Cryptography in Modern Communication Systems

Daniel V. Bailey[1], William Cammack[2], Jorge Guajardo[1], and Christof Paar[1]

[1] WPI – ECE Department, 100 Institute Road, Worcester, MA 01609 USA

{bailey,guajardo,christof}@ece.wpi.edu

[2] DSP R&D Center, Texas Instruments Inc., P.O. Box 655303, MS 8374, Dallas, TX 75265 USA

cammack@hc.ti.com

Presented at: TI DSPS FEST'99, August 3-6, 1999, Houston, TX, USA

Abstract

It is widely recognized that data security will play a central role in the design of future IT systems. Many of those IT applications will be realized as embedded systems which rely heavily on security mechanisms. Examples include security for wireless phones, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products and digital cinemas. Note that a large share of those embedded applications will be wireless, which makes the communication channel especially vulnerable.

An important part of almost all modern security protocols is public-key algorithms. Such algorithms are extremely computationally intensive and therefore, the arithmetic capabilities offered by DSPs are a very good match for them. This contribution introduces several important cryptographic concepts and their relevance to DSPs and embedded system applications. We give an overview of the previous work in the area of DSPs and cryptography. In addition, we present the design and features of the comprehensive cryptographic library SecuriTI. The library is currently under development and it was designed to provide an efficient algorithm agile cryptographic environment for the TMS320C54x and TMS320C6x DSPs. The library includes most public-key and symmetric-key algorithms of practical importance and it will be an important tool to support real-world applications and protocols.

1 Introduction

It is widely recognized that data security will play a central role in the design of future IT systems. Until a few years ago, the PC had been the major driver of the digital economy. Recently, however, there has been a shift towards IT applications realized as embedded systems. Many of those applications rely heavily on security mechanisms, including security for

wireless phones, faxes, wireless computing, pay-TV, and copy protections schemes for audio/video consumer products and digital cinemas. Note that a large share of those embedded applications will be wireless, which makes the communication channel especially vulnerable and the need for security even more obvious.

This merging of communications and computation functionality requires data processing in real time, and DSPs have shown to be good solutions for many applications. Digital signal processors have been designed and optimized to perform certain arithmetic operations at very high speeds. That is the reason why DSPs are an essential part of many communications devices that we, directly or indirectly, use and will use every day. Examples of such applications are cellular phones, faxes, pagers, and Internet solutions such as modems, multi-service network solutions that allow the implementation of IP telephony, Digital Subscriber Line (DSL) technologies, and some electronic commerce devices, to name just a few [12]. Since many of these applications will need security functionality in the future, this paper discusses cryptographic algorithms and their implementation on DSPs.

In addition to embedded devices, the explosive growth of digital communications also brings additional security challenges. Millions of electronic transactions are completed each day, and the rapid growth of eCommerce has made security a vital issue for many consumers. In the future, valuable business opportunities will be realized over the Internet and megabytes of sensitive data will be transferred and moved over insecure communication channels around the world. Thus, it is imperative for the success of modern businesses that all these transactions be realized in a secure manner. Specifically, unauthorized access to information must be prevented, privacy must be protected, and the authenticity of electronic documents must be established. Cryptography, or the art and science of keeping messages secure [19], allows us to solve these problems. We believe that cryptographic engines realized on DSPs are a promising option for protecting eCommerce systems.

Implementation of cryptographic systems presents several requirements and challenges. First, the performance of the algorithms is often crucial. One needs encryption algorithms to run at the transmission rates of the communication links. Slow running cryptographic algorithms translate into consumer dissatisfaction and inconvenience. On the other hand, fast running encryption might mean high product costs since traditionally, higher speeds were achieved through custom hardware devices.

In addition to performance requirements, guaranteeing security is a formidable challenge. An encryption algorithm running on a general-purpose computer has only limited physical security, as the secure storage of keys in memory is difficult on most operating systems. On the other hand, hardware encryption devices can be securely encapsulated to prevent attackers from tampering with the system [10]. Thus, custom hardware is the platform of choice for security protocol designers. Hardware solutions, however, come with the well-known drawback of reduced flexibility and potentially high costs. These drawbacks are especially prominent in security applications which are designed using new security protocol paradigms.

Many of the new security protocols decouple the choice of cryptographic algorithm from the design of the protocol. Users of the protocol negotiate on the choice of algorithm to use for a particular secure session. The new devices to support these applications, then, must not only support a single cryptographic algorithm and protocol, but also must be "algorithm

agile," that is able to select from a variety of algorithms. For example, IPSec (the security standard for the Internet) allows the following algorithms for data encryption: DES, 3DES, Blowfish, CAST, IDEA, RC4 and RC6. Thus, software-based systems would seem to be a better fit because of their flexibility. Therefore, the security engineer is faced with a difficult choice. Should he/she choose in favor of performance and security, and pay the price of inflexibility and higher costs? Or should he/she favor flexibility instead? Fortunately, DSPs combine the flexibility of software on general-purpose computers with the near-hardware speed and better physical security than general-purpose computers.

DSPs are already an integral part of many communications devices and their importance will continue to increase as we approach the next century. If we combine this with their flexibility to be programmed and their ability to perform arithmetic operations at very high speeds, it is easy to see that DSPs are a very promising platform to implement cryptographic algorithms. This holds in particular for the computationally intensive public-key schemes.

This paper focuses on the basics of cryptography and the implementation of cryptographic applications on DSPs. In Section 2, we introduce the general theory and concepts of Symmetric-Key and Public-Key cryptography as well as the operations which are most commonly performed. We will show that public-key operations are very computationally intensive and therefore require platforms such as the DSPs which have strong arithmetic capabilities.

In Section 3, a survey of previous cryptographic implementations on DSPs is presented, as well as some of the characteristics of the proposed algorithms.

Finally, Section 4 introduces the SecuriTI package. SecuriTI is a software package under development in our group. It is designed to address the needs of cryptographic systems in embedded applications by providing an efficient algorithm agile cryptographic library for the TMS320C54x and TMS320C6x DSPs. The library includes the most popular public-key and symmetric-key algorithms used everyday in real-world protocols and applications.

2 Cryptography: Public Key and Private Key

2.1 What We Can Do with Cryptography

Cryptography involves the study of mathematical techniques that allow the practitioner to achieve or provide the following objectives or services [15, 20]:

- *Confidentiality* is a service used to keep the content of information accessible to only those authorized to have it. This service includes both protection of all user data transmitted between two points over a period of time as well as protection of traffic flow from analysis.

- *Integrity* is a service that requires that computer system assets and transmitted information be capable of modification only by authorized users. Modification includes writing, changing, changing the status, deleting, creating, and the delaying or replaying of transmitted messages. It is important to point out that integrity relates to active attacks and therefore, it is concerned with detection rather than prevention.

Moreover, integrity can be provided with or without recovery, the first option being the more attractive alternative.

- *Authentication* is a service that is concerned with assuring that the origin of a message is correctly identified. That is, information delivered over a channel should be authenticated as to the origin, date of origin, data content, time sent, etc. For these reasons this service is subdivided into two major classes: entity authentication and data origin authentication. Notice that the second class of authentication implicitly provides data integrity.

- *Non-repudiation* is a service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions.

These security services are provided by using cryptographic algorithms. There are two major classes of algorithms in cryptography: Private-Key or Symmetric algorithms and Public-Key algorithms. The next two sections will describe them in detail.

2.2 Private-Key Algorithms

Private-Key or Symmetric algorithms are algorithms where the encryption and decryption key is the same, or where the decryption key can easily be calculated from the encryption key and vice versa. The main function of these algorithms, which are also called secret-key algorithms, is encryption of data, often at high speeds. Private-key algorithms require the sender and the receiver to agree on the key prior to the communication taking place. The security of private-key algorithms rests in the key; divulging the key means that anyone can encrypt and decrypt messages. Therefore, as long as the communication needs to remain secret, the key must remain secret.

There are two types of symmetric-key algorithms which are commonly distinguished: block ciphers and stream ciphers [19]. Block ciphers are encryption schemes in which the message is broken into strings (called blocks) of fixed length and encrypted one block at a time. Examples include the Data Encryption Standard (DES) [7], the International Encryption Standard (IDEA) [14], and many of the Advanced Encryption Standard (AES) [1] candidates such as CAST-256, MARS, RC6, Rijndael, Serpent, etc. Note that, due to its short block size and key length, DES expired as a US standard in 1998, and that the National Institute of Standards (NIST) is currently in the selection phase of the successor algorithm to be known as AES. The requirements for all the proposed AES candidates are a minimum block size of 128 bits and the ability to support keys of 128, 192 and 256 bits.

Stream ciphers operate on a single bit of plaintext at a time. In some sense, they are block ciphers having block length equal to one. They are useful because the encryption transformation can change for each symbol of the message being encrypted. In particular, they are useful in situations where transmission errors are highly probable because they do not have error propagation. In addition, they can be used when the data must be processed one symbol at a time because of lack of equipment memory or limited buffering.

It is important to point out that the trend in modern symmetric-key cipher design has been to optimize the algorithms for efficient software implementation in modern processors.

This is evident if one looks at the performance of the AES candidates on different platforms. Some algorithms have as atomic operations 32-bit and 64-bit integer multiplication. This hints at the fact that some of the AES candidates might be well suited to be implemented in devices such as DSPs which are able to do integer arithmetic quickly.

As a final remark, notice that one of the major issues with symmetric key systems is the need to find an efficient method to agree on and exchange the secret keys securely [15]. This is known as the key distribution problem. In 1977, Diffie and Hellman [9] proposed a new concept that would revolutionize cryptography as it was known at the time. This new concept was called public-key cryptography.

2.3 Public-Key Algorithms

Public-Key (PK) cryptography is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. Anyone that wants to send a message to party A can encrypt that message using A's *public key* but only A can decrypt the message using her *private key*. In implementing a public-key cryptosystem, it is understood that A's private key should be kept secret at all times. Furthermore, even though A's public key is publicly available to everyone, including A's adversaries, it is impossible for anyone, except A, to derive the private key (or at least to do so in any reasonable amount of time).

In general, one can divide practical public-key algorithms into three families:

- Algorithms based on the *integer factorization problem:* given a positive integer n, find its prime factorization. RSA [18], the most widely used public-key encryption algorithm, is based on the difficulty of solving this problem.

- Algorithms based on the *discrete logarithm problem:* given α and β find x such that $\beta = \alpha^x \bmod p$. The Diffie-Hellman key exchange protocol is based on this problem as well as many other protocols, including the Digital Signature Algorithm (DSA).

- Algorithms based on *Elliptic Curves.* Elliptic curve cryptosystems are the most recent family of practical public-key algorithms, but are rapidly gaining acceptance. Due to their reduced processing needs, elliptic curves are especially attractive for embedded applications.

Despite the differences between these mathematical problems, all three algorithm families have something in common: they all perform complex operations on very large numbers, typically 1024–2048 bits in length for the RSA and discrete logarithm systems or 160–256 bits in length for the elliptic curve systems. Since elliptic curves are somewhat less computationally intensive than the other two algorithm families, they seem especially attractive for embedded applications. For this reason, we believe that optimizing elliptic curve cryptosystems on DSPs is a promising approach.

The most common operation performed in public-key schemes is modular exponentiation, i.e., the operation $x^e \bmod n$. Performing such an exponentiation with, e.g., 1024 bit operands is extremely computationally intensive. Interestingly enough, modular exponentiation with long numbers requires arithmetic which is very similar to that performed in signal processing applications [5], namely integer multiplication. Thus, the computational

capabilities of DSPs are a very good match for public-key algorithms, and might make DSPs attractive for cryptographic engines beyond embedded applications. Section 3 contains a description of published work in this area.

Public-key (PK) cryptosystems solve in a very elegant way the key distribution problem of symmetric-key schemes. However, PK systems have a major disadvantage when compared to private-key schemes. As stated above, public-key algorithms are very arithmetic intensive and — if not properly implemented or if the underlying processor has a poor integer arithmetic performance — this can lead to a poor system performance. Even when properly implemented, all PK schemes proposed to date are several orders of magnitude slower than the best known private-key schemes. Hence, in practice, cryptographic systems are a mixture of symmetric-key and public-key cryptosystems. Usually, a public-key algorithm is chosen for key establishment and authentication through digital signatures, and then a symmetric-key algorithm is chosen to encrypt the communications and the data transfer, achieving in this way high throughput rates.

3 DSPs and Cryptography

The field of efficient algorithms for the implementation of cryptographic schemes is a very active one (for an overview on current techniques see [15, Chapter 14]). However, essentially all cryptographic research is being conducted independent of hardware platforms, and little research focuses on algorithm optimization for specific processors. In the following, we will review three previous implementations of public-key algorithms on DSPs. We will also summarize two of the fastest software implementations of PK schemes on general purpose computers. This will give the reader an idea as to the kind of speeds that are to be expected on general purpose machines, and which speeds can be expected in embedded system applications which are equipped with DSPs.

3.1 An RSA Implementation on the TI TMS320 DSP

The earliest reference to a cryptographic algorithm implementation on DSPs is [4]. In this contribution, the author proposes a new algorithm to perform modular multiplication which is the basic operation used to implement the RSA scheme. Barret explains how at the time the TI TMS320 DSP was a natural choice. The core operation of RSA is an exponentiation $x^e \bmod m$, where x, e, and m are in general multi-precision integers. Implementing RSA involves many integer multiplications and thus, it is particularly well suited for a dedicated hardware multiplier/accumulator (MAC) architecture. The modular multiplication operation was divided into two subroutines:

1. A modified schoolbook method for long number multiplication that takes advantage of a feature of the TI TMS320 which allows auto increment and decrement of data pointers during multiply and accumulate operations. This method requires $2n^2$ operations as opposed to the $6n^2$ operations when using the regular school book method for multiplication. Notice that, here n means the number of digits in the operands and

"operations" is a term used to include not only multiplications and additions but fetches and stores as well.

2. A modular reduction routine which came to be known as the Barret reduction method (for a good treatment of this technique see [15, Chapter 14]). This method performs the modulo m reduction operation while avoiding division.

This implementation of the RSA algorithm took on average (that is with an exponent composed of half zeros and half ones) 2.6 seconds to execute on the TI TMS32010 running at its maximum speed of 20 MHz, with a 512-bit modulus and exponent. Notice that the TMS302010 was the first general purpose DSP on the market

3.2 A Cryptographic Library for the Motorola DSP56000

In [11], the authors describe a cryptographic library designed for the Motorola DSP56000 that provided speeds comparable to hardware implementations of the same algorithms. The library includes modular arithmetic, an implementation of DES, message digest (hash) algorithms, and other methods. The paper points out that as cryptography becomes widespread, fast yet flexible cryptographic tools are important. Thus, in general, the right tool for cryptographic applications is not custom hardware but rather fast general-purpose processors such as the DSPs.

This contribution describes in detail the authors' implementation of the RSA algorithm. In particular, they focused on the integration of modular reduction and multi-precision multiplication according to Montgomery's method [16] (for a detailed analysis and comparison of Montgomery's multiplication algorithms see [5]). The authors derived their modified Montgomery multiplication algorithm by successive improvements which included interleaving multiplication and modular reduction and minimizing the number of shifts in favor of single precision multiplications. Right shifts were traded in favor of multiplications because right shift operations in the DSP56000 are slower than single precision multiplications. It is also important to point out that the extent to which this algorithm will be faster than the regular Montgomery reduction algorithm depends mostly on the relative speeds of multiplication and shifting. Thus, for example, in the Intel 80386 multiplication is an order of magnitude slower than right shift operations and therefore, the authors did not see more than a 10% improvement in the total execution time. However, in the DSP the improvement was manyfold.

The other major algorithm described in the paper was DES. In implementing DES, Dussé and Kaliski, followed the paper by Davio et al. [6]. The authors pointed out that one of the difficulties in implementing DES is the expense of applying the expansion function E and the permutation function P. However, one way of getting around this problem is by modifying the S-boxes to incorporate the permutation P and the mapping E.

The contribution ends by summarizing the performance of their cryptographic library and comparing it to equivalent software implementations. Their RSA implementation achieved a data rate of 11.6 kbits/s for the 512-bit exponentiation using the Chinese remainder theorem and 4.6 kbits/s without using it. These numbers refer to a full or private-key exponentiation, public-key exponentiation was faster. Notice that this implementation is more than an order

of magnitude faster than their software on a fast PC (20 MHz Intel 80386). The DES implementation run at 350 kbits/s in CBC mode which was comparable to the corresponding software implementation on a fast PC. Finally, their RSA-MD2 implementation achieved 190 kbits/s for large blocks of data (data blocks larger than 600 bytes) which was twice as fast as their software on a fast PC.

3.3 Fast Implementation of RSA, DSA, and ECDSA on the TI TMS320C6201 DSP

In [13], the authors propose two new methods for implementing public-key cryptography algorithms on a TI TMS320C6201 DSP. The authors suggest using the DSP as a cryptographic engine for server systems such as those found in electronic commerce applications. They also point out that DSPs present two major advantages. First, DSPs are designed with efficient hardware multipliers and can perform high-speed modular multiplications which are the basic operations in most of public-key cryptosystems. The second major advantage of DSPs is that they can be used as hardware engines for various algorithms since they are programmable.

The first method proposed by [13] is a modified implementation of the Montgomery modular multiplication algorithm. In particular, they improved on the Finely Integrated Operand Scanning (FIOS) algorithm proposed in [5]. Since the architecture of the TI TMS320C6201 DSP is designed for instruction level parallelism, they modify the algorithm to make it suitable for pipelining. The second approach relates to efficient methods of implementing elliptic curve cryptosystems. The contribution suggests a method for reducing the number of multiplications and additions used to compute $2^m P$ where P is a point on the elliptic curve and m is some integer. They accomplish this by reusing certain intermediate values in the computation of $2^m P$. They also point out that in architectures like the TI TMS320C6201 DSP, reducing additions is just as important as reducing multiplications since they take a similar number of clock cycles to perform (as opposed to the general case in which additions take a lot fewer clock cycles to perform than multiplications). It is also important to point out that their method reduces the number of multiplications as the value of the exponent m increases.

In their implementation, the authors use the TI TMS320C6201 DSP running at 200 MHz. They implemented RSA and DSA using the modified Montgomery algorithm combined with the k-ary method for exponentiation. Similarly, they implemented ECDSA based on their improved method for computing $2^m P$ combined with the sliding-window exponentiation technique and signed-binary encoding of the exponent. The total instruction code size was 41.1 kbytes which is important since the TMS320C6201 DSP has a total instruction code size of 64 kbytes. This means that this implementation can deal with RSA, DSA, and ECDSA without reloading the program for every public-key algorithm. Table 1 summarizes their results. Notice that 192-bit ECDSA provides slightly stronger security than 1024-bit RSA. Finally, the RSA timings for verification assume a 17-bit exponent and the RSA timings for signature generation assume the use of the Chinese Remainder Theorem.

Table 1: Comparison of ECDSA, DSA, and RSA signature operations. All times in ms.

Type of Operation	192-bit ECDSA $GF(p)$	1024-bit RSA	2048-bit RSA	512-bit DSA
signature	1.67	11.7	84.6	2.93
verification	6.28	1.2	4.5	5.14
general point multiplication	4.64	–	–	–

3.4 Fast Software Implementations

The purpose of this section is to summarize two software implementations of public-key algorithms. To our knowledge, these implementations are the fastest ones in the literature. Thus, we provide a point of comparison if implementing those algorithms on DSPs.

The primary focus of the paper [8], is to describe a fast software implementation of the elliptic curve version of DSA (known as ECDSA). In addition, the paper provides speed comparisons between different public-key schemes. These types of comparisons are scarce in the literature.

The authors implemented RSA, DSA, and ECDSA, for both $GF(2^n)$ and $GF(p)$, on a Pentium-Pro 200 MHz-based PC running Windows NT 4.0 and using MSVC 4.2 and maximal optimization. The code for RSA and DSA was written in C, using small macros in assembly language. The elliptic curve code was mainly written in C++ and for $GF(p)$ the same multi-precision routines in C were called as for RSA and DSA. RSA and DSA used a 1024-bit long modulus whereas elliptic curve operations used a 191-bit long modulus instead. Notice that elliptic curve cryptosystems with 191-bit arithmetic is slightly stronger than RSA with 1024-bit operations. Table 2 presents the timings for the signature operation using the four algorithms discussed.

Table 2: Comparison of ECDSA, DSA, and RSA signature operations [8]. All times in ms.

Type of Operation	ECDSA $GF(2^n)$	ECDSA $GF(p)$	RSA	DSA
signature	11.3	6.3	43.3	23.6
verification	60	26	0.65	28.3
general point multiplication	50	21.1	–	–

In [2], the authors introduced an entirely new type of finite field which can be used to optimize the arithmetic needed to implement elliptic curve cryptosystems. The authors called the new class of finite field Optimum Extension Fields or OEFs. This contribution is further improved in [3]. Notice that this paper shows the recent trend of optimizing algorithms for modern processors. Although, OEFs have been optimized for the type of microprocessors found in workstations and PCs, they seem relevant since modern DSPs, in particular the TI C6x series, seem capable of achieving close to the same speeds as these processors.

The authors implemented the algorithms on several platforms. Their focus was on the DEC Alpha 21064 and 21164A workstations. This implementation was written in optimized C, resorting to Alpha assembly to perform polynomial multiplications. In addition, the authors implemented the algorithms on a 233 MHz Intel Pentium/MMX PC using MSVS C++ 6.0. The PC implementation was entirely in C. They chose an OEF which carried arithmetic with a 183-bit modulus and applied it to an elliptic curve cryptosystem. Thus, with this implementation [3] achieved a security level roughly equivalent to that offered by 1024-bit RSA. Table 3 summarizes the timings to perform an elliptic curve point multiplica-

Table 3: Comparison of the timing required for an elliptic curve point multiplication [3]. All times in ms.

Type of Operation	Alpha 21064, 150 MHz	Alpha 21164A, 600 MHz	Pentium/MMX, 233 MHz
general point multiplication	7.0	1.09	13.1

tion which in an EC-based cryptosystem is equivalent to the encryption operation. Notice that one can achieve an ECDSA verification operation 20%-25% slower than a general point multiplication. The signature operation can be performed much faster than a general point multiplication [8].

4 An Implementation: The SecuriTI Library

It is well known that computationally intensive algorithms are a natural domain of DSPs, thus, we propose the implementation of a comprehensive cryptographic library for the TI TMS320C6201 and TMS320C54x processors which offers:

- Comprehensive high speed multi-precision arithmetic library for modular integer and Galois field computation. Operand lengths from 150 bits to 2048 bits will be supported.
- High-performance public-key algorithm library including RSA key agreement and key transport, RSA digital signatures, the NIST/ANSI Digital Signature Standard, Diffie-Hellman key exchange, and elliptic-curve based systems.
- Block cipher library including DES, IDEA, and possibly other algorithms currently under development in the cryptographic community such as the Advanced Encryption Standard (AES).
- Hash function library including MD5 and SHA-1.
- All algorithms will be highly speed optimized. The goal is to be considerably faster than PC or workstation based solutions and come close to modern ASIC speeds.

In order to attain these specifications, the library is partitioned into four logical levels, thus facilitating all stages of development and implementation while giving an unprecedented degree of control to the user. Design principles include encapsulation of functionality into four distinct levels. These levels are the subject of the next section.

4.1 Library Architecture

The library was partitioned into four different functional levels:

1. A high-level application interface (simple).
2. Cryptographic Algorithms.
3. Multi-precision Arithmetic Operations.
4. Basic Arithmetic Algorithms.

Level 4 and Level 3 routines provide the foundation for the rest of the library. Specifically, they provide the arithmetic routines required to support the cryptographic algorithms offered in Levels 2 and 1. For our purposes, the fundamental arithmetic operations are: addition/subtraction, multiplication, squaring, and inversion (division) for both, modular integer arithmetic and Galois fields $GF(2^k)$ arithmetic. The choice of best performing algorithm for each operation will depend on operand length, processor architecture, and available memory. We intend to offer a variety of fundamental algorithms to the users in Level 4. Examples of such algorithms include Montgomery multiplication, Montgomery squaring, Montgomery reduction, Barret reduction, school book method for multiplication, extended Euclidean algorithm, and multi-precision division. These basic algorithms will be given as a parameter choice in the Level 3 arithmetic functions. Level 3, thus, will provide the mathematical operators used in the construction of Level 2 public-key cryptographic algorithms as single operators and not as multiple algorithms.

Level 2 includes public-key algorithm implementations given the operators made available by Level 3. Examples of these algorithms include RSA Digital Signatures, RSA Encryption, DSA, Diffie-Hellman Key Exchange, and signature and encryption techniques based on elliptic curves. In particular, we will try to provide most of the basic functionality required to implement the algorithms which are included in the comprehensive emerging IEEE P1363 standard [17]. It is important to point out that all of the algorithms mentioned above depend on modular exponentiation as their basic building block. Elliptic curve cryptosystems can alternatively also be based on Galois field arithmetic.

Level 2 provides crypto operands based on symmetric-key algorithms or block ciphers, such as DES, IDEA, and the Advanced Encryption Standard which is currently under development by NIST as the next federal encryption standard. Level 2 will also include hash algorithm functionality. The goal is the implementation of the two important hash functions, MD5 and the Secure Hash Algorithm-1 [15]. As final remark, notice that the high level functions of Level 1 provide an easy interface to the crypto algorithms in Level 2.

4.2 Advantages of the SecuriTI Library

The modular design of the library allows for flexibility and ease of use. A user developing an application that makes use of functions that are offered by Level 3 does not need to be aware of the algorithms used to implement these operators. In fact, if the Level 4 arithmetic algorithms change, the user's application will still work correctly. This is important if further optimized arithmetic functions are added to the library. This structure also enables us to offer user interfaces tailored for different types of users.

Many users of cryptographic functions are not interested in managing such details as algorithm choice. The typical user would much rather call a simple function that handles encryption, for instance, rather than calling separate functions for parameter generation, key generation, and then encryption. For this reason, we offer a top level interface that hides as much complexity as possible. For those users requiring additional flexibility and who are willing to manage the additional complexity, Level 2 functions are a better choice. In addition, this system architecture allows us to enforce a simple rule: A function in level n may only call functions in level n or $n + 1$. This rule allows for modularization of code. As long as the lower levels support the specified operations, a program will always work correctly, regardless of code changes in the lower levels. In addition, this architecture provides us with an added level of flexibility, which may be desirable to some users. In most multi-precision arithmetic libraries, the user is not given the ability to choose which fundamental algorithms will be used to perform various operations. However, by their very nature, the performance of these algorithms depends heavily on the application.

Consider, for example, the operation of modular exponentiation. Methods exist which allow for a classical time-memory tradeoff, increasing throughput by precomputing results and storing them in memory. We list some exponentiation methods here in ascending order of performance, along with memory requirements.

1. Square and Multiply: Requires 1.5 n multiplications on average, requires no additional memory.

2. k-ary Methods: Requires storage of about 2^k long numbers. k is chosen based on n. In practice, a choice of $k = 4 \cdots 7$ is generally near optimal.

3. Addition Chains: Flexible memory requirements. These perform well in the case of an application needing to repeatedly raise varying bases to some fixed exponent.

4. Fixed-base Exponentiation: Flexible memory requirements which can be large (several 100 long numbers). These perform well in the case of an application needing to repeatedly raise some base to varying exponents.

Clearly, not all applications will offer the same amount of free memory, and some will have very little to offer. In addition, some applications may want to precompute values and store them for the life of the parameters, as in 3 and 4 above. The choice of the arithmetic algorithm, then, at the user's option, may be changed to perform better in his/her environment. Alternatively, if the user chooses not to specify an algorithm, the library will provide a safe default which will provide good performance in most situations.

5 Conclusions

We have introduced the basic concepts, characteristics, and goals of various cryptographic algorithms. We have shown how DSPs are essential parts of most communications systems and how this makes them especially attractive as a potential platform to implement cryptographic algorithms. Furthermore, previous implementations seem to indicate that DSPs are a platform very well suited for arithmetic-intensive cryptographic algorithms.

Following these ideas, we proposed the design of a comprehensive long number and cryptographic library for the TI TMS320C54x and TMS320C6x DSPs. We described the architecture of the library and the reasoning behind it. Finally, we showed how our design is unique and encompasses most features that an embedded systems engineer might desire when integrating security into a given application.

References

[1] Advanced encryption standard. At http://www.nist.gov/aes. The Advanced Encryption Standard (AES) ongoing development effort should finish sometime in the year 2001.

[2] Daniel V. Bailey and Christof Paar. Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. In *Crypto '98*, Berlin, 1998. Springer Lecture Notes in Computer Science.

[3] Daniel V. Bailey and Christof Paar. Inversion in Optimal Extension Fields. In A. Odlyzko, G. Walsh, and H. Williams, editors, *Conference on The Mathematics of Public Key Cryptography*, The Fields Institute for Research in the Mathematical Sciences, Toronto, Canada, June 1999.

[4] P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology – CRYPTO '86*, pages 311–323. Springer-Verlag, 1986.

[5] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *IEEE Micro*, pages 26–33, June 1996.

[6] M. Davio, Y. Desmedt, M. Fosséprez, R. Govaerts, J. Hulsbosch, P. Neutjens, P. Piret, J.-J. Quisquater, J. Vandewalle, and P. Wouters. Analytical characteristics of the DES. In D. Chaum, editor, *Advances in Cryptology – Crypto '83*, pages 171–202. Plenum Press, 1984.

[7] *Data encryption standard*. National Bureau of Standards, U.S. Department of Commerce, 1977.

[8] Erik DeWin, Serge Mister, Bart Preneel, and Michael Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. *Algorithmic Number Theory: Third International Symposium, Lecture Notes in Computer Science*, pages 252–266, 1998.

[9] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[10] Robert Doud. Hardware crypto solutions boost VPN. *Electronic Engineering Times*, Issue 1056:57–64, April 12th 1999.

[11] Stephen R. Dussé and Burton S. Kaliski Jr. A Cryptographic Library for the Motorola DSP56000. In *Eurocrypt '90 '*, Berlin, 1990. Springer Lecture Notes in Computer Science.

[12] Tom Engibous. The Communications Age – It's Real Time. Presentation given at the Dow Jones/Wall Street Journal Europe CEO Summit, June 22nd 1999. At http://www.ti.com/corp/docs/investor/speeches/wsj99/.

[13] Kouichi Itoh, Masahiko Takenaka, Naoya Torii, Syouji Temma, and Yasashi Kurihara. Fast Implemenation of Public-Key Cryptography on a DSP TMS320C6201. In *Proceedings of the First Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer-Verlag, 1999.

[14] X. Lai and Y. Massey. Markov Ciphers and Differential Cryptoanalysis. In *EUROCRYPT '91*, Berlin, 1991. Springer Lecture Notes in Computer Science.

[15] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1997.

[16] Peter L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[17] *IEEE P1363 Standard Specifications for Public Key Cryptography*, February 1999. Submitted to the ballot body by IEEE for voting.

[18] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[19] B. Schneier. *Applied Cryptography*. Wiley & Sons, 2nd edition, 1996.

[20] William Stallings. *Network and Internetwork Security – Principles and Practice*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1995.