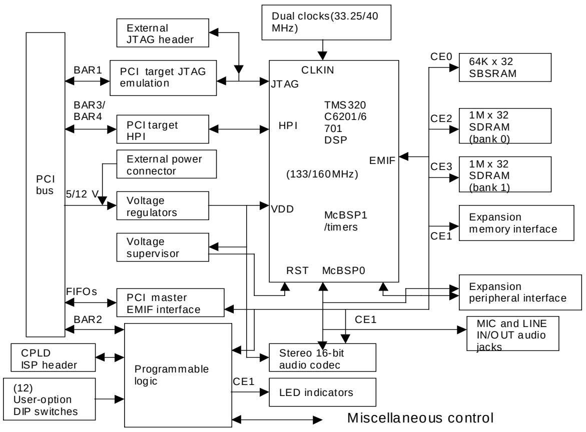


Fig. 2.1 TMS320C6x EVM Block Diagram [7].

2.2. Software

The C6x EVM software consists of host support software and DSP support software. The host support software supplied with the C6x EVM board includes the Win32 host utilities and libraries. The host utilities and host libraries run on an Intel ^{TM} compatible PC under either Windows 95 or Windows NT 4.0 [6]. The Code Composer Studio speeds and enhances the development process for programmers who creates and test real-time, embedded signal processing applications. It provides tools for configuring, building, debugging, tracing, and analysing programs.The RSA encryption can be summarised in Table 1.

TABLE 1RSA Encryption [5]

Public Key:	
n	product of two primes, *p* and *q* (*p* and *q* must remain secret)
e	relatively prime to (*p* – 1)(*q* – 1)
Private Key:	
d	e^{-1} mod ((*p* – 1)(*q* – 1))
Encrypting:	
$c = m^e \bmod n$	
Decrypting:	
$m = c^d \bmod n$	

For security reasons, prime numbers p, q should be large enough, say greater than 10^{100} [5]. Hence, selection of and processing of these two large prime numbers and their derivatives constitute major difficulties in DSP implementation. For prime number generation, we adopted the modified Euler's Theorem that has been used in [4]. It states that if a number p is a prime, then

$$x^{P-1} \bmod(P) = 1 \qquad (1),$$

for all integer values of x. A practical procedure is testing only 5 different values of *x* [4].

Discussion:

Prime number selection:

Obviously, the primary goal of equation 1 is to test whether a number is a prime or not. There is still plenty of room left on how to select these two prime numbers. The issue related to factoring problem is that a passive adversary tries to recover the plain text *m* from the corresponding cipher text *c*, given the public key (*n*, *e*) of the intended receiver *A*. The adversary would factor *n* first, and then compute ϕ and *d* just as *A* did to generate the key pair. Once *d* is obtained, the adversary can decrypt any cipher text intended for *A* [5].

Small encryption or decryption components (*e, d*) are used when the communication parties want to improve the algorithm efficiency for encryption or decryption. However, this would weaken the security of the algorithm since it is easier for eavesdroppers to search for the keys and/or plain text. It is recommended that the size of encryption or decryption components (*e, d*) should be roughly the same size as *n* [5].

In forward search attack, if the message space is small or predictable, adversary can decrypt a cipher text *c* by simply encrypting all possible plain text messages until cipher text *c* is obtained. To prevent this problem, the plain text message should be appended with fixed length random text (at least 64 bits) before message encryption.

In the key generation process, the prime numbers *p* and *q* should be selected so that factoring *n* = *pq* is computational infeasible [5]. The major restriction on *p* and *q* in order to avoid the elliptic curve factoring algorithm is that *p* and *q* should be about the same bit length, and sufficiently large. Another restriction on the primes *p* and *q* is that the difference $p - q$ should not be too small. If $p - q$ is small, then $p \approx q$ and hence $p \approx \sqrt{n}$. Thus, *n* could be factored efficiently by trial division by all odd integers close to $\sqrt{n}$. If *p* and *q* are chosen randomly, then $p - q$ will be appropriately large with overwhelming probability. In addition to these restrictions, many experts recommended that *p* and *q* be strong primes. A prime *p* is said to be *a strong prime* if the following three conditions are satisfied [5]:

(i) p – 1 has a large prime factor, denoted *r;*
(ii) p + 1 has a large prime factor; and
(iii) r – 1 has a large prime factor.

If the prime *p* is randomly chosen and is sufficiently large, the both *p –1* and *p + 1* can be expected to have large prime factors. The strong primes can protect against the *p –1* and *p + 1* factoring algorithm in any cases.

Improved encryption computation:

In order to improve the performance of the decryption engine, instead of direct computation of $m = c^d \bmod n$, we compute $m1 = c^{dp} \bmod p$ and $m2 = c^{dq} \bmod q$ (where $d_p = d \bmod (p\text{-}1)$ **and,** $d_q = d \bmod (q\text{-}1)$) then use Garner's algorithm to construct *m.* This procedure seems to be more complicated but more efficient since the moduli are smaller [5]. In addition, the small *decryption component* could be used to increase the speed of the algorithm. However, the use of this technique would weaken the algorithm. Therefore, the best choice for decryption component size is roughly same size of the public key *n*.

2.3. Outline of program coding and project building:

All the program source codes are in C. The encryption/decryption program works as follows:

i. The encryption starts by trying to open the message (source file) to be encrypted. If the source file could not be found, an error message is displayed and the program terminates. If the source file exists, the program continues to execute. At this point, a timing function is started to calculate the execution time of the encryption process. Then the contents of the source file are read character by character, each character is encrypted using the improved encryption technique discussed above with the public key and the result is written into another file (ciphertext file). When end of file of the source file has been reached, the encryption process completes and the timing function is stopped. Next, the encryption process time is calculated and displayed based on the “timestamp” produced by the timing function.

ii. The decryption program works similar to that of the encryption process. The program first looks for the ciphertext file, if the file exists then the program starts by reading the contents of the source file, character by character. If the ciphertext file does not exist then the program displays an error message and terminates. To calculate the decryption process, a timing function is performed before the contents of the file are read. Then the each ciphertext is decrypted by using the improved encryption technique proposed above with the private keys passed into the algorithm. The result of the computation is written into an output file, which is the recovered message. After the program finishes reading the ciphertext file, the decryption process ends and the timing function is stopped. The execution time of the decryption process is then calculated and displayed.

BUILDING THE APPLICATION PROJECT

After coding the application program in C, we need to make it working on DSP board. The procedure of creating a new project under Code Composer Studio is described below [6] [7] .

(1) Select Project → New Project from the menu and navigate to a desired directory where the project is going to be created. Save the project’s file name (eg. “enc.mak”) in the “File Name” field and click Save. A new project file is created with an empty project list.

(2) Add necessary files to the project list:

(2.1) Select Project → Add Files to Project. The project manager identifies files by their file extension:

- *.c : C source file.
- *.asm : Assembly source file.
- *.lib : Library file.
- *.cmd : Linker command file.

(2.2) Add the C source file to the project (eg. “enc.c”).

(2.3) Add the assembly source file to the project (eg. “vectors.asm”)

(2.4) Add the library file to the project (eg. “rts6201.lib”).

(2.5) Add the linker command file to the project (eg. “rsa.cmd”).

3. EXPERIMENTAL RESULTS

Experiments have been conducted to allow investigation into the performance of data encryption and decryption, in terms of elapse time and speed of execution in hardware and software environments, under the same platform of MS Windows NT operating system.

The experiments that performed in *hardware environment* were conducted using Code Composer Studio™ 1.2, which is a fully integrated development environment supporting Texas Instruments industry-leading TMS320C6201™ platform of DSP.

The experiments that performed in *software environment* were conducted using Microsoft Visual C++ 6.0 Enterprise Edition, Copyright © 1994-1998 Microsoft Corporation.

3.1. Building and running the program for experiment
In Code Composer Studio™ 1.2 [7], carry out the following procedures to execute RSA encryption and decryption program that has been implemented as discussed above.

(i) Choose Project → Rebuild All. Code Composer Studio recompiles, reassembles, and relinks all the files in the project. Messages about this process are shown in a frame at the bottom of the window.

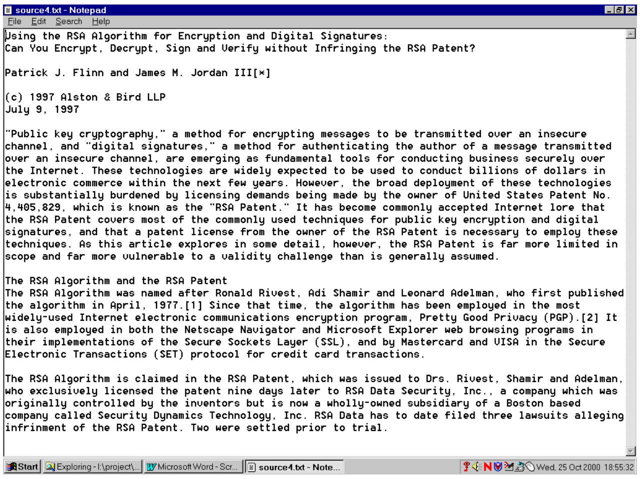
(ii) Choose File → Load Program. Select the program that has just been rebuilt, (example, encryption.out), and click Open. Code Composer Studio loads the program onto the target DSP and opens a dis-assembly window that shows the disassembled instructions that make up the program. (Notice that Code Composer Studio also automatically opens a tabbed area at the bottom of the window to show output the program sends to stdout.)

(iii) Click on an assembly instruction in the Dis-Assembly window. (Click on the actual instruction, not the address of the instruction or the fields passed to the instruction.) Press the F1 key. Code Composer Studio searches for help on that instruction. This is a good way to get help on an unfamiliar assembly instruction.

(iv) Choose Debug → Run.

3.4 Statistics data

Encryprion and decryption experiments have been conducted both in the software and hardware environment. Experimental data are shown in following figures.



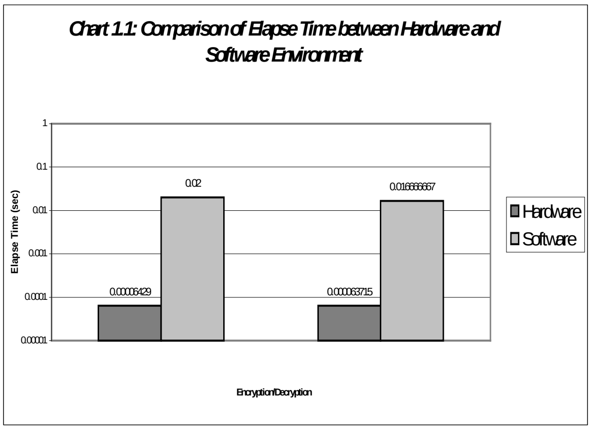
```
Using the RSA Algorithm for Encryption and Digital Signatures:
Can You Encrypt, Decrypt, Sign and Verify without Infringing the RSA Patent?

Patrick J. Flinn and James M. Jordan III[*]

(c) 1997 Alston & Bird LLP
July 9, 1997

"Public key cryptography," a method for encrypting messages to be transmitted over an insecure
channel, and "digital signatures," a method for authenticating the author of a message transmitted
over an insecure channel, are emerging as fundamental tools for conducting business securely over
the Internet. These technologies are widely expected to be used to conduct billions of dollars in
electronic commerce within the next few years. However, the broad deployment of these technologies
is substantially burdened by licensing demands being made by the owner of United States Patent No.
4,405,829, which is known as the "RSA Patent." It has become commonly accepted Internet lore that
the RSA Patent covers most of the commonly used techniques for public key encryption and digital
signatures, and that a patent license from the owner of the RSA Patent is necessary to employ these
techniques. As this article explores in some detail, however, the RSA Patent is far more limited in
scope and far more vulnerable to a validity challenge than is generally assumed.

The RSA Algorithm and the RSA Patent
The RSA Algorithm was named after Ronald Rivest, Adi Shamir and Leonard Adelman, who first published
the algorithm in April, 1977.[1] Since that time, the algorithm has been employed in the most
widely-used Internet electronic communications encryption program, Pretty Good Privacy (PGP).[2] It
is also employed in both the Netscape Navigator and Microsoft Explorer web browsing programs in
their implementations of the Secure Sockets Layer (SSL), and by Mastercard and VISA in the Secure
Electronic Transactions (SET) protocol for credit card transactions.

The RSA Algorithm is claimed in the RSA Patent, which was issued to Drs. Rivest, Shamir and Adelman,
who exclusively licensed the patent nine days later to RSA Data Security, Inc., a company which was
originally controlled by the inventors but is now a wholly-owned subsidiary of a Boston based
company called Security Dynamics Technology, Inc. RSA Data has to date filed three lawsuits alleging
infrinment of the RSA Patent. Two were settled prior to trial.
```

Fig. 3.1 Sample of plaintext file before encryption *source4.txt*

RESULT TABLE 1

		Elapse Time (second)	
	Result	Encryption	Decryption
DSP	1	0.00006429	0.000061485
Hardware environment	2	0.00006429	0.000065945
	AVERAGE	0.00006429	0.000063715
Speed (kbytes/sec)		36724.21839	147406.4192
MS Visual C++	1	0.02	0.02
Software environment	2	0.02	0.01
	3	0.02	0.02
	AVERAGE	0.02	0.016666667
Speed (kbytes/sec)		118.05	563.52

(a)



(b)

Fig.3.2 (a)-(b) Statistics data drawn from the encryption/decryption experiment

The original plaintext file source4.txt was encrypted. The speed of encryption in hardware environment was as high as 36,724.21839 Kbytes per second, which was approximately 311 times faster than software environment. The encrypted file cipher4.txt file was decrypted. The speed of decryption in hardware environment was as high as 147,406.4192 Kbytes per second, which was approximately 261 times faster than software environment.

4. CONCLUSION

The paper has reported a successful DSP hardware implementation of RSA in a PC PCI board using the latest commercial TMS320C6201 fix-point DSP EVM and latest TI Code Composer Studio DSP software development tools. A strong prime concept is used in key generation, which is more secure than conventional key generation that is used in [4]. An improved encryption/decryption implementation technique based on Garner's algorithm [5] is utilised to avoid the direct computation of $m = c^d \bmod(n)$. This procedure is more efficient than that was used in [4] as our procedure produces smaller moduli. Experiments of encryption/decryption have shown that the performance of using DSP hardware encryption can be as high as 300 times faster than in software environment. This better result than that of [4] is expected since PC PCI environment uses internal bus and contains fewer overheads. As all the resources used here are very popular available products, such implementation can be particularly helpful for IT engineers who are interested in this field.

5.REFERENCES

[1] R.L. Rivest, A. Shamir and L. Adelman, "On digital signatures and public key cryptosystems," Commun. ACM, Vol. 21, pp. 120-126, 1978.
[2] A. Selby and C. Mitchell, "Algorithms for software implementations of RSA," Proc. IEE, Vol.136, Pt. E, *pp.166-170, 1989.*
[3] J.J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," Electron. Letter, 18, pp. 905-907, 1982.
[4] M.H. Er, D.J. Wong, A/L Sethu and K.S. Ngeow, "Design and implementation of RSA cryptosystem using multiple DSP chips," IEEE International Symposium on Circuits and Systems, 1991, vol.1, Page(s): 49 -52
[5] B. Schneier, "Applied cryptography---Protocols, algorithms, and source code in C," John Wiley & Sons, 2nd Edition, USA.
[6] TMS320C6000 Code Composer Studio, Tutorial, Texas Instruments, *1999.*
[7]TMS320C6201/6701Evaluatiion Module, User's Guide, Texas Instruments, 1999.