Security Management in Communications Systems

Everything You've Always Wanted to Know About Public-Key Cryptosystems

by

Eric Howard Michelman

B.S., Massachusetts Institute of Technology
(1976)

M.S., University of California, Berkeley
(1977)

Submitted in Partial Fulfillment
of the Requirements for the
Degree of

Master of Science

at the

Massachusetts Institute of Technology

June, 1978

Signature of Author Signature redacted 5/8/78
Sloan School of Management, Date

Certified by Signature redacted
Thesis Supervisor

Accepted by Signature redacted
Chairman, Departmental Committee of Theses

Security Management in Communications Systems
Everything You've Always Wanted to Know About Public-Key Cryptosystems

by

Eric Howard Michelman

Submitted to the Sloan School of Management
on May 12, 1978 in partial fulfillment of the requirements
for the Degree of Master of Science

ABSTRACT

Recently the idea of public-key cryptosystems has received a lot of attention, particularly the implementation algorithm suggested by Rivest, Shamir, and Adleman. This thesis analyzes that algorithm with a view towards applications considerations.

The technical properties of the algorithm are first discussed, including the actual steps used to encrypt/decrypt and to effect digital signatures. Special attention is paid to the functional capabilities of signatures in commercial applications. The speed and cost of both hardware and software implementations are analyzed and then compared to the National Bureau of Standard's Data Encryption Standard (DES). The ease of system operation and security effectiveness are also compared to the DES. Finally, an example application in a commercial bank's money transfer system is examined.

It appears that public-key cryptosystems provide substantially more functional capabilities than conventional systems. The two primary advantages are the ability to easily effect signatures and the ability to make public the encryption key, which vastly simplifies the key management problem. There is a question, though, as to whether this particular algorithm can be implemented economically enough for widespread commercial application.

Thesis Supervisor: Jeffrey A. Meldman, Assistant Professor of Management Science

1. INTRODUCTION

Within the past two years, there has been a major advance in the area of communications security, that of both the idea and a practical way to implement public-key cryptosystems (PKCS).

Public-key cryptosystems make use of a method for encryption and decryption in which the encryption key is different from the decryption key. Not only are the keys different, but revealing one doesn't provide any practical help in determining the other. A particular value for one key does of course "set" the value for the other, but for practical purposes, the other key is impossible to determine without additional information.

One major implication of this scheme is that encryption keys can be public; literally anyone can have access to them without threatening the security of encrypted communications. This eliminates the need for a secure transferral of keys, as is the case with conventional encryption methods.

In systems using conventional encryption methods, before two parties can communicate, they must agree on a key to be used for the encryption and decryption. This key must be kept secret, as well, or someone who intercepts a message will be able to read and modify it. This agreement on a key is generally either very expensive and time consuming or relatively insecure.

Public-key cryptosystems eliminate this problem entirely. A potential

recipient of messages simply tells the world what key to use for encryption. Everyone then knows how to encrypt messages being sent to him, but only he is able to decrypt them. There is never any need to transmit the decryption key anywhere.

The second major implication of PKCS is that it turns out to be possible to "sign" messages in a way that is unforgeable but easily verifiable. In other words, John can send Mary a "signed" message which she can prove came from him, even though the content and the encryption key may be public knowlege. This is due to the fact that the keys can be used in either order; encrypting with the private key and then decrypting with the public key produces the original message, although with this order of use there would be no security since anyone could use the public key to read the encrypted message. To sign a message, John first encrypts the message with his secret key (yes, his decryption key), then encrypts the result with Mary's public key (her encryption key). Mary now has a doubly encrypted message which she first decrypts using her private decryption key, then decrypts again using John's public key (his encryption key) to arrive at the English message. Since only John knows his private key, only John can create an encoded message which produces English when his public key is applied to it.

The idea of PKCS was first publicized in an article by Diffie and Hellman of Stanford (Diffie 1976). In this article, they discussed the advantages that could be obtained from an algorithm which allowed the implementation of PKCS. They did not, however, suggest a specific algorithm. A specific algorithm was first published in April 1977 by Profs. Ronald Rivest, Adi Shamir, and Len Adleman, all of MIT. This algorithm is based on the computational difficulty of factoring large numbers. This thesis is based on that algorithm, the only one published to date.

These, then, are the major capabilities of PKCS, as understood at this time. The purpose of this thesis is to explore the issues involved in implementing and using PKCS. These issues include:

a) Security effectiveness--both technical and operational.

b) Cost of implementing and using a PKCS.

c) Speed--how fast can messages be encrypted and decrypted? This is particularly relevent for software implementations.

d) Organizational implications--what extra capabilities do PKCS offer system users? What conveniences are provided?

The thesis is divided into eight sections, including the introduction. Section 2 discusses the technical aspects of PKCS. It is taken largely from Rivest's paper.

Section 3 discusses many of the design considerations of PKCS. It also covers in more detail the advantages to be gained by their use.

Section 4 discusses the speed limitations of software implementations. The operations are broken down into computer instructions, with the time requirements determined for both encryption/decryption and key generation.

Section 5 discusses different common structures for communications systems and for each compares the cost, effectiveness, and convenience of using PKCS with the corresponding attributes of the DES (data encryption standard of the National Bureau of Standards) implementation.

Section 6 analyzes a case study based on the international funds transfer and message switching systems used by major money-center banks. While these systems do not incorporate the PKCS approach, a detailed comparison is made between their present operations and what the operations would be if the PKCS approach was used.

Section 7 discusses future projections and suggestions for future work.

The last section comprises a summary

2. THEORETICAL BACKGROUND

As discussed in the Introduction, PKCS have the property that revealing an encryption key does not reveal the corresponding decryption key. The two major advantages of this in communications systems are:

1) A secure transmittal of keys is not necessary. With conventional systems, an initial secure communication is necessary to transmit the encryption and decryption keys to the sender and receiver, respectively. With PKCS the keys may be sent over insecure communication channels.

2) Messages con be electronically "signed" in a way that is unforgeable, but easily verifiable.

2.1 PKCS CONCEPTUAL BASIS

In a PKCS, the encryption and decryption keys need to have the following properties (Rivest 1977):

a) Deciphering an encrypted message yields the original message.

b) Encryption and decryption is easily performed.

c) Revealing the encryption key does not reveal an easy way to determine the decryption key.

d) Applying the decryption key to a message, and then applying the encryption key to that result (the reverse of the natural order), will yield the original message. This is necessary for the signature capability.

A method satisfying (a), (b), and (c) is known as a "trap-door one-way function" If it also satisfies (d), it is a "trap-door one-way permutation". These methods are called "one-way" because they are difficult to compute in the other direction. They are known as "trap-door" because they are easy to compute in either direction once certain private information is known. A method satisfying (d) is called a permutation because each message may be encrypted and decrypted in either order.

Diffie and Hellman's article provides an excellent description of the concept of PKCS (Diffie 1976).

2.2 CONVENTIONAL ENCRYPTION METHODS

With the rapidly increasing amount of information being stored in computerized data banks and being transmitted over communications lines, efficient and effective encryption techniques are needed.

Recently the National Bureau of Standards adopted the Data Encyption Standard (DES) developed by IBM. DES is based on a 56 bit key (plus 8 parity bits) which is used to encrypt messages in 64 bit blocks. The algorithm has the desir able property that each bit of the ciphertext is a function of all the bits of both the plaintext and the

key. Each group of 64 bits of plaintext is completely independent, however, so there is no need for time or position synchronization. The DES performs the encryption function through combinations of substitutions and permutations involving the message, the key, and eight selection functions (S-boxes) which each consist of a static set of 64 integers between 0 and 15.

DES does not satisfy the property that revealing the encryption key does not reveal the decryption key; it is therefore not useable for a PKCS.

All conventional encryption methods (including DES) suffer from key distribution or "key management" problem. As disussed above, before two parties can communicate, an initial private transaction is necessary to distribute the encryption and decryption keys to the two parties. Often a private courier is used. For a communications system with many users, however, this would be a very expensive and time-consuming approach. A PKCS doesn't have this problem, as only the encryption keys need to be transmitted (each user sends the other his encryption key) and that may be done over insecure communication channels since the keys are public.

2.3 THE ENCRYPTION AND DECRYPTION METHODS SUGGESTED BY RIVEST

Both the encryption key and the decryption key are composed of a pair of numbers, (e,n) and (d,n) respectively.

To encrypt a message, the message must first be represented as an integer between 0 and n-1. This can be done in any way that is convenient, such as stringing

together the characters' ASCII representations. Long messages can be broken up into a series of blocks of this size, with each block treated as an individual message. The purpose of this is not to encrypt the message, only to put it into numeric form as required by the encryption procedure.

The encryption is performed by simply raising the message to the e-th power modulo n. That is, the encrypted message C (for ciphertext) is obtained by taking the remainder of the message M raised to the power e and then divided by n.

Decryption is similar to encryption. The d key is used where the e key is used for encryption.

Stated as formulas, the encryption and decryption algorithms are:

Encryption: $C = M^e \pmod{n}$ for a message M

Decryption: $M = C^d \pmod{n}$ for a ciphertext C

The encryption key, then, is the pair of integers (e,n) and the decryption key is the pair of integers (d,n).

2.3.1 CHOOSING THE ENCRYPTION AND DECRYPTION KEYS

The first step is to generate two very large primes, p and q. To compute n, simply multiply p and q:

$$n = p*q$$

The d key is a large random number which is prime relative to (p-1)*(q-1). This implies that

$$\gcd(d,(p-1)*(q-1)) = 1$$

where gcd means greatest common denominator.

Finally, the e key is computed from p, q, and d to be the "multiplicative inverse" of d modulo (p-1)*(q-1):

$$e*d = 1 \pmod{(p-1)*(q-1)}$$

Although n will be revealed publicly, it will be virtually impossible for others to determine p and q by factoring n. This is due to the tremendous computational difficulty involved in factoring very large numbers.

An example of generating keys and then encrypting a message is presented in Section 4.1.3.

The size of the keys involved are typically, but not necessarily, about 50 to 200 decimal digits for p and q and about 100-200 decimal digits for n. In Section 4 the speed and security tradeoffs presented by the key size are explored. Section 4 also provides the details on how to compute p, q, e, and d using fairly simple programs (Rivest

1977)

3. DESIGN CONSIDERATIONS FOR PKCS COMMUNICATIONS SYSTEMS

There are several issues involved in determining the applicability of a PKCS approach to a particular communication system. These range from what the actual costs and benefits are to the options available for implementing a PKCS.

The major advantages a PKCS has over a conventional approach are added security, easier security management (leading in practice to added security), and the ability to effect unforgeable electronic signatures.

In implementation, the two options are software and hardware. The primary functions involved are encryption/decryption and key generation. These can be performed by software programs or by special hardware devices. A hardware implementation can be implemented in several ways, considering both the medium of implementation (e.g., microprocessor or a specially designed chip) and where it is placed in the communications system (e.g., between the communications line and the terminal or computer, or components inside the terminal or computer). A software implementation must run on the user's computer or a dedicated CPU, which would likely be a microprocessor (which implies that a microprocessor would be a combination of both a hardware and software implementation). It should be noted that there is a lot of flexibility in designing a system in terms of deciding which functions to implement in hardware and which to implement in software.

3.1 SECURITY

The security considerations of PKCS have two major aspects, technical and organizational. The technical security comes from the difficulty in deciphering a message without the decryption key. The organizational aspect is based on the administrative procedures involved in the operation of a PKCS.

3.1.1 TECHNICAL SECURITY

Technically, PKCS provide very robust security control. The size of the n parameter provides a built-in trade-off between cost and security.

From a theoretical point of view, PKCS provide virtually unbreakable security. It should be noted, though, that they are based on a class of mathematical problems which are strongly believed to be intractable, but this hasn't been proven yet.

In terms of actual security, using the fastest factoring algorithm known to the authors of the Rivest paper (RIVEST 1977), the amount of time needed to crack a decryption key is given in the following table. This table, taken directly from that paper, shows the estimated time needed to break the key based on factoring the n parameter on a computer which performs 1 operation per microsecond.

Digits in n	Number of Operations	Time
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years

300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

There does exist the possibility that a faster factoring algorithm will be discovered. In that case, the time needed to crack the system with a given key length will decrease accordingly. However it is likely that any improvement in the factoring algorithm will produce a marginal rather than a dramatic improvement, although that would be relative to the key length in use. An improvement of a factor of ten, for example, would be a matter of concern with a key length of 50, but not with a key length of 200. A breakthrough which would render this algorithm unuseable would likely require solving the general class of mathematical problems mentioned above, which isn't believed possible.

3.1.2 ORGANIZATIONAL SECURITY

In this area of security, PKCS win big over conventional systems. In systems where the encryption key is the same as the decryption key, two parties must agree on a key prior to commencing useful communication. This is often an expensive and time consuming process. There is always a tradeoff between security and expense--the easier and cheaper ways to transfer the keys, such as by telephone or mail, are relatively insecure. The more secure methods, such as by trusted carrier, are expensive, especially over long distances. Often these systems are not as secure as they should be because of the expense involved and the tradeoff present between expense and security. The risk exposure may be greater than the cost of the desired security; however as is often the case in computer security, the risk of possible security failures in the future is preferable to the manager in charge than the immediate outflow of dollars necessary to provide the

desired level of security.

PKCS eliminate this problem by simply allowing everyone to know the key. It can be sent via telephone, mail, or, probably the most convenient way, over the communications lines. Using the communications lines to broadcast the new public keys is especially appealing, as then the new key can be signed with the old key to prevent an impostor from sending out a new encryption key for someone else with the impostor having the new decryption key.

Overall, security management, and key management in particular, becomes vastly simplified using PKCS. Section 5 examines this in depth by comparing PKCS with comparable systems using the National Bureau of Standards' Data Encryption Standard.

3.2 SIGNATURES

3.2.1 USES

There are two primary uses for signatures in communications systems--future verification and immediate identification.

Future verification is potentially very important in commercial systems where users may want to effect a contract over the system. In this sense the signature is just like a written signature; its purpose is to prove at a later date that the sender did in fact author the message. This is valuable and particularly important among mutually suspicious parties, such as two banks executing a contract over communications lines.

Immediate identification is important for a more technical reason. In an electronic communications system, if someone taps into the communications line he can originate a message purporting to be from someone else. However, with the use of PKCS signatures, a receiver of a message can verify that the message is in fact from who it claims. This is important for both mutually suspicious parties and those who are not. In communications between two trusted parties, one still wants to ensure that they cannot be impersonated.

3.2.2 ADDED COST

As always, nothing is free. There is a potentially substantial added cost to using the signature capability, depending on how it is implemented.

With a hardware implementation, the added cost is either an extra encryption/decryption device on each terminal and communications link or extra time consumed by cycling messages through one encryption box twice. This applies to both encryption and decryption.

With a software implementation, if the encyrpted message is signed as it is, it will require twice the time to encrypt and decrypt a signed message as an unsigned message. There is however, a way to save time in this operation. That is to sign a compressed version of the message and send the signed compressed version along with the unsigned version. A reduction procedure for this would have to be such that it wouldn't be practical for someone to change part of the message without changing the

compressed version.

3.2.3 NEED FOR IMPARTIAL RECORDING OF ENCRYPTION KEYS

Unfortunately, there is another operational requirement in terms of future verification. That is one must also be able to prove what the encryption key was when the message was sent before a disputed signature can be considered proven. A good example of the issue here would be a money transfer system connecting Citibank and Chase. It sounds very easy for them to send signed messages back and forth secure in the knowlege that if there is a dispute each can produce the signed messages from the other as proof of contract. However, aumming that the keys change periodically, they must also be able to prove what the other's encryption key was at the time any given message was received. Otherwise a situation could arise in which one bank fabricates a message purporting to be from the other, generates a decryption key to sign the fake message with, and claims that the message was actually received from the other and at the time it was received the corresponding encryption key was the public key in effect for the other bank.

An analogy can easily be drawn between this problem and conventional paper and ink signatures. That paper signatures can be used as proof is partly based on the fact that someone's signature doesn't change over time. In a court of law, then, the signature on a document can be compared to the person's signature at that or any other time. If people changed their signatures frequently, or at all, it would be much harder to prove that someone did in fact sign a document. Similarly, if the keys didn't change in a PKCS this problem wouldn't exist. However it does since it is desireable to change keys

from a security point of view (Changing keys both limits the exposure caused by a compromised key to the time until the next key change, and limits the time available for an intruder to crack the key while it is still in use.)

The implication of this is that for signatures to be useful for future verification between two mutually suspicious parties, an impartial referee must record the public keys with the time that they are in effect. In the bank system mentioned, that judge would most likely be the Federal Reserve Bank. In the electronic public mail systems predicted for the future, the referee might well be the U.S. Postal Service, or whatever government agency is operating the system. Section 3.2.3 discusses a less efficient alternative which also requires a trusted referee, but doesn't require that the referee keep records.

An alternative that has been suggested is to have the system controller sign with its own decryption key a message for each user confiming what any given user's encryption key is--sort of a notarized affidavit (Kohnfelder 1978). This both enables a user to prove what another user's encryption key was at a given point in time and also provides a secure means for obtaining the encryption key of others users (it prevents possible impersonation of the system controller).

3.3 HARDWARE VS. SOFTWARE IMPLEMENTATION

As mentioned above, a major design consideration is what functions to put in hardware and which to implement in software.

3.3.1 AVAILABILITY

Presently, hardware implementations are not available, although at least one hardware implementation project is under way and various companies have given some, if not a lot of, thought to a chip design.

The software programs needed are rather simple and straightforward. With multiple precision arithmetic routines to work with, they can be on the order of a couple of dozen lines of code. The multiple precision arithmetic routines are more complicated than the rest of the code, but algorithms and example programs are provided in Knuth's volume on semi-numerical algorithm's (Knuth 1969). Complete functional specifications and design instructions for encryption/decryption and key generation are provided in Rivest's paper (Rivest 1977). The problem with software implementations is that they seem to be comparatively slow. This consideration is explored in detail in Section 4.

Until a PKCS chip is available, the alternative for hardware implementations is the use of microprocessors and discrete IC circuits. The one hardware implementation project currently known is using IC ciruits. The component costs are expected to come to between 1900 and 2000 dollars for one and about $1200 in quantities of 1000's. For commercial products of this sort, the component cost is usually multiplied by 3 to 7 to arrive at selling price for a finished product.

These figures are high for commercial systems. DES boards are presently available for about $500 and microprocessor implementations are available for much less. From this it appears that PKCS won't become a viable hardware alternative until a chip is

developed and marketed. Since chips today are generally made with slower components than discrete components, there is question as well as to whether a chip could perform the necessary operations fast enough. However in view of past improvements in speed, capability, and price, it can't be right to rule that out.

Microprocessors are now far too slow in arithmetic operations to be used for PKCS applications. This is true for several reasons: the actual instruction speed is slow compared to larger machines, the word size is generally 8 bits (necessitating more operations), and they do not have cache memories. On the other hand, microprocessors are continually getting faster, 16-bit microprocessors will soon be commonplace (the Intel 8086 and TI's TMS9900), and cache memories on the chip are expected within a couple of years. So while microprocessors are not presently an adequate vehicle for implementing PKCS, they may well be in a few years. Another interesting possibility is that since message blocks can be processed independently, microprocessor arrays may become a viable alternative. In this case the speed of encrypting one block would be the effective speed for encrypting an entire message.

The cost of a software implementation, in terms of time required to perform the operations, will be discussed in detail in Section 4.

3.3.2 HARDWARE DEVICES WE'D LIKE TO SEE

One of the most attractive features of PKCS is the degree to which they can contribute to a completely automated security management system. This is accomplished best through a group of specially designed hardware devices. While none of these are

available at this time and many won't until PKCS become common, discussing what one would like to have is useful for illustrating the power and advantages of PKCS.

It should perhaps be noted that automation here refers to not requiring user action or, in some cases, even knowlege of the initiation and execution of security functions such as encryption/decryption and key generation and broadcasting. In many systems, the user will want to retain control over some or all of these functions; however in others the users' responsibilities should be limited as much as possible. Extreme examples of the later are systems where the users aren't even human, such as industrial process control monitors. In that case, building the decision making logic into all of the remote stations would be far more expensive and far less versatile than having it concentrated at one central point.

3.3.2.1 ENCRYPTION/DECRYPTION BOXES

The first and most straightforward requirement is for a hardware device to encrypt and decrypt messages. This would presumably be on both ends of the communication lines.

These can be fairly simple in functional design, requiring two inputs, the encryption/decryption key and the message. As encryption and decryption are functionally the same (encryption uses the e key while decryption uses the d key in the same function) one box can be used for both, simply supplying the appropriate key for either encryption or decryption.

3.3.2.2 SMART TERMINALS

The next step up would be terminals incorporating the encryption/decryption boxes and the appropriate control logic. We see three main functions for a smart terminal in this area: encryption and decryption, key generation, and signatures.

3.3.2.2.1 AUTOMATIC ENCRYPTION AND DECRYPTION

It would be desirable that all messages going out would be automatically encrypted with the receiver's encryption key and all incoming messages would be automatically decrypted with the user's private decryption key.

If the communication lines pass through the encryption/decryption mechanism, this is a matter of the terminal controller supplying the right keys at the right time.

3.3.2.2.2 AUTOMATIC KEY GENERATION AND BROADCAST

As discussed above, depending on the system's use and the sophistication and interest of its users, it may be desirable for the system to take responsibility for initiating key changes and indicating the desired key size. These functions would be housed in a central system controller, as described below. Upon request from the system controller, the terminal would generate a new encryption/decryption key pair. Not only would the signal to change keys come from the controller, but the size of the new key

would be sent as a parameter set by the system controller. The encryption key would be sent out over the network and the decryption key should be stored in the proper place as the new decryption key. This entire operation would be transparent to the user.

3.3.2.2.3 SIGNATURES

It seems that one would like three modes of operation, one where all messages are signed, one where messages are signed only when the terminal is requested to by the system controller, and one where messages are signed only when the user indicates.

The actual signature operation can be effected either by passing the messages through one encryption unit twice or by having two encryption units in series.

Again, whether or not the signature operation should be user controlled depends on the system and the purpose of the system. If its purpose is to ensure the identity of the sender, then it is probably more convenient to have the signature operation under system control. An example of this might be water level sensors in a large dam's flood gate control system. There is an instance in which someone proved the ability to dial into the control computer in a large dam and have it release all of the flood gates. Similarly, if the water level sensors indicated that the flood gates should be opened, the computer might well ask for a signature to make sure that a disgruntled employee didn't tap into the line and send the message.

On the other hand, in an electronic bill paying system, one would probably

want the user to retain very explicit control over what gets signed.

The major difference between these two examples seems to be that in the first, the signature is used to ensure proper identification. In the second, it is used in the conventional sense as well--for future verification. Through the signature, the user is presumably committing himself to a contract. Another way of looking at it is that the first case concerns itself with ensuring system integrity and therefore should be under system control. The second case concerns itself with user intent, clearly the domain of the user.

3.3.2.3 THE NETWORK CONTROLLER

The final major piece of hardware is the network controller. The three major responsibilities of the network controller are encryption/decryption and managing its own keys, key management for the system as a whole (to the extent discussed above), and maintenance of signature records.

3.3.2.3.1 BASIC KEY FUNCTIONS

The network controller must perform all of the operations a user's terminal needs to do, but on a larger scale. It must be able to encrypt and decrypt messages from several or many users in real time. This implies a substantial increase in both power and hardware cost compared to the simple one line terminal or encryption/decryption device.

3.3.2.3.2 KEY MANAGEMENT

It may be the network controller's responsibility to control system parameters such as time between key changes and the size of the keys generated. These parameters should be operator controlled, but then handled completely by the network controller.

If encryption is done on a link to link basis, a message is first encrypted with the encryption key of the link it is to be sent to first. That link decrypts it, determining the next place it is to be sent (either another link or the final receiver) and re-encrypts it with the appropriate key. This process continues until it reaches the intended reciever.

3.3.2.3.3 SIGNATURE RECORDS

As discussed in Section 3.2.3, for signatures to be effective for future verification, records must be kept of the public keys of system users, along with the times they are effective. When the system controller is trusted by all of the users, it would seem to be the logical entity to keep these records. With a link-to-link encryption setup or end-to-end with all traffic passing through the system controller, the system controller will have the information needed at little cost. A situation to avoid is having the users send special messages to the system controller informing it of the new keys and not having these particular messages serve the function of broadcasting the new keys. In other words, care must be taken to ensure that the keys recorded are in fact the keys in use

4. INSTRUCTIONS/SPEED ANALYSIS

One of the primary considerations in determining the usefulness of PKCS is the cost of using them. In terms of a software implementation, this refers primarily to the time required to encrypt and decrypt messages. This is dependent on the number and types of instructions necessary to carry out these operations, as well as the speed of the individual computer. In terms of a hardware implementation, this is more of a question of the cost of a hardware device which operates at the desired communications rate. Typically, a faster device requires either faster (and more expensive) components or more components to effect a greater degree of parallelism.

Since software implementations do not offer the option of a speed/cost tradeoff, the purpose of this section will be to deytermine what the speed constraints are in software encryption/decryption and key generation. This will include a description of the operations used, those operations broken down into computer instructions, and the time required to carry out these instructions on some common computers. In addition, we will look at a comparison of the time requirements for other encryption methods and the time required to break the system.

It should ne noted that these time estimates are based on simple, straightforward techniques. We are not suggesting that this is the best that can be done. With optimization techniques and coding tricks, a time reduction of as much as half might be had.

4.1 DESCRIPTION OF THE OPERATIONS USED

4.1.1 ENCRYPTION AND DECRYPTION

Operationally, encryption and decryption are indistinguishable; encryption uses the e key where decryption uses the d key.

The basic operation for generating an encrypted message C from a cleartext message M is:

$$C=M^e \pmod n$$

A procedure for computing this, which is recommended in Rivest's paper (RIVEST 1977), is called "exponentiation by repeated squaring and multiplication":

```
FOR I=LOG2(e) TO 1 BY -1
        C=REM((C*C)/n)
        IF ei =1 THEN C= REM((C*M)/n)
END
```

4.1.2 GENERATING A KEY PAIR

4.1.2.1 p & q

The p and q parameters are chosen to be any large primes. About 100 digits (base 10) is suggested; however, the length provides a tradeoff between speed and security. This is be discussed in more detail in section 3.1.1.

To pick a prime number, one can generate a random odd number and increment it by two until a prime is found. The expected number of numbers to be tested is (xln 10)/2 where x is the number of digits. For a 100-digit number, this comes to 115 tries.

To test a number b for primality, the following probabilistic procedure is suggested:

1) generate a random number $1 <= a <= b-1$

2) calculate whether $\gcd(a,b)=1$ and $J(a,b)=a^{((b-1)/2)} \pmod b$ where $J(a,b)$ is the Jacobi symbol as decribed below and gcd is the greatest common divisor. If these two steps are performed y times for a given b, and the two conditions in (2) are true each time, then there is only 1 chance in 2^y that b is not prime. Obviously for large y, 100 for example, this would be proof of primality for practical purposes. If a non-prime number were used, it would be noticed in that decryption would not work.

An efficient algorithm for calculating $J(a,b)$ as suggested in Rivest's paper is:

$J(a,b)= 1$ if $a=1$

else $J(a/2,b)*((-1)^{((b^2-1)/8)})$ if a is even

else $J(b \pmod a,a)*((-1)^{((a-1)(b-1)/4)})$

An easy algorithm for computing gcd(a,b) is:

$x_0 = a$

$x_1 = b$

$x_{i+1} = x_{i-1} \pmod{x_i}$

when $x_k = 0$, then $x_k - 1 = \gcd(a,b)$

For extra protection, it is recommended that (p-1) and (q-1) should each contain a large prime foactor. This can be done by generating a large prime number u and assigning the first prome in the series i*u+1, for i=2,4,6..., to p, and similarly for q. It is also desireable to ensure (u-1) has a large prime factor.

4.1.2.2 CALCULATING d AND e

The d key is calculated rather simply; any prime number greater than both p and q will do.

The e key, the multiplicative inverse of d (mod(p-1)*(q-1)), is more difficult to determine. To do so, calculate gcd((p-1)*(q-1),d) by the method suggested above. For each x_i, calculate a_i and b_i such that x sub i = $(a_i)*(x_0)$ + $(b_i)*(x_1)$. When $x_k - 1 = 1$ then b sub k-1 = e. To calculate the series of a's and b's (although for this application it is not necessary to calculate the a's), set $a_0 = b_1 = 1$ and $a_1 = b_0 = 0$, and then for t such that $x_i = (x_i - 2) - t*(x_i - 1)$, $a_i = (a_i - 2) - t*(a_i - 1)$ and $b_i = (b_i - 2) - t*(b_i - 1)$.

4.1.3 A SMALL EXAMPLE

For the purposes of illustration, we present here the example shown in Rivest's paper (Rivest 1977) of actually generating keys and encrypting a message. The key size is small to allow the reader to follow the calculations.

"Consider the case p = 47, q = 59, n = p*q = 47*59 = 2773, and d = 157. Then [(p-1)*(q-1)] = 2668, and e can be computed as follows:

x_0=2668, a_0=1, b_0=0,

x_1=157, a_1=0, b_1=1,

x_2=156, a_2=1, b_2=-16 (since 2668=157*16+156),

x_3=1, a_3=-1, b_3=17 (since 157=1*156+1).

Therefore e = 17, the multiplicative inverse (mod 1668) of a = 157.

"With n = 2773 we can encode two letters per block, substituting a two-digit number for each letter: blank = 00, A = 01, B = 02,..., Z = 26. Thus the message

ITS ALL GREEK TO ME

(Julius Caesar, I, ii, 288, paraphrased) is encoded:

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

Since e = 10001 in binary, the first block (M = 920) is enciphered:

$M^{17} = (((((1)^2*M)^2)^2)^2)^2 * M = 948 \pmod{2773}$.

The whole message is enciphered as:

0948 2342 1084 1444 2663 2390 0778 0774 0219 1655

The reader can check that deciphering works: $948^{157} = 920 \pmod{2773}$, etc."

4.2 COMPUTER OPERATIONS BREAKDOWN

In this section we examine the breakdown of the operations described in 4.1 into the actual computer operations necessary for execution. This includes both type and number.

As all of the functions descibed make use of multiple precision multipication and division, we will first examine the computer instructions for these.

4.2.1 MULTIPLE PRECISION MULTIPLICATION AND DIVISION

With an n size of 200 digits, one must deal with internal representations of over 1300 bits long. Since computer instructions don't handle numbers that long, software routines must be used.

The multiplication and division routines analyzed here are basically from Knuth Vol.2 (Knuth 1969). A combination is used, in that each operand is assumed to be split into eighths and the basic "high school" algorithm is used in multiplying the individual eighth sections. They are split into eighths to utilize the following technique:

$A = A_0 + (2^n)*A_1$ example: $110010 = 010 + (2^3)*110$

$B = B_0 + (2^n)*B_1$

$A*B = A_1B_1*(2^{2n}) + (A_1B_0+A_0B_1)(2^n) + A_0B_0$

Note that only three multiplications were required, whereas in the basic algorithm, if A and B where each more than one computer word long, 4 multiplications would have been necessary. Note also, that multiplying by a power of two in a computer is only a shift

operation and can be done very quickly. Since splitting the operands in half yields only 3/4 the number of multiplications that would be needed otherwise, splitting them into eighths yields $(3/4)^3 = 27/64$ multiplications. We don't suggest splitting them up into more than eighths since splitting an operand that isn't more than one computer word gains nothing.

Adding time for overhead to the 27/64 figure, we will estimate that multiplications are done in half the time estimated by Knuth's basic algorithm (he does also describe more advanced algorithms, including the one just mentioned, but he only provides detailed time estimates for the basic one).

Both routines break the numbers to be operated on into word size "digits". For example, with a word size of 32, a 320-bit number would be considered as a 10-digit number. The number of these digits in each operand is used as a parameter in detemining the number of actual machine instructions.

Knuth also calculates the number of machine cycles needed to execute each routine. One cycle is counted for each instruction fetch plus one for each additional memory reference needed. The multiplication instruction is estimated to be 10 cycles.

To multiply an M digit number by an N digit number requires

$$28MN + 7M + 4N + 3$$

cycles with Knuth's procedure. Again, the number of digits in the operands really refers to the number of machine words taken up by the operands.

If we separate the number of cycles required for the actual hardware

multiple instructions (as opposed to these interpretive software multiply instructions) from the total number of cycles, we get:

18MN + 7M + 4N +3 cycles

plus MN hardware multiple instructions

To divide an (M+1) digit quotient by an N digit divisor requires

30MN + 30N + 91M + 113

cycles with Knuth's procedure.

Again, if we separate the number of cycles required for the hardware multiply instructions (no division instructions are used), we get

20 MN + 20N + 66M + 93 cycles

plus MN + 2.5M + n + 2 multiplication instructions

It should be noted that the number of cycles for these procedures includes all of the control logic for carrying out the operations in subroutines.

4.2.2 COMPUTER INSTRUCTIONS FOR ENCRYPTION AND DECRYPTION

From Section 4.2.1, encryption requires $1.5\log_2(e)$ multiplications, of which

$\log_2(e)$ will be C*C where $0 <= C <= n$

$.5\log_2(e)$ will be C*M where $0 <= C <= n$

and M is on the order of n.

In addition, there are $1.5\log_2$ divisions, of which all are of the form

X/n where $0 <= X <= n^2$

For the multiplications of the form C*C, there are

$$28((2\log_2 n)/b) + 11((\log_2(n/2))/b) + 3$$

cycles, or

$$18((2\log_2 n)/b) + 11((\log_2(n/2))/b) + 3 \text{ cycles}$$

$$\text{plus } (2\log_2 n)/b \text{ multiplication instructions}$$

where b is the number of words in a machine word. Actually, the term $(2\log_2 n)/b$ should be $(2\log_2 n)/b - 2.9/b$, however the $2.9/b$ factor is small enough to be ignored. This was arrived at by integrating to find the expected value of C^2.

For the multiplications of the form C*M, the calculations are simpler since the expected lengths of the operands are independent. This comes to

$$28((\log_2 n/b)X(\log_2(n/2))/b) + 7((\log_2(n/2))/b) + 4((\log_2 n)/b) + 3$$

cycles, or

$$18((\log_2 n)/b)X(\log_2(n/2))/b) + 7((\log_2(n/2))/b) + 4((\log_2 n)/b) + 3 \text{ cycles}$$

$$\text{plus } ((\log_2 n)/b)X(\log_2(n/2)/b) \text{ multiplication instructions}$$

For the divisions, there are:

$$30(((2\log_2 n)/b)-1)X(\log_2(n/2))/b) + 30((\log_2(n/2))/b) + 91(((2\log_2 n)/b)-1) + 113$$

cycles, or

$$20(((2\log_2 n)/b)-1)((\log_2(n/2))/b) + 20((\log_2(n/2))/b)$$
$$+ 66(((2\log_2 n)/b)-1) + 93 \text{ cycles}$$
$$\text{plus } (((2\log_2 n)/b)-1)((\log_2(n/2))/b) + ((\log_2(n/2))/b)$$
$$+ 2.5(((2\log_2 n)/b)-1) + 2 \text{ multiply instructions}$$

Summing up for all the multiplication and division operations, with the proper weights, we get the following number of cycles to encrypt a message of size $\log_2 n$ bits:

$$(\log_2 e)(28L + 11((\log_2(n/2))/b) + 3)$$
$$+ (.5\log_2 e)(28((\log_2 n)/b)((\log_2(n/2))/b) +$$
$$7((\log_2(n/2))/b) + 4((\log_2 n)/b) + 3)$$
$$+ (1.5\log_2 e)(30(L-1)((\log_2(n/2))/b) +$$
$$30((\log_2(n/2))/b) + 91(L-1) + 113)$$

cycles, or

$$(\log_2 e)(28L + 11((\log_2(n/2))/b) + 3)$$
$$+(.5\log_2 e)(18((\log_2 n)/b)((\log_2(n/2))/b)$$
$$+ 7((\log_2(n/2))/b) + 4((\log_2 n)/b) + 3)$$
$$+(1.5\log_2 e)(20((L-1)(\log_2(n/2))/b)+20((\log_2(n/2))/b)$$
$$+66(L-1)+93) \text{ cycles}$$

$$\text{plus } (\log_2 e)((2\log_2 n)/b)+$$
$$(.5\log_2 e)(L*((\log_2(n/2)/b)+$$
$$(1.5\log_2 e)(2.5(L-1)+2)) \text{ multiply instructions}$$

where $L = (2\log_2 n)/b$.

4.2.3 COMPUTER INSTRUCTIONS FOR KEY GENERATION

4.2.3.1 GENERATING A PRIME NUMBER

With the procedure recommended in Rivest's paper for testing primality, the number of steps involved in generating an x-digit prime with a probability of error of $1/(2^y)$ is:

((xln 10)/2)*

((1 addition: 2 + x digit #),

(2 * (gcd(random(1, x digit #), x digit #),

(J(random(1, x digit #), x digit #))))

+ (y-2) * ((gcd(random(1, x digit #), x digit #),

(J(random(1, x digit #), x digit #))

The major operations for calculating gcd(a,b) are:

$$(\log_{phi}(\min(a,b)))*(\text{divide}(x_i\text{-}1, x_i))$$

where the expected value of x_k is

$$(\min(a,b))/((\ln(phi))(\log_{phi}(\min(a,b))))$$

(actually, this expression slightly undervalues the value since it was arrived at by using integration to find the expected value and this considered the whole curve instead of just the integral points.) Phi is approximately equal to (sqrt(5)-1)/2. It is known as the golden ratio, from the Fibonacci series.

To calculate J(a,b), there are approximately $\log_{phi}a$ operations, with each operation being

$$J(a/2,b)*((-1)^{(((b^2)-1)/8)})$$

or $J(b(\text{mod } a),a)*((-1)^{(((a-1)*(b-1))/4)})$

This yields

$\log_{phi} a$ *

1 division with average operand size $a/((\ln(phi))*(\log_{phi} a))$

1 multiplication with average operand size "

1 division with divisor size 4 or 8 and average dividend size

"

4.2.3.2 PICKING p AND q

The p and q parameters are chosen simply as large random primes of the desired size. Section 4.3.3.1 discusses choosing primes.

Altogether, the operations for generating an x digit number with a $(1-2^{y})$ probability of primality come to

$(x\ln 10 + y - 2)(2\log_2(y/2))$ divide operations with average operand size $x/((2\ln(phi))(\log_{phi}(x/2)))$

$(x\ln 10 + y - 2)(\log_2(x/2))$ divide operations with average dividend size $x/((2\ln(phi))(\log_{phi}(x/2)))$ and a divisor of 4 or 8

$(x\ln 10 + y - 2)(\log_2(x/2))$ multiply operations with average operand size $x/((2\ln(phi))(\log_{phi}(x/2)))$

This comes to the following number of machine cycles and multiply instructions:

$(x\ln 10 + y - 2)(2\log_2(x/2))(30(Q^2-Q)+30Q+91(Q-1)+113)$ +

$$(x\ln 10 + y - 2)(\log_2(x/2))(30(Q-1)+30+91(Q-1)+113) +$$

$$(x\ln 10 + y - 2)(\log_2(x/2))(28Q^2 + 11Q + 3)$$

cycles, or

$$(x\ln 10 + y - 2)(2\log_2(x/2))(20(Q^2-Q)+20Q+66(Q-1)+93)+$$

$$(x\ln 10 + y - 2)(\log_2(x/2))(20(Q-1)+20+66(Q-1)+113)+$$

$$(x\ln 10 + y - 2)(\log_2(x/2))(18Q^2 + 11Q + 3)$$

cycles, plus

$$(x\ln 10 + y - 2)(2\log_2(x/2))((Q^2-Q)+2.5(Q-1)+Q+2) +$$

$$(x\ln 10 + y - 2)(\log_2(x/2))((Q-1)+2.5(Q-1)+1+2) +$$

$$(x\ln 10 + y - 2)(\log_2(x/2))(Q^2)$$ multiply instructions.

where $Q = (\log_2(x/(2\ln(phi)(\log_{phi}(x/2)))))/b$

4.2.3.3 CHOOSING d

As d is a randomly chosen prime, refer to section 4.3.3.2, CHOOSING p AND q, for the analysis of operations necessary.

4.2.3.4 DETERMINING e FROM d, p, and q

The only substantial operations involved in determining e are calculating gcd(d,(p-1)(q-1)). This involves $\log_{phi}d$ divisions with and average operand size of $d/(\ln(phi)(\log_{phi}d))$. This yields

$$\log_{phi}d*(30(Q^2-Q)+30Q+91(Q-1)+113)$$

cycles, or

$$\log_{phi}d*(20(Q^2-Q)+20Q+66(Q-1)+93)$$ cycles

plus $\log_{phi} d((Q^2-Q)+2.5(Q-1)+Q+2)$ multiply instructions

where $Q = (\log_2(d/(\ln(phi)\log_{phi} d))/b$

4.3 TIME REQUIREMENTS

The time requirements for encryption, decryption, and key generation are dependent primarily on message length, security desired, and machine speed and word length. In this section we will consider levels of security in terms of the size of n. Section 4.6 relates n size to time required to break the security by factoring n.

One further parameter is the probability that a prime number generated is actually prime, as discussed in Section 4.2.3.1.

The machines we will consider are the PDP11/45, PDP11/70, IBM370/168, and the CDC6400.

4.3.1 ENCRYPTION/DECRYPTION TIME REQUIREMENTS

We will consider the time required to encrypt 1000-byte messages (8000 bits). The time required to encrypt longer or shorter messages is simply that time multiplied by the length differential, e.g., a 5000-byte message would take 5 times as long as a 1000-byte message.

The number of cycles calculated here for the encryption/decryption operations is based primarily on the number of cycles in the multiple precision

multiplication and division routines used for the operations. There is also a relatively insignificant amount of overhead in controlling the encryption process, such as shifting the message blocks in and out. The extra time involved in these tasks is quite small in comparison to the actual encryption operation and will not be counted.

Note again, that the times and cycle counts shown are half those indicated by the above figures. This is to adjust for the inclusion of the multiplication algorithm described in Section 4.3.1. The numbers presented so far are those given by Knuth for the simple algorithms; they do not take account of the more economical procedures.

Since the encryption/decryption routines are small programs running in tight loops, we assume in these calculations that if a cache memory is available, the instructions will be found in the cache. Further, since the number of operands, parameters, and counters is also small, we assumne that operands are found in the general registers. While both of these assumptions will not always be true, they will be true the great majority of the time. One last assumption is that of the total number of cycles required to perform some function, half will be for accessing instructions and half will be for obtaining operands.

4.3.1.1 16-BIT MACHINE SIZE

4.3.1.1.1 200-DECIMAL-DIGIT n SIZE

The number of cycles required to encrypt a 1000-byte message on a 16-bit machine, using a 200-digit n, is

key size of 3: 2,028,863

key size of $\log_2 n$: 9,514,984

key size of n: 674,245,816

The key size is the size of the e or d key being used to encrypt of decrypt, respectively.

The key size of 3 refers to actually using 3 as a key; it is the smallest key that will work. Using a key of less than $\log_2 n$ risks the encrypted message coming out the same as the cleartext; that is, no encryption will take place. However if a message block is filled out through all $\log_2 n$ places, using a key of three will work. Message formating schemes can be easily devised to ensure that all messages will be encoded, however there is a corresponding overhead penalty.

Note that encryption and decryption of a given message require the use of two different keys, at least one of which must be on the order of size n. This implies that while a small key may be used on one end, providing very fast operation, a large key must be used on the other end, with its correspondingly higher overhead. To find the total time involved in encoding and then decoding a message, add the time required to encode with whatever size key is to be used to the time required to decode with the decryption key.

For a machine without a cache store, specifically the PDP11/45, the total encryption time is:

key size of 3: 1,826,877 microseconds, or 1.82 seconds

key size of $\log_2 n$: 8,563,485 microseconds, or 8.56 seconds

key size of n: 606,821,235 microseconds, or 10 min. 6.82 sec.

Memory access time on the PDP11 is 1.2 microseconds. The time for an add instruction, used as the average instruction execution time (except for multiply instructions, which are 10x), is .3 microseconds.

For a machine with a cache memory, the PDP11/70, the total execution time is:

key size of 3: 811,945 microseconds, or .81 seconds

key size of $\log_2 n$: 3,805,993 microseconds, or 3.8 seconds

key size of n: 269,698,325 microseconds, or 4.5 minutes

The cache access time for the 11/70 is .2 microseconds.

4.3.1.1.2 100-DIGIT n SIZE

The number of cycles for encrypting a 1000-byte message on a 16-bit machine with a 100-digit n size is:

key size of 3: 1,155,761

key size of $\log_2 n$: 4,840,230

key size of 189,144,029.

For the 11/45, without a cache, the total encryption time is:

key size of 3: 1,040,185 microseconds, or 1.04 seconds

key size of $\log_2 n$: 4,356,207 microseconds, or 4.35 sec.

key size of n: 170,229,626 microseconds, or 2 min 50.23 sec.

For the 11/70, with a cache, the total encryption time is:

key size of 3: 462,304 microseconds, or .46 seconds

key size of $\log_2 n$: 1,936,092 microseconds or 1.93 sec.

key size of n: 75,657,611 microseconds, or 1 min. 15.66 sec.

4.3.1.2 32-BIT MACHINES

In this section, we will consider the IBM 370/168, which comes with a cache memory.

4.3.1.2.1 200-DIGIT n SIZE

Encrypting a 1000-byte message with an n size of 200 digits on a 32-bit machine requires the following number of cycles:

key size of 3: 555,212

key size of $\log_2 n$: 2,602,787

key size of n: 184,437,355

On the 370/168 the basic cycle time is 80 nanoseceonds. This includes accesses to cache memory as well as instructions such as fixed point add. The encryption time for a 1000 byte message is:

key size of 3: 44,417 microseconds, or .044 seconds

key size of $\log_2 n$: 208,223 microseconds, or .21 seconds

key size of n: 14,754,988 microseconds, or 14.75 seconds

4.3.1.2.2 100-DIGIT n SIZE

Encrypting a 1000-byte messae with an n size of 100 digits requires the following number of cycles:

key size of 3: 320,366

key size of $\log_2 n$: 1,341,665

key size of n: 53,180,719

The time for encryption is:

key size of 3: 25,629 microseconds, or .026 seconds

key size of $\log_2 n$: 107,333 microseconds, or .107 seconds

key size of n: 4,254,457 microseconds, or 4.25 seconds

4.3.1.3 60-BIT MACHINE

This section considers the encryption time on a 60-bit machine, the CDC 6400. Section 4.5 compares the results with the encryption times for other encryption methods of the 6400, as discussed in Friedman's comparison of software encryption methods (FRIEDMAN 1974).

4.3.1.3.1 200-DIGIT n SIZE

Encrypting a 1000-byte message on a 60-bit machine requries the following number of cycles:

key size of 3: 179,730

key size of $\log_2 n$: 842,555

key size of n: 59,697,276

As there is no cache in this machine, we will assume that half the cycles are for instruction fetches from memory. This process is pipelined, however, resulting in a wait time of 50 nanoseconds per quarter word instruction, such as multiply and add.

The time for encrypting 1000 bytes is:

key size of 3: 112,331 microseconds, or .11 seconds

key size of $\log_2 n$: 526,597 microseconds, or .53 seconds

key size of n: 37,310,797 microseconds, or 37.31 seconds

4.3.1.3.2 100-DIGIT n

Encrypting a 1000-byte message with an n size of 100 digits requires the following number of cycles:

key size of 3: 122,878

key size of $\log_2 n$: 514,553

key size of n: 20,409,466

On the 6400, the execution time comes to:

key size of 3: 76,799 microseonds, or .077 seconds

key size of $\log_2 n$: 321,595 microseconds, or .32 seconds

key size of n: 12,755,916 microseconds, or 12.76 seconds

4.3.2 SUMMARY OF EXECUTION TIMES

The following table summarizes the execution times (in microseconds) for encrypting a 1000-byte message:

	KEY SIZE		
	3	$\log_2 n$	n
16-BIT MACHINE SIZE (the PDP11)			
200-DECIMAL-DIGIT n SIZE			
without cache store	1,826,877	8,563,485	606,821,235
with cache store	811,945	3,805,993	269,698,326
100-DECIMAL-DIGIT n SIZE			
without cache store	1,040,185	4,356,207	170,229,626
with cache store	462,304	1,936,092	75,657,611
32-BIT MACHINE SIZE (the 370/168, which has a cache)			
200-DECIMAL-DIGIT n SIZE	44,417	208,223	14,754,988
100-DECIMAL-DIGIT n SIZE	25,629	107,333	4,254,457
60-BIT MACHINE SIZE (the CDC 6400, which has no cache)			
200-DECIMAL-DIGIT n SIZE	112,331	526,597	37,310,797
100-DECIMAL-DIGIT n SIZE	76,799	321,595	12,755,416

The following table provides the number of bits per second that can be encrypted using the different key sizes and machine word sizes:

	KEY SIZE		
	3	$\log_2 n$	n
16-BIT MACHINE SIZE (the PDP11)			
200-DECIMAL-DIGIT n SIZE			
without cache store	4,379	934	13
with cache store	9,853	2,102	30
100-DECIMAL-DIGIT n SIZE			
without cache store	7,691	1,836	47
with cache store	17,305	4,132	106
32-BIT MACHINE SIZE (the 370/168, which has a cache)			
200-DECIMAL-DIGIT n SIZE	180,111	38,420	542
100-DECIMAL-DIGIT n SIZE	312,146	74,534	1,880
60-BIT MACHINE SIZE (the CDC 6400, which has no cache)			
200-DECIMAL-DIGIT n SIZE	71,218	15,192	214
100-DECIMAL-DIGIT n SIZE	104,168	24,876	627

4.4 ENCRYPTION TIME COMPARISON

An article on encryption time requirements for various encipherment methods on the 6400 was published by Friedman and Hoffman (FRIEDMAN 1974). They determined through experimentation the time requirements on the 6400 for encryption methods involving "exclusive-or'ing" the data with keys of various lengths.

By exclusive-or'ing a message with a key, one gets an encrypted message which is easily decrypted--by exclusive-or'ing the encrypted version with the same key. Exclusive-or'ing works by matching up the individual bits of themessage with the individual bits of the key and for a given pair, if they are both the same (both 0's or both 1's) the result is 0, otherwise 1. If the key is shorter than the message, the key gets used over as needed. As would be expected, the longer the key, the more secure the encryption is. If the key is as long as the message (and randomly generated), the scheme is theoretically uncrackable. Pseudo-random key generators provide keys of unlimited length that are effectively random. They do this by feeding the output of a pseudo-random number generator into a key generation algorithm.

Friedman and Hoffman came up with the following results for assembly langauge encryption programs:

one-word key	4.76 microseconds/word
long key (125 words)	8.25
double key (two long keys, equivalent to one	12.56

15,375 word key)

pseudo random key 20.05

The words refered to are 60 bits, as found on the 6400.

The public-key cryptosystem approach yields the following figures for encrypting 60 bits on the 6400, using an n size of 100 digits:

key size of 3: 577 microseconds/word

key size of $\log_2 n$: 2,411

key size of n: 96,247

Using an n size of 200 digits yields the following results:

key size of 3: 844 microseconds/word

key size of $\log_2 n$: 3,949

key size of n: 279,829

In comparison, the PKCS approach is quite a bit slower. It should be noted, however, that the pseudo random key method is the only one of the other methods which yields theoretical security comparable to PKCS.

5. SYSTEM STRUCTURES--COMPARING PKCS TO DES

To examine the tradeoffs between using PKCS and conventional encrytpion/decryption methods, we will discuss two common communications structures, a centrally linked structure and a fully-linked network structure. In discussing these, we will compare their use with two different encryption/decryption schemes, PKCS and the National Bureau of Standards' Data Encryption Standard (NBS' DES). DES is likely to become the commonly used encryption method in commercial installations. For each communications structure, we will consider the operation of encryption/decryption within the structure, the cost of implementing and using the method, and how effectively security is provided.

5.1 ONE LEVEL KEY SYSTEM

The simplest way to use DES for communications encryption is just to encrypt and decrypt using a common key which is changed only upon agreement. This means that the key will either never be changed or changed only at a mutually agreed time using a key that is agreed upon by the two parties in advance. The two major considerations are deciding how often to change the keys and how to communicate the new key in a secure fashion.

The reason there is concern with changing the keys at all is to limit exposure should a key be compromised. Security would then be recovered with the next key change. Of course, the new key must be initiated securely or the process is pointless. Additionally, the longer a key is in use, the greater the odds that it can be cracked.

Unfortunately, it turns out to be an expensive process to securely communicate keys between two parties. They cannot be transmitted over the data communications link since if one key is compromised, succeeding key changes will have no effect; it is the same as never changing the key. Telephone lines and mails are suspect from a security viewpoint. For most commercial applications this leaves couriers as the only secure means and only then if the courier is uncompromisable, a questionable assumption. Furthermore, using couriers is very expensive and time consuming; the time between key changes must be measured in days and if there are many connections it becomes organizationally infeasible.

This leaves a tradeoff between changing keys infrequently, incurring a correspondingly higher security risk, and accepting the expense of changing them often, assuming that a secure method of external communication exists. There is also the question of which is the bigger security risk--changing a key or not changing it.

5.2 TWO LEVEL KEY SYSTEM

One common method for minimizing this tradeoff between security and expense in key changes is to employ a two level key system. With this technique, data communications are encrypted using the secondary key, with the primary used solely for communicating new secondary keys. A key manager is employed at the communications link(s) to coordinate the key changes and keep track of the keys in use. This system structure may be illustrated as shown in Figure 1, where P_i refers to the primary key for the i^{th} user and S_i refers to the secondary key for the i^{th} user.

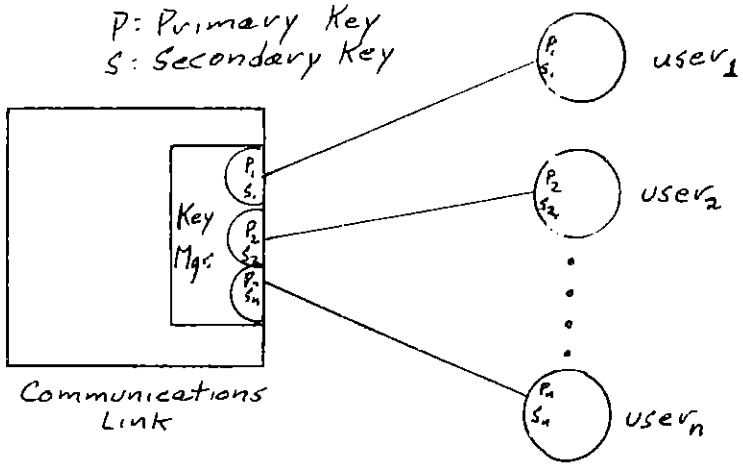

Figure 1--Two Level Key System

The primary key is set manually at each location and transmitted via courier. It is changed fairly infrequently, perhaps on the order of once a month. This primary key is used to encrypt the new secondary keys, which are generated by the key manager and transmitted over the communications line. The user's terminal or computer then automatically loads the new secondary key and communication continues using the new key. The secondary key may be changed frequently with little expense or risk.

The main advantage of this system is that the keys used to encrypt data transmissions can be changed frequently, automatically, and inexpensively. The main disadvantage is that if the primary key is compromised, security can be lost for a relatively long period of time, until the next primary key change. On the other hand, due to the infrequency with which it is changed, greater care may be taken in the change process. Another disadvantage is the extra hardware and/or software expense for the two-level key system.

5.3 COMPARISON OF DIFFERENT SYSTEM STRUCTURES

In this section we will compare two common communication system structures using a two level DES system and a PKCS system. Additionally, we will explore a hybrid two level system which uses DES as the secondary key system and PKCS as the primary key system.

The two communication structures we will examine are a centrally linked one with a small number of links and a network structure with interconnected links. These may be modelled as shown in Figures 2 and 3.

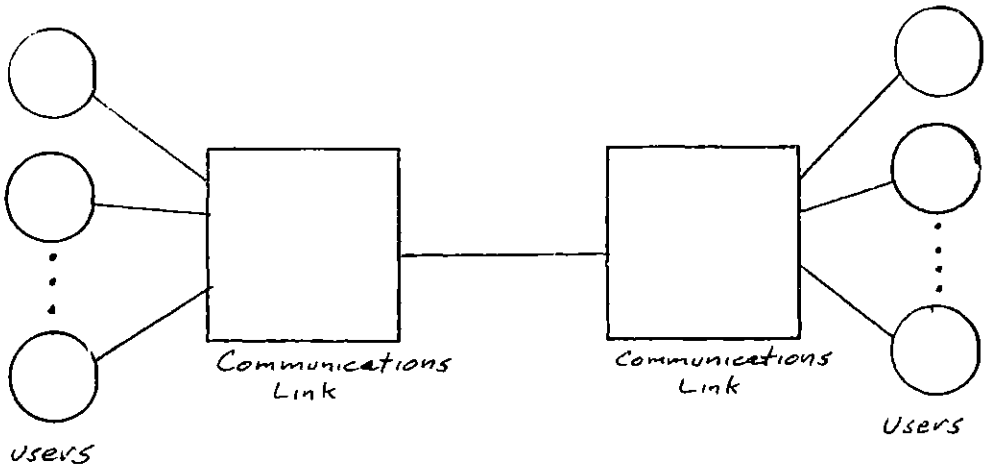

Figure 2--Centrally Linked Communications Structure

In each case we assume several or more terminals or users' computers directing their communications to the links, with these communications requiring encryption.

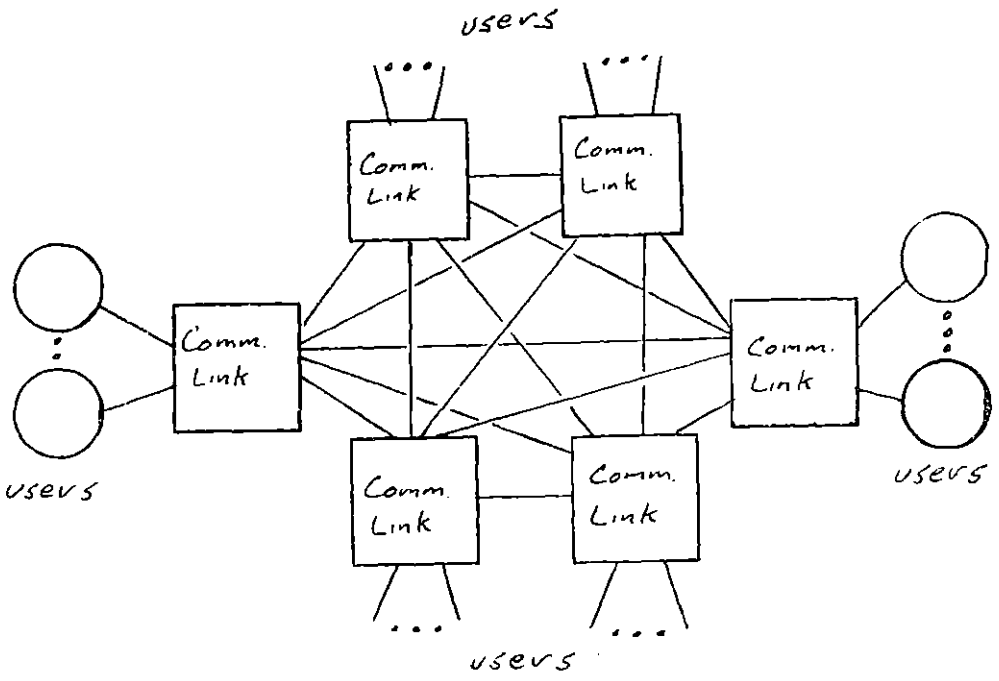

Figure 3--Network Communications Structure

In the case of a small number of central links. we will not consider the cost of encyrpting transmissions between links, as this cost for a small number of important lines can be considered neglible. This is perhaps defining the second case as one in which there are enough link interconnections to make this cost important. An example of the first type is the international communications systems used by major banks.

5.3.1 CENTRALLY LINKED SYSTEM EMPLOYING DES ENCRYPTION

OPERATION

As described above, the primary key is used to transmit the secondary key changes. The primary key change is a manual operation which is performed infrequently and is transmitted by courier. The secondary key is changed more often, up to a few

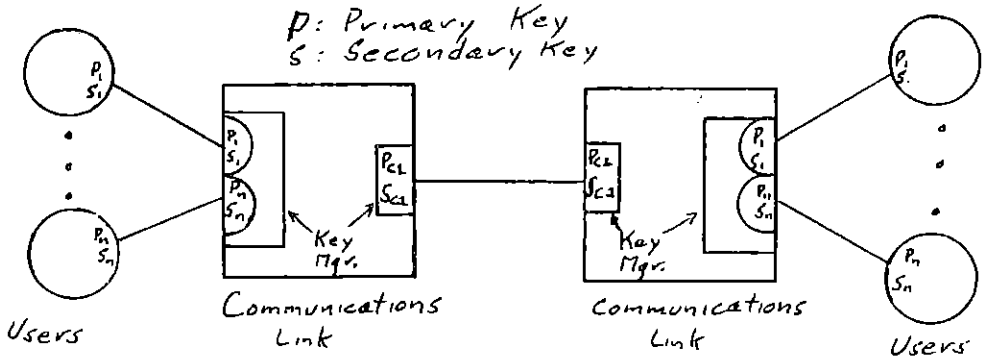

Figure 4--Centrally Linked Structure Employing DES

times a day, or more. Messages are encrypted only with the secondary key.

COST

There are two major cost elements--installing the hardware encryption/decryption boxes at both ends of all links and the courier service which is necessary to transmit and install primary key changes. The hardware DES devices run about $500 (e.g., a DES board from Rockwell). The courier service is recurring and expensive since it is necessary whenever a primary key is to be changed at any link.

SECURITY

With secure keys, the security is probably as good as anyone would want for commercial applications. We say probably because there is some controversy over whether the algorithm has a trap-door that the National Security Agency designed into it (DIFFIE 1977). This suspicion was aroused because the National Bureau of Standards, at NSA's request, refuses to reveal the design details--normally a standard and important

step in generating confidence in the security of a device or system. It is an axiom in computer security that a system's security should not depend on a potential intruder's ignorance of the system structure.

If the secondary key is compromised, security is lost until the next secondary key change, which occur fairly frequently. No obvious flaws in the security of the secondary keys are apparent.

If the primary key is compromised, security is lost until the next primary key change, which occur infrequently. In addition, the courier operation used for installing promary key changes is a serious security weakness since it depends on a human's not exposing the key, either through error or deliberately (having been bribed, for example).

5.3.2 CENTRALLY LINKED SYSTEM EMPLOYING PKCS

OPERATION

For both the user locations and the links, keys are generated on location and publicly broadcast. Users send messages by encrypting them with the link's encryption key. The link decrypts the message and then re-encrypts it with the receiver's encryption key. Keys may be changed as often as the owner likes (or the system, if key changes are under system control); generating and broadcasting the new key is all that is involved.

COST

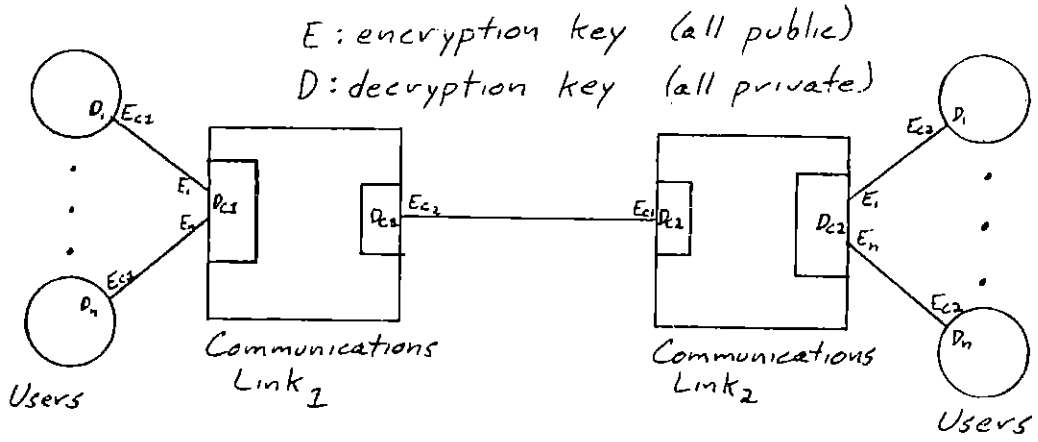

Figure 5--Centrally Linked Structure Employing PKCS

Either the hardware cost of installing encryption/decryption devices on both ends of all the communication lines must be absorbed, or a software emplementation must be used. As Sections 3 and 4 showed, both of these options are expensive, either in terms of hardware expense or computation time. Additionally, key generation and broadcasting produce system overhead.

There is, however, no expense for courier service or any similar human involvement as it is not necessary.

SECURITY

PKCS provide very high security. This is true from both a technical basis and on organizational one. Technically the system may be made arbitrarily secure through the key lengths. Organizationally, there is no human link in the security system, eliminating the most common, and often most severe, security weakness. Due to the fact

that there is no human involvement and the decryption key remains internal to the machine at the user's location, the security effectiveness approaches the theoretical security effectiveness of the encryption algorithm. This is very unusual in actual implementations, since the human involvement usually weakens considerably the actual level of security.

5.3.3 NETWORK STRUCTURE EMPLOYING DES

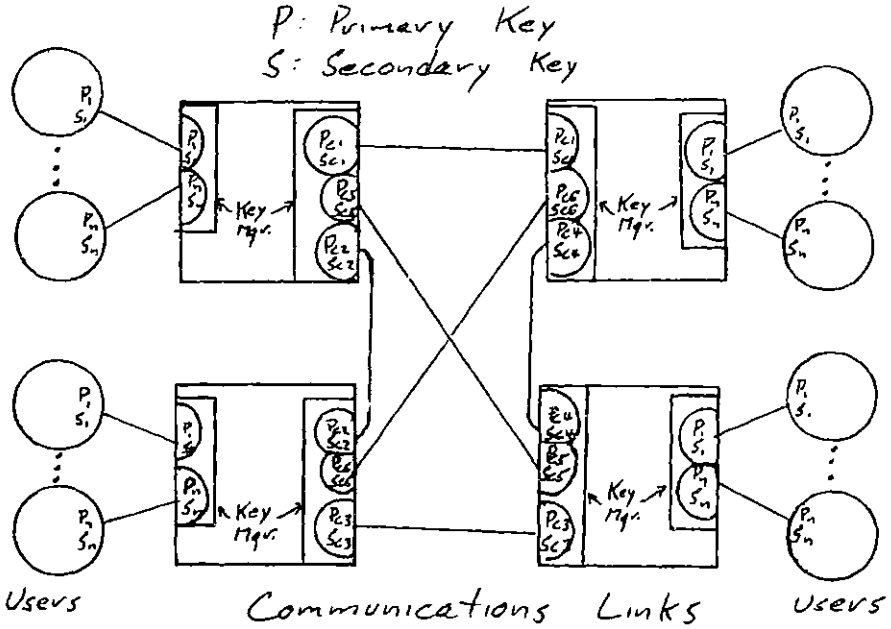

Figure 6--Network Structure Employing DES

OPERATION

For the terminals hanging off the links, operation is the same as it was for the centrally linked system. For the switches, though, separate keys are needed for each pair. Otherwise, if one were compromised, all would be compromised, and also one link would have the ability to read messages between other links.

With separate keys, manually set, courier sent primary keys are needed for

each pair. This comes to n(n-1) operations of this sort for n links. The problem becomes far more severe if the link structure is dropped and the system just consists of many interconnected user locations.

COST

Aside from the hardware investment in encryption boxes, the primary cost item is the n(n-1) courier operations required to effect primary key changes. In a system with 100 nodes in which the primary keys are to be changed once per month, this means that there needs to be 9900 courier sent, manually set primary key changes, a very expensive proposition (actually, only 4500 would be required if it is assumed that each pair will be generated at one end of a link it is to be used on). Actually, in practice it would not be this bad, as one courier could service all of the key changes at one link, however it does provide the flavor of the intractability of such a system.

SECURITY

Security becomes far more difficult. In the example with 100 links, at least 4500 key changes must be communicated and manually installed without security being compromised; obviously a difficult request. It is doubtful whether security could be reasonably assured with this type of system without great expense.

5.3.4 NETWORK STRUCTURE EMPLOYING PKCS

OPERATION

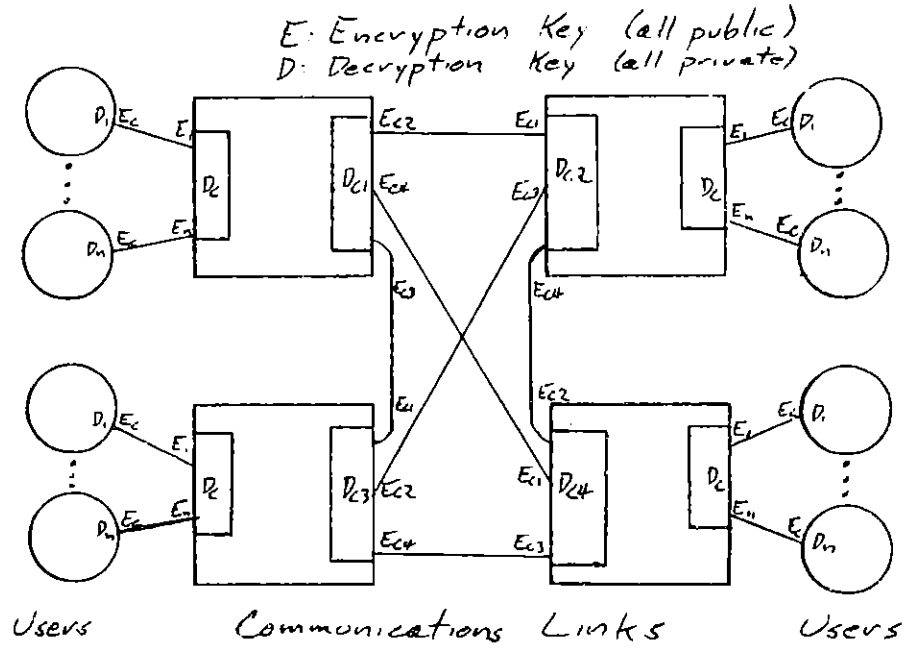

Figure 7--Network Structure Employing PKCS

Operationally, the network configuration would work very similarly to the centrally linked system. The major difference would be that when the links generate a key change they would need to broadcast it to all of the other links.

PKCS have a great advantage in this type of system. Primarily, this is because of the low overhead involved in the key change and maintenance process. It can be completely automated, involving only the usual communication system hardware and with no human involvement. Additionally, since a user's encryption key is good for all potential message senders, the number of keys to manage grows linearly with the number of links. When a separate pair of keys is needed for each link pair, the number of interactions grows with the square of the number of links (n**2-n).

COST

There is no major cost increase going to a network structure from a centrally linked structure. The encryption keys must be transmitted to the connected

links, but the operation is the same.

SECURITY

Security is as high as with the centrally linked structure.

5.4 HYBRID SYSTEM--PKCS AND DES

A promising technique for receiving the respective benefits of both DES and PKCS is implementing a hybrid two-level system utilizing both PKCS and DES.

Messages would be encrypted and decrypted using DES encryption devices. The keys for these boxes would be transmitted over the communications line encrypted by a software PKCS implementation. This would in effect be a two level system with PKCS constituting the primary key system and DES constituting the secondary key system.

The advantages of such a system would be that the speed and cost of a DES implementation would be available with the security of a PKCS implementation. Generating a new secondary key a few times a day would only involve encrypting a 56 bit message each time. This would go fairly quickly, providing negligible system overhead. Changing the primary key, which would be considerable more time consuming would only need to be done infrequently, perhaps on the order of once a month. At the same time, the hardware expense would be less that required for a straight DES system, as only the DES encryption/decryption boxes would be needed; the primary key controls would be unnecessary.

Meanwhile, all of the human involvement in transmitting key changes would be eliminated. This would be true for both a centrally linked system and a network structure. Security would then be very high, approaching that of a straight PKCS system

6. EXAMPLE SYSTEM--MONEY TRANSFER SYSTEMS AT COMMERCIAL BANKS

PKCS seem to be especially suitable for large-scale geographically diverse communication systems. An example of such systems is the type of international message switching and funds transfer systems used in large commercial banks.

We will examine the general workings of these systems as they are and then propose a new structure using the PKCS approach. In doing this, we hope to show through example the advantages and disadvantages of implementing PKCS. At the same time, we will attempt to show the potential conveniences PKCS can provide with the proper hardware equipment. As such, we will freely assume the availability of hardware as described in Section 3.3.2, Hardware Devices We Would Like to See.

6.1 PRESENT SYSTEM OPERATIONS

For background and as a basis for comparison, this section will describe the present systems.

6.1.1 SYSTEM PURPOSE

The systems are designed to serve as an international computer communicaitons system whose purpose is to transmit text and money-transfer messages between the banks main office, its branches, and its correspondents. Text messages may be any information to be transmitted, while money-transfer messages signal the transfer

of funds.

6.1.2 WHO IS LINKED

Typically, the main offices, all of the branches, correspondent banks, and many of the major customers. are linked into the systems.

6.1.3 SYSTEM DESCRIPTION

6.1.3.1 MESSAGE DESCRIPTION

There are about 30,000 messages per day, averaging about 1000 bytes of information each.

Each message contains a destination, text, amount, and a testword if it is a money-transfer message.

The testword is the sum of codes based on certain fields, including amount, a correspondent code number, and another code number that comes off of a sequence list. Each correspondent has his own list. These lists may either be reused or new lists may be distributed periodically.

Messages may be initiated by any user and directed to any user.

6.1.3.2 COMMUNICATIONS STRUCTURE

There are three major communications links, one each in New York London, and Tokyo. Each user has a terminal or computer attached to the appropriate link.

The communications lines are generally leased common carrier lines. Some of the smaller correspondents use dial-up lines.

6.1.3.3 HARDWARE COMPONENTS

The central links comprise two minicomputers each. One is used for message switching and the other for testword checking and handling incorrectly formatted messages.

6.1.3.4 SOFTWARE STRUCTURE

There are two major software functions, validity checking and message routing. Validity checking consists of checking for proper format and checking the testword against the message contents and the correspondent's sequence number list.

When a message comes in, if it is correctly formatted and it is not a funds transfer message, it is sent directly to the destination correspondent. Otherwise, it is directed to the second machine which performs the appropriate operation.

If it is a funds transfer message with the correct format, the system calculates the correct checksum and checks it against the message testword. If they

match, the message is sent on its way. If it doesn't match, a check is made to see if the problem is just one of sequence, i.e. an earlier message was delayed for some reason, causing the sequence numbers to appear out of order. If this is the problem, the message is held until the others come in. Otherwise the message is rejected and sent to an operator, as described below.

If the message is held up due to formatting problems, the system makes an effort to correct it automatically. If this is successful, the messgae is released. Otherwise the message is sent to a human operator at a CRT. There is a small group of these operators who attempt to correctly format these messages, perhaps telephoning the sender if necessary.

6.1.3.5 OPERATIONAL PROCEDURES

In addition to handling incorrect message formats and testwords, the system also often requires human involvement on the user end in setting up messages and testwords.

For the smaller users with only a terminal hooked into the system, the testword calculation must be done manually. This generally involves summing different codes for the date, the amount, the user identification number, the message sequence number, and possibly others. Once this is done, the testword is inserted into its proper place in the message and it is sent off. As this procedure consists of table lookups and arithmetic operations, all done manually, it is a likely point for errors to be generated. While these errors will be caught at the links, that generally requires human involvement,

which adds substantially to the cost.

As discussed above, when the messgae arrives at a link it is sent right through if it is formatted correctly and it is not a money transfer message. If it is a money transfer, it is sent to the verifying unit, where it is checked and released if verified. If the testword is incorrect or if the message is badly formatted, it is sent to the human clerks.

No human intervention is required on the receiving end.

It should be noted that although encryption is not presently used, two level DES systems are being planned for the near future for more than one major bank. Were these implemented, the corresponding operational procedures to coordinate key management as described in Section 5.2 would be necessary.

6.1.4 SECURITY EFFECTIVENESS

As these systems presently are (that is, without encryption), there is little in terms of direct security measures. The main feature is the testword procedure, but the main emphasis and effectiveness of that procedure is in ensuring system integrity, not security.

Security is ensured to the extent that someone who doesn't know the testword algotithm and doesn't have access to enough actual messages to figure it out, can't modify messages of generate unauthorized new ones. It is farily easy, however, to

figure out the code with enough messages to work with in a fairly short period of time, perhaps an afternoon for someone without training in cryptanalysis.

Unfortunately, these cases in which the system is protected from a security viewpoint are just the ones least likely to occur. If someone is in a position to modify messages or insert new messages, that person is almost certainly in a position to view legitimate messages. This includes office clerks, Western Union employees, and EE majors, not to mention members of the validity checking crew at the central links.

Again, though, it should be noted that the system was not designed with security in mind. One should consider it as the automation of a manual system; certainly the banks have never had whole buildings full of people checking signatures on checks that were being cleared. With the implementation of the two level DES system the communication system will be immediately far more secure. As long as the keys were uncompromised, transmissions would be safe against everyone but entrusted employees. Section 5.2 discusses the problems with keeping the keys uncompromised in that type of system.

6.2 PROPOSED PUBLIC-KEY CRYPTOSYSTEM APPLICATION

Given the purpose and basic functioning of these systems, we will now present a description of the same system incorporating the PKCS approach.

6.2.1 SYSTEM DESCRIPTION

6.2.1.1 MESSAGES

The messages would be unchanged except for the testwords. Since their only purpose now would be to ensure integrity, a clearer algorithm could be used which is easier to compute manually. Even the repitition of some fields, perhaps in a truncated form, might be adequate.

6.2.1.2 COMMUNICATIONS STRUCTURE

The communications facilities requirements and structure would likely be unchanged. One possible optional change, though, would be to switch from more expensive, more secure, lines to less expensive, less secure lines.

6.2.1.3 HARDWARE

The greatest tangible changes are in the hardware. The PKCS scheme is fully automated through the use of PKCS encryption/decryption boxes, smart terminals, and specially programmed central links.

The encryption/decryption boxes take the key as one input and the message as another. The same unit can be used for both encryption and decryption through the key that is supplied as input. All messages pass though these devices both coming and going.

The smart terminals have several functions. The most used is feeding the

appropriate key to the encryption/decryption boxes. With link-to-link encryption, this is easy--on incoming messages, that user's decryption key is used and on outgoing messages, the link's encryption key is used. If signatures are used the messages must be operated on twice, but no additional information is needed.

To implement signatures there is the choice to either have two encryption boxes in series or to cycle the messages through one box twice. If cycling the messages through one twice is chosen and many of the messages are not sensitive, the signature on request command, as discussed in Section 3.3.2.2.3, could be implemented to minimize the time consumed.

The last major function of the intelligent terminals is key generation and broadcast. Since this will be done reatively infrequently, it is likely that it will be done in software, although if inexpensive hardware becomes available, that would be an attractive alternative. The size of the key should be a parameter, set by the central links to allow for system-wide tradeoffs between speed and security. If the command to generate and broadcast keys is also given by the central links, this provides a very flexible and robust security system. It could easily, for example, set the standard key length to be relatively small for standard transmissions, and when an exceptionally high security message is expected, issue a generate key command with a long key length. This feature would be more useful, however, in a system in which encryption and decryption are done by software. In that case, shorter key lengths would produce substantial savings on all transmissions.

The central links have responsibility for decrypting all messages sent to it,

determining their destinations, and re-encrypting them appropriately. As it must do this for all messages coming and going, it must have either super-fast encryption/decryption devices or many of them.

As with the users' terminals, the links must also be able to generate keys and broadcast them over the communications lines.

An additional responsibility of the links is key management, such as it is in PKCS. This involves little more than keeping track of all the encryption keys, using them as necessary to encrypt messaes and decrypt signatures. ' A small additional responsibility in this area is signalling the user terminals and conputers when to generate and broadcast keys. As this is an important system parameter, control should porbably be kept centrally. In addition, this eliminates the necessity of having a continually running timer utility in the users' terminals.

6.2.1.4 SOFTWARE STRUCTURE

With the system described, the only software demands are key generation, if it is not implemented in hardware, supplying the encryption/decryption boxes with the proper key, and key recording.

The problem of proving which key was in use when, as discussed in Section 3.2.3, can be handled here by the central links by simply making a record of all keys and the time of use. The system overhead due to this would be very neglible. This will be a solution, however, only if the operator of the central links is trusted. In this case that

presumably will not be the case, since many independent banks and corporations will have access to the system (presumably they don't all trust Citibank that much). However that is no worse than the present system; it mainly means that the signature capability will be of limited value in this case for future verification.

Validity checking and destination determination can be performed unimpeded as the messages are unencoded during the time they are in the link machines (admittedly not optimal from a security viewpoint).

6.2.1.5 OPERATIONAL PROCEDURES

Operationally, the encryption system does not require any human involvement.

Typically, a message will be typed in at one of the user locations, encrypted by hardware automatically, sent over common carrier lines, decrypted automatically at the link, routed by the link, re-encrypted, sent out to the destination, where it is again decrypted.

One small complication is how the link is to know who a signed message is from if it comes on a dial-up line. This is easily accomplished, though, by preceding messages that are to be sent over dial-up lines with an extra block that has the name of the sender in an unencrypted form. When the link receives a message over a dial-up line, then, it either supplies a key of "1" to decrypt the first block, or, probably more difficult, by-passes the encryption/decryption box.

6.2.2 SECURITY

6.2.2.1 DESCRIPTION

With the PKCS approach, the system may be considered to have very high external security. That is, it would be very difficult for someone outside the bank and its users to compromise the system's security.

Internally, it has many of the some flaws that the existing systems have. While we could easily suggest measures to lessen or eliminate these threats, we don't see any which depend on the PKCS approach.

6.2.2.2 COST

The major cost involved with the system descibed is the initial investment in hardware encryption/decryption boxes and smart terminals.

Importantly, once this initial investment is made there is very little cost in running the system as a PKCS. Indeed, it seems that it would be easier and more convenient while at the same time providing substantially more security and the benefits of signature capability.

6.3 COMPARISON

The big advantage PKCS have over the existing systems is that it protects the system from being compromised via the communications lines, and it does this at little operational cost. Additional benefits are the availability of signatures for future verification and to prevent spoofing and the use of simpler integrity checks.

As noted, the present systems are very vulnerable to intruders who have access to the communications lines. With the use of the PKCS approach, these communications are secure to the extent that the encryption algorithm is unbreakable.

The increase in system integrity provided by PKCS should not be underestimated. The simplification of integrity checks should lead to fewer errors as well as being less of a nuisance for those who need to do this manually. The signature capability provides added insurance that the messages are authentic and to the extent that they provide positive records in the future, add confidence in the trustworthyness of the system (consider both selling the idea to correspondents and selling the system to other banks).

The major drawback to this system is the initial investment in hardware. This can be minimized by increasing the load on software; however that has its own, possibly unacceptable, increases in operating costs. Another drawback is that to the extent that it would not be merely automating an existing system, additional user education would be required

7. SUGGESTIONS FOR FUTURE WORK

The most pressing area in which additional work is needed is in exploring additional implementation possibilities. The more straightforward methods were analyzed here; hopefully, though, more economical approaches can be developed. Hardware implementations appear to have the most potential in this direction.

An alternative approach for exploring more economical implementations would be looking at different algorithms for implementing PKCS. Perhaps an algorithm could be found which is computationally much faster than that which was discussed here.

Further interesting areas of study are other applications of PKCS. The signature capability especially seems to have potential for being widely applicable. One application might be unforgeable files. One use for this would be payroll files. In this case employees' salaries would be encrypted with a private key and the decryption key would be public. DP operations would then be able to read the salaries to produce checks and reports, but only the possessor of the private key (perhaps head of personnel) could alter the file. In connection with this setup, one could envision a terminal which accepts the decryption key via a card with a magnetic strip.

Note that with the scheme just mentioned, the integrity of the file is ensured requardless of the security capabilities of the operating system or system hardware. This raises the question of how much of a computer security system could be implemented solely with PKCS capabilities. Read-only files could be effected by allowing only the decryption key to be public. Read-write file could be effected by making both

keys public, or not encyrpting at all. Publicizing neither key could keep the file competely private. Write-only files could be effected by allowing only the encryption key to be public. Additionally, a method of access control similar to capabilities could be effected by controlling whom the keys are given to. While this scheme obviously won't provide the the optimum security system, it might well be useful for sensitive applications running on machines not equipped to provide adequate security.

8. SUMMARY

An analysis of PKCS from an applications viewpoint has been presented. This has included cost and speed analyses for both software and hardware implementations, discussion of the capabilities and limitations of PKCS, and comparison with other methods.

This has been meant as an introduction to PKCS, rather than an exhaustive study. Suggestions for future work were also discussed.

REFERENCES

(Diffie 1976) Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, November 1976.

(Diffie 1977) Diffie, W. and M. Hellman, "Exhaustive Cryptanalysis of the NBS Data Encryption Standard", Computer, June 1977.

(Friedman 1974) Friedman, T. and L. Hoffman, "Execution-time Requirements for Programmed Encipherment Methods", Communications of the ACM, August 1974.

(Kohnfelder 1978) Kohnfelder, L., "Towards a Practical Public-Key Cryptosystem", Bachelor's Thesis, MIT Department of Electrical Engineering, June 1978.

(Knuth 1969) Knuth, D., The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, 1969.

(Rivest 1977) Rivest R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", MIT Laboratory for Computer Science, TM LCS/TM82, April 1977.