MICROPROCESSORS AND DATA ENCRYPTION

by

G. I. Davida National Science Foundation, Washington, D.C.

D. L. Wells System Development Corporation, McLean, VA

Abstract

The use of microprocessors for implementing some of the most recent cryptographic algorithms is examined. Time and space requirements for the various encryption programs are studied and applications where microprocessor based encryption would be able to meet throughput requirements are identified.

It is concluded that with currently available microprocessors, micro-based encryption is useful for offline encryption/decryption, electronic mail systems, and most terminal applications.

Keywords: Encryption, microprocessors, data security, Data Encryption Standard, public key, digital signature, encryption, electronic mail.

Introduction

The use of cryptography for the protection of information outside of the traditional military and diplomatic spheres has recently received a great deal of attention in both the technical and popular presses. [Diffie, 76], [Washington Post, 79], [Science, 78], [Electronic News, 79]. Encryption schemes such as the Data Encryption Standard (DES) [NBS, 77-1]; the public key cryptosystems of Rivest, Shamir and Adelman (RSA), [Rivest, 78], and of Merkle and Hellman [Merkle, 78], and the database encryption technique of Davida, Wells and Kam (DWK) [Davida, 78] are powerful new additions to this once closed field. It is not our intent to debate the security provided by any of them. Rather, we will examine the ease with which these schemes can be implemented using currently available microprocessors.

To do this, we describe the fundamentals of three representative encryption systems, the specific use of which they were designed, and any area for which they are clearly not applicable by design. For example, a database encryption scheme will not be generally usable for enciphering messages for transmission since that is not the purpose for which it was designed.

Applications of Encryption

Encryption is frequently thought of as being useful only for protecting messages which are being transmitted from one point to another and may be intercepted and read along the way. While this is likely to remain the major use of encryption for some time, many new applications for encryption are appearing.

In addition to guarding the contents of messages in transmission, encryption can be used to allow the recipient of a message to verify its authenticity. A system which allows sequencing or time stamping of messages will also allow the recipient of a message to determine that a message is current and not an old message which has been recorded and subsequently played back. A new use of encryption for message transmission is the concept of "electronic mail". Here, a person may send a message to any person who is a member of the mail system. If it is possible to send a private message to a person with whom one has had no previous contact, the system is said to have the "public key property". In a public key system, anyone can send an encrypted message to anyone else, but only the valid recipient has the ability to decrypt it. A system in which it is possible to "sign" a message in some unforgeable way is said to have the "digital signature property". Both are desirable for electronic mail systems since they allow private, unforgeable communications where the validity and origin of any message (for example a contract) can be determined by an impartial third party (a judge).

Encryption can also be used by a communications system to encipher all messages sent by the system. In this case, a stream cipher can be very useful since it is not important that the system be able to differentiate between two enciphered messages; the two messages can be separated after being decrypted by the system. For this application, speed, and security in the form of a long key are the primary considerations.

Data stored internal to a computer can also be protected cryptographically. Generally, this resolves into two areas: file encryption and database encryption. The difference between these is that a database must be able to be accessed randomly, while files are generally referenced sequentially.

Thus, a stream cipher which requires the decryption of (or at least secondary key generation for) all records preceding the desired record is inappropriate for database encryption. In addition, any scheme used on stored data must have sufficient speed to prevent significant disk access delays if mass transfers are likely to be made.

In any encryption scheme, the issue of key management must be considered. The use of micros rather than specialized hardware poses potential problems in this area.

As can be seen, each of these applications requires specific properties from the design and implementation of a cryptographic system. Often two applications will require conflicting properties of a cipher system.

454
Thus, it is unlikely that one encryption system will be able to be used for all applications, and the fact that a system is not suitable for one use does not mean it will not be good for others.

Considerations for Microprocessor Implementation

To implement any of these cryptographic systems using microprocessors requires a consideration of where the system will be used as well as the computational requirements of the particular system.

Microprocessors, because of their limited instruction set and short word length, pose a number of difficulties for the implementation of some of the mathematically based schemes. The time and space requirements of the software packages needed to implement these algorithms of these packages must be examined. Alternative ways of performing the same transformations (i.e., table driven encryption) should be considered.

The Data Encryption Standard poses a different problem with respect to microprocessor implementation. Since DES has been adopted (by the Department of Commerce) as the Federal Information Processing Standard, it is subject to certain special requirements. The primary of these is that only hardware implementations of the DES will be certified by NBS [NBS, 77-2]. This does not mean that a micro based implementation will not perform as well as a hardware version, but there will be some applications where the micro based version will not be able to be used.

The Microprocessor and the Implementation Philosophy

As an illustrative tool, we will assume a 16-bit microprocessor with all of the commonly available instructions. The cycle time assumed is 1.2\(\mu\text{sec}\), giving a time of 2.4\(\mu\text{sec}\) for ordinary memory reference instructions and 3.6\(\mu\text{sec}\) for instructions using indirect or indexed memory access.

Since maximum throughput is desired, straight line code is used whenever feasible and useful. Looping is only considered if the overhead is very small compared to the space saved.

For each encryption technique, the general outline of an encryption program is given, and the total number of each type of frequently executed instruction execution is summarized. Storage requirements are also summarized. From these, throughput rates and memory requirements can be determined.

Implementation of DES

The Data Encryption Standard (DES) is a substitution/permutation block cipher which encrypts 64-bit blocks using a 64-bit key (including 8 parity bits, since the key is specified by 8 ASCII characters). Decryption is done exactly the same as encryption, with the 64-bit key being applied to a 64-bit deciphered block to obtain the original text. This is the Electronic Code Book mode of DES. The Cipher Feedback and Block Chaining mode will have lower throughputs.

The operation of DES is an initial permutation of the 64-bits, 16 iterations (rounds) of substitution/permutation encryption, and a final permutation. Each of the 16 rounds consists of the selection of a 48-bit secondary encryption key from the 64-bit primary key according to a changing key schedule, expansion of the rightmost 32 bits of the text into 48 bits, an exclusive-or of these 48 bits with the secondary key, substitution of a 4-bit block for each 6-bit block, permutation of the resultant 32 bits, an exclusive-or of the leftmost 32 bits with the new rightmost 32 bits to produce a new right half of the block, and an interchange of the right and left halves of the block. [NBS, 77-1].

Since frequent permutations must be performed, any microprocessor implementation of DES would be forced to store only one bit of the block per addressable byte to prevent massive performance degradation.

The elementary operations which comprise DES are very simple and are available directly on all currently popular microprocessors. Because the secondary key selection algorithm is time consuming, it is preferable that all secondary keys be precomputed and stored (one bit per byte) for immediate use during the encryption.

Once the block is in memory, one bit per word, the initial permutation can be made, moving the block to another array. (To save transfer costs, two alternating working arrays can be used.)

The substitution-permutation encryption is best performed in a loop for space considerations. By storing all secondary keys and substitution tables with one bit per word, the use of masks can be avoided with a significant time savings.

The final permutation is similar to the initial permutation. The instructions and the number of times each is executed are summarized in Table 1. This gives a total of 23648 cycles at 1.2\(\mu\text{sec}\) each, for a total throughput of 2200 bits/second. Since there will certainly be other overhead, a more reasonable estimate of throughput would be 1800-2000 bits/second.

Storage requirements for this algorithm are summarized in Table 2. There exists which can implement DES in 5K words of memory and produce a throughput of 1800-2000 bits/seconds.

If storage is constrained to 4K, the best place to economize is in the substitution tables, storing 4 bits to the word. This would then require masking the bits during the substitution, with the resultant increase in the number of machine cycles causing throughput to drop to approximately 1600-1800 bits/second.

A microprocessor compatible special purpose DES chip is currently available with a throughput of 1200 bits/second. While this would clearly be much easier to make use of than 5K program, it is
Step | Type of Instruction(s) | Number Times Executed
--- | --- | ---
Initial Permutation | (LOAD, STORE) | 48 1
Substitution-Permutation | (LOAD, STORE) | 48
| (LOAD, XOR (indirect), STORE) | 48
Substitute | (SHIFT, ADD) - Deciding what to substitute | 48
| (LOAD, STORE) - Doing the substitution | 32 16
Permutation | (LOAD, STORE) | 32
Loop Counters and key select overhead | (LOAD, XOR, STORE) | 10
Left-Right XOR | (LOAD, STORE) | 32
Interchange Left and Right | (LOAD, STORE) | 64 15
Final Permutation | (LOAD, STORE) | 64 1

Table 1. Instructions Needed for DES

<table>
<thead>
<tr>
<th>Code</th>
<th>Keys (Secondary)</th>
<th>Substitution Tables</th>
<th>Work Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1760 words</td>
<td>16 x 48 = 768 words</td>
<td>2048 words</td>
<td>192 words</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td>4768 words</td>
</tr>
</tbody>
</table>

Table 2. Storage Needed for Implementation of DES

rather surprising that a higher data rate can be obtained in a software implementation which can be retrofitted to existing micros or intelligent terminals. Note however, that in our analysis of the encryption times, secondary key generation was considered to be a one time cost and not included. For short messages, this time would not Since only one bit is being stored per word, be trivial. The hardware implementation does not have this extra time cost.

the DES algorithm would be easily implemented on an 8-bit micro. Assuming a 2^{sec} cycle time, this would give a throughput rate of 1000-1200 bits per second in the 5K implementation or 900-1100 bits per second in the 4K implementation. It must be noted that the National Bureau of Standards (NBS) will not certify a software implementation of DES [NBS, 77-2]. Hence, there are many applications in which (for policy reasons) a micro based implementation of DES would not be used even if there were no technical problems.

Implementation of the RSA Algorithm

The Rivest, Shamir, Adleman public key algorithm is a numerical encryption based on the difficulty in factoring large numbers. As such, it is currently feasible only on 16-bit micros which have extended capabilities for multiplication. We will take 20^{sec} per cycle as a typical time for multiplying two 16-bit numbers.

In the RSA algorithm, a user selects two large prime numbers p and q, each on the order of 100 digits (320 bits). The user then selects a number e. Together n=pq and e form the encryption key. n and e are made public. The user then computes an integer d from p, q, and e. Together, n and d form the decryption key. These are kept secret. The size of the product de is on the order of n. In the worst case, either d or e will be the order of n. We will assume that this is true in the following.

To create an enciphered text C from a message M, the computation:

\[ C = M^e \mod n \]

is performed.

To decrypt, the recipient performs:

\[ M = C^d \mod n \]

Either computation can be performed by repeatedly squaring M (or C) modulo n and multiplying the running partial product of C (or M) by appropriate squares as dictated by e (or d). To keep the size of the numbers to a reasonable level, the squares and partial products are reduced modulo n after each operation.

Thus, there must be on the average 640 squaring operations performed on a 640-bit number, 320 multiplications (since about half the bits of e or d will be 1), and 960-bit modulo C operations. Since only 16-bit operations are available, each of these operations on large numbers of these must be done in stages. The techniques used for operations on large numbers are outlined in [Knuth,69]. Essentially, the computations are performed in steps resembling longhand operations on 16-bit segments of the number. In the squaring
operations, large savings can be obtained by noting that many of the partial products are identical, and that multiplication by 2 can be performed by a left shift.

Each squaring operation requires 820 16-bit multiplications, and 1600 additions and shifts for a total time of 20240 μsec. Each partial product multiplication requires 1600 16-bit multiplications and 3200 additions for a total time of 39680 μsec.

Performing a modulo n reduction can be done by precomputing a 640-bit approximation of the inverse of n. This can be performed in time \( O(640) \) [Aho, 74].

Consider \( C = \left[ \begin{array}{ll} B_1 & B_2 \end{array} \right] \) Dividend
\[
\begin{array}{c}
\text{where each block is 640 bits long.}
\end{array}
\]

The residue \( r \) is found by:

\[
\begin{align*}
\text{multiply } B_1 B_2 \text{ by } 1/n \text{ to obtain } q \\
\text{set } r' = B_1 B_2 - q \cdot n \\
\text{if } r' = 0 \text{ then } r = r' + n \\
\text{else } r = r'
\end{align*}
\]

To multiply \( B_1 B_2 \) by 1/n requires 3200 16-bit multiplications and 6320 16-bit additions. To find \( r' \) requires 1600 16-bit multiplications and 3200 16-bit additions. This gives a total time for one modulo n operation of 0.12 sec.

The 640 squaring, 320 long multiplications and 960 modulo operations required to encrypt a block thus require:

\[
\begin{align*}
T &= (0.12 + 0.13 + 115.2) \text{ sec} \\
T &= 115.5 \text{ sec}
\end{align*}
\]

Thus it takes about 120 seconds to encrypt a 640 bit message, giving a throughput rate of 5 bits/sec.

There is a potential for savings since the modulo operation need not always be performed. (The square or partial product may already be less than \( n \).) Also, if the encryption and decryption keys are both on the order of \( \log n \), a time savings of 50% can be achieved by only having to consider 32 random bits are reasonable. The \( e_i \) are computed from the \( d_i \), and are on the order of 4096 bits. The product \( T d_i \) is on the order of 4096 bits.

As illustrated above, encryption consists of 16 long multiplications, 15 long additions, and one long division.

To multiply a 4096 bit number by a 256 bit number, split each into 16-bit segments which can be multiplied directly. Multiply all 16-bit segments of the multiplier. The requires 4096 16-bit multiplications, each producing a 32-bit partial product. To form the final product from these requires about 8000 16-bit additions. On our micro, this requires approximately \( 0.1 \) sec. Each long addition (4252 bits) requires 272 16-bit additions for a total addition time of 9.7 sec. Sixteen long multiplications must be performed, followed by 15 long additions.

As we saw in the RSA algorithm, the time required to determine residues modulo a 640 bit number by division is large. For a 4096 bit modulus, the time requirements are far worse. However, due to certain properties of this cipher system, an easier method exists. The result of the summation will be no larger than \( 2^{256} T d_i \). One long addition (4252 bits) requires 272 16-bit additions for a total addition time of 9.7 sec. Sixteen long multiplications must be performed, followed by 15 long additions.

As we saw in the RSA algorithm, the time required to determine residues modulo a 640 bit number by division is large. For a 4096 bit modulus, the time requirements are far worse. However, due to certain properties of this cipher system, an easier method exists. The result of the summation will be no larger than \( 2^{256} T d_i \). One long addition (4252 bits) requires 272 16-bit additions for a total addition time of 9.7 sec. Sixteen long multiplications must be performed, followed by 15 long additions.

As we saw in the RSA algorithm, the time required to determine residues modulo a 640 bit number by division is large. For a 4096 bit modulus, the time requirements are far worse. However, due to certain properties of this cipher system, an easier method exists. The result of the summation will be no larger than \( 2^{256} T d_i \). One long addition (4252 bits) requires 272 16-bit additions for a total addition time of 9.7 sec. Sixteen long multiplications must be performed, followed by 15 long additions.

As we saw in the RSA algorithm, the time required to determine residues modulo a 640 bit number by division is large. For a 4096 bit modulus, the time requirements are far worse. However, due to certain properties of this cipher system, an easier method exists. The result of the summation will be no larger than \( 2^{256} T d_i \). One long addition (4252 bits) requires 272 16-bit additions for a total addition time of 9.7 sec. Sixteen long multiplications must be performed, followed by 15 long additions.
Consider:

\[
\begin{align*}
C \left[ B_1 \right] & \left[ B_2 \right] \ldots \left[ B_{16} \right] \\
\text{Dividend} & \\
\text{Divisor} &
\end{align*}
\]

where each block is 256 bits long.

The residue \( x_i \parallel f_i \) is found by:

1. multiply \( B_1 \parallel B_2 \) by \( 1/d_i \) to obtain \( q \)
   set \( r_2 = B_2 - q \cdot d_i \)
   if \( r_2 < 0 \) then \( r_2 = r_2 + d_i \)
   else \( r_2 = r_2 \)

2. repeat (1) for \( r_2 \parallel B_3, r_3 \parallel B_4 \ldots \) to \( r_15 \parallel B_{16} \)
   get \( r_{16} \), the residue.

Multiplying \( B_i \parallel B_j \) by \( 1/d_i \) requires 512 16-bit multiplications and 992 16-bit additions. Multiplying \( q \) by \( d_i \) requires 256 16-bit multiplications and 496 16-bit additions. Determining \( r_2 \) requires 32 16-bit additions. Since the entire process must be done 16 times, 12288 16-bit multiplications and 24320 16-bit additions are required.

This gives a decryption time of 0.31 sec for each field, and a throughput rate of 735 bits per second or 700 bits per second allowing for overhead.

As in the RSA algorithm, the amount of overhead is comparatively small because of the tightness of the computations.

This particular implementation can run in 4K of memory. It should be noted that since each field is decrypted separately, an array of 16 microprocessors could increase throughput to over 11700 bits/sec by deciphering the fields simultaneously.

### Key Management

Using micros for encryption forces thought to be given to the problems of key management. Since the key must be stored in the machine in an accessible location, care must be taken to ensure that the encryption program does not accidentally or intentionally transmit the key (or some form of information from which the key can be computed), and that the key and all workspaces are purged after use to ensure that a subsequent use of the machine will not compromise the key.

### Alternative Implementations

There are really only two types of alternative implementations for these encryption algorithms: special purpose hardware and table driven encryption. Because of the block sizes involved, table look up is not applicable to any of these schemes.

Special purpose hardware which is microprocessor compatible can offer substantial benefits. A number of hardware versions of DES are currently available. There is at least one special purpose board on the market designed specifically for microprocessor use. While its throughput rate is low, improvements seem likely.

Multiplication hardware capable of directly multiplying long numbers would greatly speed up both the RSA and DWK algorithms.

Work is also being done at MIT on a programmable device capable of achieving a throughput of about 4000 bits per second for the RSA algorithm. While this is not a micro, it is similar in scale [Adleman, 79].

### Table 3. Comparison of Throughput Rates

<table>
<thead>
<tr>
<th></th>
<th>DES</th>
<th>RSA</th>
<th>DWK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>64 bits</td>
<td>640 bit</td>
<td>4096 bit</td>
</tr>
<tr>
<td>Assumed</td>
<td>(3648 bits information)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Encrypt</td>
<td>58 msec</td>
<td>120 sec</td>
<td>1.6 sec</td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Decrypt</td>
<td>58 msec</td>
<td>120 sec</td>
<td>4.96 sec</td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Decrypt</td>
<td></td>
<td></td>
<td>0.31 sec</td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encrypt Throughput Rate</td>
<td>1800-2000 bps</td>
<td>5 bps</td>
<td>2000 bps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decrypt Throughput Rate</td>
<td>1800-2000 bps</td>
<td>5 bps</td>
<td>735 bps</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assumption: 16 bit micro</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4 sec instruction time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 sec multiplication time</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 458 |
Conclusions

The calculated throughput rates are summarized in Table 3.

Since the highest rate which can be achieved by any of these algorithms is 2000 bits per second, the use of current micros for encryption in database machines and disk/channel interfaces is unrealistic.

None of the algorithms require excessive memory, so all can be used for offline encryption/decryption when speed is not a major consideration.

The DES and DWK algorithms have sufficient speed to be used in intelligent terminals or terminal front ends if some performance degradation can be accepted to gain security. The DES algorithm causes the least degradation. The DWK database encryption algorithm can only decrypt at slightly more than 700 bits per second. However, this will be sufficient for use in the type of interactive database queries which it was designed for.

With the specialized hardware previously discussed, a micro based RSA scheme should be able to function in an electronic mail system or at terminal speeds. Such hardware would also eliminate all of the problems which the DWK scheme has with meeting terminal speeds.

Thus it seems that microprocessors have a bright future in providing low cost data security for most types of terminals.

BIBLIOGRAPHY


