A Cryptographic Library for the Motorola DSP56000

Stephen R. Dussé
Burton S. Kaliski Jr.
RSA Data Security Inc.
Redwood City, CA

Abstract. *We describe a cryptographic library for the Motorola DSP56000 that provides hardware speed yet software flexibility. The library includes modular arithmetic, DES, message digest and other methods. Of particular interest is an algorithm for modular multiplication that interleaves multiplication with Montgomery modular reduction to give a very fast implementation of RSA.*

Key words. *Data Encryption Standard (DES), Encryption hardware, Message digest, Modular arithmetic, Montgomery reduction, Motorola DSP56000, Multiple-precision arithmetic, RSA.*

1. Introduction

As cryptography becomes more widespread, fast yet flexible cryptographic tools are becoming important. Experience with hardware tools has shown that speed often cannot fully be realized unless all cryptographic methods of interest are implemented in hardware. For example, digital signatures are often implemented with a message digest followed by a public key encryption (as suggested first by [8]), so speeding up only the public key encryption may not be sufficient. Nevertheless, hardware implementations of many important yet nonstandard methods are hard to find.

We therefore propose that the right tool for many applications is not custom hardware but a fast general-purpose processor.

We have recently developed a cryptographic library for one such processor, the Motorola DSP56000 digital signal processor. The library includes the following methods:

Multiple-precision arithmetic. Several cryptosystems [12][16][18][19][26] involve integers hundreds of digits long, so this is a necessity.

Data Encryption Standard [7]. Though its security has been questioned [2], it remains an important tool.

Message digest. This operation is essential to almost every signature scheme. Flexibility is important as there is no widely accepted, secure, standard message

I.B. Damgard (Ed.): Advances in Cryptology - EUROCRYPT '90, LNCS 473, pp. 230-244, 1991.

digest; our choices include FIPS 113 MAC [6] and RSA-MD2, both of which were proposed for Internet electronic mail [22]. We are also considering RSA-MD4 [25].

In evaluating various general-purpose processors we found that the DSP56000 is especially well-suited because it can multiply two 24-bit integers and add the product to a 56-bit integer in 100 ns [14]. Such an operation is important not only in digital signal processing but also in multiple-precision arithmetic. The 24-bit word size also matches the 48-bit round keys of DES nicely. However, we expect that most of our results can be applied on other general-purpose processors.

This paper is organized as follows. We begin by describing our algorithms for RSA and DES. Then we present the design of a "crypto-accelerator card" for the IBM PC. Finally, we summarize the performance of the cryptographic library.

2. Related work

Work that motivated ours is Barrett's, Wiener's and Davio *et al*'s. Barrett observed the effectiveness of digital signal processors for cryptography and presented an implementation of RSA on Texas Instruments' TMS32010 [1]. Wiener developed a general software implementation of RSA on the DSP56000 that achieves 10.2K bits/s for 512-bit modular exponentiation with the Chinese remainder theorem [4]. An implementation specific to 512-bit moduli is even faster [30]. Davio *et al* made considerable progress in efficient techniques for DES [9], some of which we apply in our implementation.

Among other recent work on fast cryptography are Buell and Ward's implementation of multiple-precision arithmetic on a a Cray computer [5] and Laurichesse's fast implementations of RSA on conventional processors [21]. A number of fast hardware implementations can be found in Brickell's 1989 survey [4].

Currently the record for the fastest implementation of RSA is held by Shand, Bertin and Vuillemin of Digital Equipment Corporation's Paris Research Laboratories, who have achieved 226K bits/s for 508-bit modular exponentiation with the Chinese remainder theorem [29].

3. RSA algorithm

We now describe our implementation of RSA on the Motorola DSP56000. This section addresses the algorithms; performance is dealt with in Sec. 6.

In the RSA cryptosystem [26] one performs *modular exponentations*: computations of the form $C = M^E \bmod N$ where C, M, E and N are multiple-precision integers. This

computation is central to several other cryptosystems [12][16][18][19] so our results apply to those as well.

Speeding modular exponentiation has been of interest for some time, and there are a number of speedups [3][20][24][27]. We focus on one particular aspect, the integration of multiple-precision multiplication with modular reduction according to Montgomery's method [23]. Our speedup is complementary to others that focus on reducing the number of multiplications and reductions so ours and the others can be applied concurrently.

Our algorithm is most effective on a processor on which multiplication is fast relative to shifting, for then the convolution-sum approach described below outperforms the conventional shift-and-add method. We believe our algorithm will result in some speed improvement on every processor, but given that it is a little more complicated than conventional methods, the algorithm may not be justified on all processors.

3.1 Montgomery's method

We now outline Montgomery's method for modular arithmetic. Readers familiar with the topic may skip to Sec. 3.2.

In Montgomery's method we represent residue classes in an unusual way and redefine modular arithmetic within this representation. Specifically, let N be an integer (the modulus) and let R be an integer relatively prime to N. We represent the residue class A mod N as AR mod N and redefine modular multiplication as

$$\text{MONTGOMERY-PRODUCT}(A,B,N,R) = ABR^{-1} \bmod N$$

It is not hard to verify that Montgomery multiplication in the new representation is isomorphic to modular multiplication in the ordinary one:

$$\text{MONTGOMERY-PRODUCT}(AR \bmod N, BR \bmod N, N, R) = (AB)R \bmod N$$

We can similarly redefine modular exponentiation as repeated Montgomery multiplication. This "Montgomery exponentiation" can be computed with all the usual modular exponentiation speedups. To compute ordinary modular exponentiation $C = M^E \bmod N$, we compute $M' = MR \bmod N$ (ordinary modular reduction), $C' = (M')^E R^{1-E} \bmod N$ (Montgomery exponentiation), and $C = C'R^{-1} \bmod N$ (Montgomery reduction).

The practicality of Montgomery's method rests on the following nice theorem, which leads directly to an algorithm for Montgomery multiplication.

Theorem 1 (Montgomery, 1985)

Let N and R be relatively-prime integers, and let $N' = -N^{-1} \bmod R$. Then for all integers T, $(T+MN)/R$ is an integer satisfying

$$(T+MN)/R \equiv TR^{-1} \bmod N \tag{1}$$

where $M = TN' \bmod R$.

Proof Equation 1 is straightforward. The fact that $(T+MN)/R$ is an integer can be shown by substituting M. ■

If we choose the right R—say, a power of the base in which we represent multiple-precision integers—then division by R and reduction modulo R are trivial. With such an R Montgomery reduction is no more expensive than two multiple-precision products, and we can make it even easier.

3.2 Computing the Montgomery product

We now describe our algorithm for the Montgomery product. For the discussion we will let b be the base in which multiple-precision integers are represented. That is, we will represent an integer A as a sequence of digits $\langle a_0,\ldots,a_{n-1}\rangle$ where

$$A = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_1 b + a_0 \tag{2}$$

We will further require that all inputs to our algorithms can be represented in n base b digits, and that $R = b^n$. In Sec. 3.3 we determine limitations on the individual digits a_0, $\ldots$, a_{n-1}.

We derive our algorithm by successive improvements, beginning with the following algorithm taken directly from Theorem 1. (We note that our algorithm does not "normalize" its output to the range $[0,N-1]$. Sec. 3.3 shows why.)

MONTGOMERY-PRODUCT(A,B,N,R)
1 $N' \leftarrow -N^{-1} \bmod R$
2 $T \leftarrow AB$
3 $M \leftarrow TN' \bmod R$
4 $T \leftarrow T+MN$
5 **return** T/R

Improvement 1. Instead of computing all of M at once, let us compute one digit m_i at a time, add $m_i N r^i$ to T, and repeat. The resulting T may not be the same as in the original algorithm but the effect of adding multiples of N will be: namely, to make T a multiple of R. This is essentially the approach Montgomery gives for multiple-precision integers. We note that this change allows us to compute $n_0' = N^{-1} \bmod b$ instead of N'.

MONTGOMERY-PRODUCT′(A,B,N,R)
1 $n_0' \leftarrow -n_0^{-1} \bmod b$
2 $T \leftarrow AB$
3 **for** $i \leftarrow 0$ **to** $n-1$
4 **do** $m_i \leftarrow t_i n_0' \bmod b$
5 $T \leftarrow T + m_i N b^i$
6 **return** T/R

Improvement 2. Now let us interleave multiplication and reduction. We note that Montgomery reduction is intrinsically a right-to-left procedure. That is, m_i depends only on t_i. So we can begin adding this multiple to T as soon as we know t_i. This results in the following algorithm:

MONTGOMERY-PRODUCT″(A,B,N,R)
1 $n_0' \leftarrow -n_0^{-1} \bmod b$
2 $T \leftarrow 0$
3 **for** $i \leftarrow 0$ **to** $n-1$
4 **do** $T \leftarrow T + a_i B b^i$
5 $m_i \leftarrow t_i n_0' \bmod b$
6 $T \leftarrow T + m_i N b^i$
7 **return** T/R

Improvment 3. At this point we can begin to observe a potential difficulty for the DSP56000. The operation $T \leftarrow T + a_i B b^i$—the basic shift-and-add operation—is likely to break down into the following single-precision operations:

4.1 **do** $x \leftarrow t_i$
4.2 **for** $j \leftarrow 0$ **to** $n-1$
4.3 **do** $x \leftarrow x + a_i b_j$
4.4 $t_{i+j} \leftarrow x \bmod b$
4.5 $x \leftarrow x/b$ - right shift
4.6 $t_{i+n} \leftarrow x$ - (initial $t_{i+n} = 0$)

These operations involve not only n single-precision multiplications but also n right shifts. On many processors the "high part" and the "low part" of accumulators are separately addressable and the right shift can be accomplished with move instructions. This is true also on the DSP56000, but such shifting takes longer than a multiplication on the DSP56000, which motivates us to minimize the number of right shifts. Happily, there is a good way to avoid right shifts, and that is with the convolution-sum method of multiplication. In this method instead of performing operations like $T \leftarrow T + a_i B b^i$, we perform operations like $T \leftarrow T + (\Sigma_{0 \le i \le k} a_i b_{k-i}) b^k$. These involve $k+1$ multiplications but only one shift. The fact that Montgomery reduction is intrinsically right-to-left helps us again, and leads to our final algorithm.

MONTGOMERY-PRODUCT‴(A,B,N,R)
1 $n_0' \leftarrow -n_0^{-1} \bmod b$

2 $T \leftarrow 0$
3 **for** $k \leftarrow 0$ **to** $n-1$
4 **do** $T \leftarrow T + (\sum_{0 \le i \le k} a_i b_{k-i}) b^k + (\sum_{0 \le i \le k-1} m_i n_{k-i}) b^k$
5 $m_k \leftarrow t_k n_0' \bmod b$
6 $T \leftarrow T + m_k n_0 b^k$
7 **for** $k \leftarrow n$ **to** $2n-1$
8 **do** $T \leftarrow T + (\sum_{k-n+1 \le i \le n-1} a_i b_{k-i}) b^k + (\sum_{k-n+1 \le i \le n-1} m_i n_{k-i}) b^k$
9 **return** T/R

We expect that our final algorithm will generally be faster than the interleaved shift-and-add version on most processors, because our algorithm has fewer right shifts, $O(n)$ versus $O(n^2)$. It also has fewer stores, again $O(n)$ versus $O(n^2)$. (The number of other operations—fecthes, multiplies, and adds—is essentially the same for both algorithms.) However, we note that our algorithm has more complex loop control. We also note that our algorithm accumulates intermediate results that are a factor of $2n$ larger in magnitude than those in the shift-and-add algorithm, so we need a larger accumulator and addition instructions that can handle the larger accumulator.

On most processors we can implement the larger accumulator with multiple registers and the additions involving it with add-with-carry instructions. The DSP56000 is especially well suited since its accumulator is eight bits longer than the largest product its ALU can produce. Thus even without multiple registers or add-with-carry instructions the DSP56000 can handle the intermediate results for n up to 128.

The extent to which our algorithm is faster depends mostly on the relative speeds of multiplication and shifting. If multiplication is relatively slow then changes in the number of shifts will have an insignificant effect on total execution time. For example, on the Intel 80386 multiplication is an order of magnitude slower than shifting and we have observed what appears to be at best a 10 percent improvement in execution time. But on the DSP56000 the improvement is manyfold.

We conclude with a couple of remarks. First, we can derive a Montgomery squaring algorithm MONTGOMERY-SQUARE(A,N,R) in the usual way that is asymptotically 25 percent faster than the alternative MONTGOMERY-PRODUCT(A,A,N,R).

Second, we can precompute $n_0' = -n_0^{-1} \bmod b$ once during a Montgomery exponentiation since it depends only on the modulus N. Computing n_0' by a general modular inverse algorithm such as extended Euclidean GCD would not be all that expensive, since b is small. We have found instead (or rediscovered?) a very nice way to compute the modular inverse in the special case that n_0 is odd and b is a power of 2:

MODULAR-INVERSE(x,b)
1 - Computes $x^{-1} \bmod b$ for x odd and b a power of 2.
2 $y_1 \leftarrow 1$
3 **for** $i \leftarrow 2$ **to** $\lg b$
4 **do** **if** $xy_{i-1} < 2^{i-1} \bmod 2^i$

5 **then** $y_i \leftarrow y_{i-1}$
6 **else** $y_i \leftarrow y_{i-1} + 2^{i-1}$
7 **return** $y_{\lg b}$

The correctness of MODULAR-INVERSE can be verified by induction with the hypothesis $xy_i \equiv 1 \bmod 2^i$.

3.3 Representation of multiple-precision integers

We have not yet defined "base b representation" for the DSP56000, so we do so now. The DSP56000 has a signed multiply instruction that multiplies two 24-bit two's-complement integers and adds their product to a 56-bit accumulator. Thus the logical choices for "base b representation" are a sequence of 24-bit signed digits and a sequence of 23-bit unsigned digits. A sequence of 24-bit *unsigned* digits is rather awkward with a signed multiply instruction. Given that the 23-bit unsigned representation of an integer would generally be longer than the 24-bit signed representation, we chose the signed representation.

Thus the digits a_i in Eq. 2 satisfy $-2^{23} \le a_i \le 2^{23}-1$.

We now prove our claim that MONTGOMERY-PRODUCT need not adjust its result to the range $[0,N-1]$ by showing that the redundant range $[-N,N-1]$ can be maintained through all intermediate calculations.

Theorem 2

Let $R = b^n$ where $b, n > 0$, and let A, B and N be n-digit, multiple-precision integers in the signed representation, where $N > 0$. If A and B are in the range $[-N,N-1]$ then for all n-digit multiple-precision integers M, $(AB+MN)/R$ is in the range $[-N,N-1]$.

Proof We begin by proving two identities:

$$N + M < R$$
$$-N + M > -R$$

The first follows from the observation that the largest positive n-digit integer in the signed representation is less than $R/2$. The second follows from the fact that the largest positive n-digit integer and the smallest negative n-digit integer differ by less than R.

The theorem follows, since

$$(AB+MN)/R \le (N+M)N/R < N$$
$$(AB+MN)/R > (-N+M)N/R > -N$$

■

We note that a similar property holds in the unsigned representation, but it requires the further condition that $N < R/4$.

4. DES algorithm

Our implementation of DES follows the paper of Davio *et. al.* [9]. We first recall the definition of DES, then describe how we implemented the improvement.

By way of review, DES consists of 16 nonlinear *rounds* that transform a 64-bit block according to a 48-bit round key. The 16 round keys are computed from a 56-bit key according to a DES key schedule that we do not describe further. The bits in the 64-bit block are permuted in a fixed way before the first round and after the last. This is summarized in the following algorithm:

DES(M,K)
- Encrypts message M under key K with DES.
$\langle K_1,\ldots,K_{16}\rangle \leftarrow$ DES-KEY-SCHEDULE(K)
$\langle L,R\rangle \leftarrow IP(M)$
for $i \leftarrow 1$ **to** 16
do $\langle L,R\rangle \leftarrow$ DES-ROUND($\langle L,R\rangle,K_i$)
return $IP^{-1}(\langle R,L\rangle)$

DES-ROUND($\langle L,R\rangle,K$)
$\langle x_1,\ldots,x_8\rangle \leftarrow E(R) \oplus K$
for $i \leftarrow 1$ **to** 8
do $y_i \leftarrow S_i(x_i)$
return $\langle R, L \oplus P(\langle y_1,\ldots,y_8\rangle)\rangle$

Here E is a linear 32-to-48-bit mapping, P is a 32-bit permutation, and S_1, ..., S_8 are nonlinear six-to-four-bit mappings (the "S boxes"). IP is a 64-bit permutation and IP^{-1} is its inverse.

The primary difficulty with a direct implementation of DES is the expense of applying E and P. Davio *et al* observed that the linearity of E and P makes it possible to remove E and P entirely from the DES round. We can do this by modifying the S boxes to incorporate the permutation P that would follow in one round and the mapping E that would follow in the next. That is, we define S boxes S_1', ..., S_8' as

$$S_1'(x) = E(P(\langle S_1(x),0,\ldots,0\rangle))$$
$$S_2'(x) = E(P(\langle 0,S_2(x),0,\ldots,0\rangle))$$
$$\vdots$$
$$S_8'(x) = E(P(\langle 0,\ldots,0,S_8(x)\rangle))$$

We also change the main algorithm to apply E at the beginning and E^{-1} at the end. The mapping E^{-1} can be any 48-to-32-bit mapping be any that satisfies $E^{-1}(E(X)) = X$ for all X. This leads to our algorithm.

DES′(M,K)
- Encrypts message M under key K with DES.
$\langle K_1,\ldots,K_{16}\rangle \leftarrow$ DES-KEY-SCHEDULE(K)
$\langle L,R\rangle \leftarrow IP(M)$
$L' \leftarrow E(L)$
$R' \leftarrow E(R)$
for $i \leftarrow 1$ **to** 16
do $\langle L',R'\rangle \leftarrow$ DES-ROUND′($\langle L',R'\rangle$,K_i)
$L \leftarrow E^{-1}(L')$
$R \leftarrow E^{-1}(R')$
return $IP^{-1}(\langle R,L\rangle)$

DES-ROUND′($\langle R,L\rangle$,K)
$\langle x_1,\ldots,x_8\rangle \leftarrow R \oplus K$
for $i \leftarrow 1$ **to** 8
do $y_i \leftarrow S_i'(x_i)$
return $\langle R,L \oplus \langle y_1,\ldots,y_8\rangle\rangle$

Davio *et al* observed that this algorithm would be especially good on a processor with 48-bit words. We almost have this on the DSP56000, which can fetch two 24-bit words in one instruction. We note finally that speedups such as Davio *et al*'s have been adapted on many types of processor [10][17] and are quite common.

5. A "crypto-accelerator card" for the IBM PC

We now describe a working deployment of the DSP56000: our crypto-accelerator card. The card is a 3/4-length card occupying one expansion slot of an IBM PC, XT, AT or compatible. It uses the +5, +12, and -12 volt DC power supplies from the PC. The card is comprised of four major components: processor, optional DES chip, PC interface, and noise circuit. The total cost to us for the card is $400.

5.1 Processor

At the heart of the card is Motorola's XSP56001ZL20, a RAM-based member of the DSP56000. It is clocked at 20 MHz. The DSP consists of an arithmetic logic unit, address generation unit, and a program controller as well as internal program and data memory. There is extensive I/O support including a dedicated host interface port, an external memory/peripheral port, and two serial ports.

For our design, the DSP is attached to two banks of external 24-bit memory. The P program memory bank can hold 8K or 32K of RAM or ROM and the X data memory bank can hold 8K or 32K of RAM. An external power-up reset circuit holds the DSP in a reset state until the PC activates the DSP. This avoids the execution of spurious code upon power-up which could damage the DSP [15].

5.2 DES chip

The optional DES chip is Western Digital's WD20C03. It is clocked at 10 MHz. It can perform the ECB and CBC modes of DES [11]. The DES chip is attached to the DSP external memory/peripheral port and is mapped into the Y data memory bank's external I/O space. It transmits and receives data under program control of the DSP.

5.3 PC interface

The card communicates with the host PC via a multi-function interface. The interface has three main components: DSP external memory interface, direct DSP interface, and control and status.

DSP external memory interface. The DSP memory words are 24 bits wide and do not directly map to the PC's 8-bit memory space. For this reason, we designed special-purpose external memory support with these features:

- a *bank select* that specifies DSP external memory bank (P or X)
- an *address generation unit (AGU)* with a PC-loadable counter that indexes through the bytes of a 24-bit DSP word and then from word to word
- a *load mode* that indicates whether the address generation unit is to index DSP words in low-medium-high byte order or low byte only
- external memory *bus request/bus grant signals* programmable by the PC

To take advantage of fast block move instructions on the PC, the memory interface responds to any address in a large range in the PC's memory. Nevertheless, the AGU and not the particular PC memory address determines which DSP external memory byte is selected.

Direct DSP interface. The PC accesses the DSP56000's internal registers through the DSP's bidirectional host interface port. The port is mapped to eight locations in the PC's I/O address space. The port allows direct control of DSP software functions and gives access to software status and completion flags.

Control and status registers. Control and status is achieved through a series of latches and registers. These are mapped to eight locations in the PC's I/O address space. The registers support such functions as the loading of the AGU, DSP bus request/bus grant, location of the PC reserved memory, and DSP chip reset.

5.4 Noise circuit

The noise circuit consists of a noise diode, some amplification and an analog-to-digital converter. The noise diode is a CND6002A diode from Koep Precision Standards which produces a minimum noise output of 7 μv per root Hz over the range 10 Hz–100 kHz. Two op-amp stages amplify the noise and a precision comparator with TTL output converts the result into bits. The circuit is connected to the DSP and the status registers.

Since the primary purpose of the noise circuit is to generate random cryptographic keys, special care is taken to decouple the noise circuit from other computer components. Coupling would not only introduce unwanted noise but might also make the output of the noise circuit predictable.

6. Performance

We now summarize the performance of the cryptographic library.

Our RSA implementation on the DSP56000, as expected, achieves hardware speed: 11.6K bits/s for 512-bit exponentiation with the Chinese remainder theorem and 4.6K bits/s without. (This is for full or "private key" exponentiation, i.e., where the exponent is the same length as the modulus; "public key" exponentiation is faster.) The implementation is more than an order of magnitude faster than our software on a fast PC. We do not lose much performance in the overhead of interfacing the crypto-accelerator card to a PC.

The DES implementation runs at 350K bits/s in CBC mode for large blocks. ECB mode is a little faster. This is comparable to our software on a fast PC. In this case the interfacing overhead affects performance somewhat, such that it is faster for a fast PC to perform DES in software for blocks less than about 600 bytes than to interface to the crypto-accelerator card. A card with the optional DES chip runs at 3.8M bits/s and is faster than PC software for almost any block size.

Our RSA-MD2[1] implementation performs much like DES: for small blocks, fast PC software would be faster. The implementation achieves 190K bits/s, which is twice as fast as our software on a fast PC for large blocks.

---

[1]To be accurate, we have only implemented RSA-MD1, a proprietary message digest algorithm almost identical to RSA-MD2. We expect to include RSA-MD2 in later versions of the library.

We note that the lack of significant speedup in RSA-MD2 does not mean that our efforts to accelerate cryptography have failed. Although we have not made those algorithms "as fast as hardware," we have offloaded them from the PC, leaving the PC to perform such operations as I/O in parallel with the cryptography. We also note that there is room for improvement in all results. One factor significantly affecting performance is the limitation on internal memory. We do not store any of our program in internal memory, though we put some data there, so we expect a fair improvement just as a result of copying procedures to internal memory before executing them. There are other improvements as well.

Our results are summarized in Figs. 1–3 where we compare performances across the following hardware/software configurations:

- DSP56000 (the one stated in Sec. 5.1 with zero-wait-state memory): software alone, or with optional DES chip (the one stated in Sec. 5.1)

- fast PC (20 MHz Intel 80386): software alone, with crypto-accelerator card, or with crypto-accelerator card and optional DES chip

- slow PC (6 MHz Intel 80286): software alone, with crypto-accelerator card, or with crypto-accelerator card and optional DES chip

Our PC software is admittedly not the fastest in the world; a better baseline for comparison is probably Laurichesse's software [21].

7. Conclusion

We have described a flexible and fast cryptographic tool based on the Motorola DSP56000. Among the techniques we used are an algorithm for modular multiplication that interleaves multiplication with Montgomery modular reduction to give a very fast implementation of RSA, and the 48-bit model of DES due to Davio *et al.*

Since the time we began developing our cryptographic library, Motorola started producing a 27 MHz version of the DSP56000 [13]. Consequently our implementations can potentially be made 35 percent faster with no further investment on our part.

One of the areas of further interest is how fast RSA can be implemented on other digital signal processors. If we make the assumption that the speed of modular multiplication (Montgomery or otherwise) is proportional in speed to one FIR filter tap and to the square of the word size, we find that the processors likely to provide the best performance are AT&T's DSP16A (40M taps/s, 256 bits^2) and TI's TMS320C50 (29M taps/s, 256 bits^2) [28]. The DSP56000 measures in at 13M taps/s and 576 bits^2. We will probably explore the DSP16A and the TMS320C50 next.

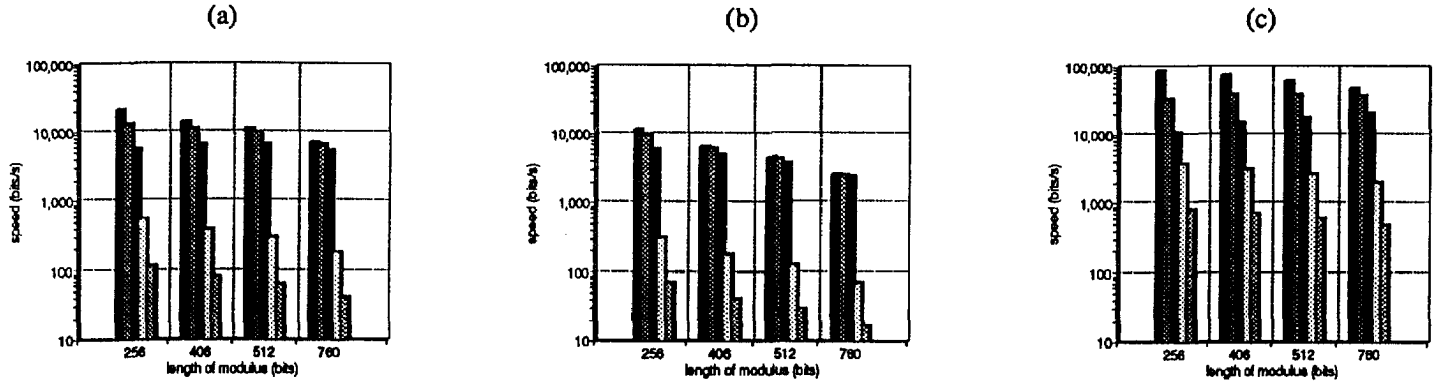

Figure 1 (a) Modular exponentiation speeds with the Chinese remainder theorem. (b) Modular exponentiation speeds without the Chinese remainder theorem. (c) Modular exponentation speeds for 15-bit exponents. From left to right, bars represent: DSP56000 alone; fast PC with DSP; slow PC with DSP; fast PC alone; and slow PC alone.

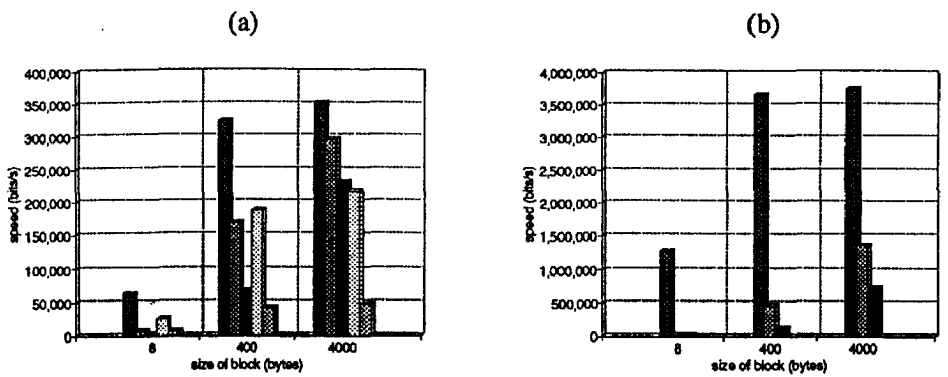

Figure 2 (a) DES speeds in CBC mode without DES chip. From left to right, bars represent: DSP56000 alone; fast PC with DSP; slow PC with DSP; fast PC alone; slow PC alone. (b) DES speeds with DES chip. Bars represent: DSP56000 with DES chip; fast PC with DSP and DES chip; and slow PC with DSP and DES chip.

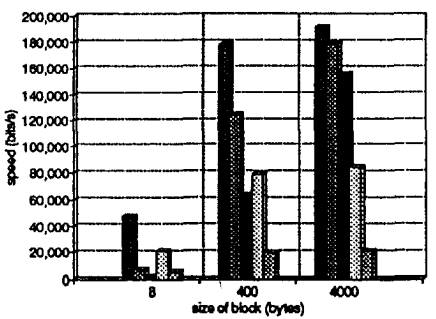

Figure 3 RSA-MD2 speeds. From left to right, bars represent: DSP56000 alone; fast PC with DSP; slow PC with DSP; fast PC alone; and slow PC alone.

Acknowledgements

We would like to thank our colleague Jeff Thompson for assisting in the implementation of the crypto-accelerator card and for preparing the timings described in Sec. 6. We also thank Jim Bidzos, Tom Knight, Ron Rivest, Ralph Sweitzer and Michael Wiener for their contributions.

References

[1] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A.M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86 Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer-Verlag, 1987.

[2] Eli Biham and Adi Shamir. Differential analysis of DES-like cryptosystems (preprint). *Proceedings of CRYPTO '90* (Santa Barbara, CA, August 12–15, 1990), to appear.

[3] Jurjen Bos and Matthijs Coster. Addition chain heuristics. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 400–407. Springer-Verlag, 1990.

[4] Ernest F. Brickell. A survey of hardware implementations of RSA. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 368–370. Springer-Verlag, 1990.

[5] Duncan A. Buell and Robert L. Ward. A multiprecise integer arithmetic package. *The Journal of Supercomputing*, 3:89–107, 1989.

[6] Computer data authentication. Federal Information Processing Standards Publication 113, National Bureau of Standards, U.S. Department of Commerce, 1985.

[7] Data encryption standard. Federal Information Processing Standards Publication 46-1, National Bureau of Standards, U.S. Department of Commerce, 1977.

[8] D.W. Davies and W.L. Price. The application of digital signatures based on public-key cryptosystems. In *Proceedings of the Fifth International Computer Communications Conference*, pages 525–530, 1980.

[9] M. Davio, Y. Desmedt, M. Fosseprez, 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: Proceedings of Crypto '83*, pages 171–202. Plenum Press, 1984.

[10] Marc Davio, Yvo Desmedt, Jo Goubert, Frank Hoornaert and Jean-Jacques Quisquater. Efficient hardware and software implementations for the DES. In G.R. Blakley and D. Chaum, editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 144–146. Springer-Verlag, 1985.

[11] DES modes of operation. Federal Information Processing Standards Publication 81, National Bureau of Standards, U.S. Department of Commerce, 1980.

[12] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[13] *Digital Signal Processors Quarter 3, 1989*. Motorola, 1989.

[14] *DSP56000/DSP56001 Digital Signal Processor User's Manual*. Motorola, 1990.

[15] *DSP56001 Advance Information*. Motorola, 1988.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.

[17] David C. Feldmeier and Philip R. Karn. UNIX password security - ten years later. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 44–63. Springer-Verlag, 1990.

[18] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A.M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86 Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[19] L.S. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C.G. Gunther, editor, *Advances in Cryptology - EUROCRYPT '88 Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer-Verlag, 1988.

[20] Donald E. Knuth. *Seminumerical algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1981.

[21] Denis Laurichesse. Mise en oeuvre optimisee du chiffre RSA. Rapport Laas No. 90052, Laboratoire d'Automatique et D'Analyse des Systemes, 1990.

[22] John Linn. Privacy enhancement for Internet electronic mail: Part III: Algorithms, modes, and identifiers. RFC 1115, Internet Activities Board Privacy Task Force, 1989.

[23] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[24] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithms for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[25] Ronald L. Rivest. The MD4 message digest algorithm (preprint). *Proceedings of CRYPTO '90* (Santa Barbara, CA, August 12–15, 1990), to appear.

[26] Ronald L. Rivest, Adi Shamir and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[27] A. Selby and C. Mitchell. Algorithms for software implementations of RSA. *IEE Proceedings*, 136 part E(3):166–170, 1989.

[28] Michael K. Stauffer and Michael Slater. General-purpose digital signal processors. *Microprocessor Report*, 3(10):25–29, 1989.

[29] M. Shand, P. Bertin and J. Vuillemin. Hardware speedups in long integer multiplication. *Proceedings of the Second ACM Symposium on Parallel Algorithms and Architectures* (Crete, July 2–6, 1990), to appear.

[30] Michael Wiener. Personal communication, 1990.