COMPUTER SYSTEMS ESTABLISHED, MAINTAINED, AND TRUSTED BY MUTUALLY SUSPICIOUS GROUPS

by

D. L. Chaum

Memorandum No. UCB/ERL M79/10

22 February 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups*

D. L. Chaum

ABSTRACT

A number of organizations who do not trust one another can build and repair a highly-secured computer system that they can all trust (if they can agree on a workable design). Banking provides an example of the need for these systems. Cryptographic techniques make such systems practical, by allowing stored and communicated data to be protected while only a small mechanism, called a vault, need be physically secured. Once a vault has been inspected and sealed, any attempt to open it will cause it to destroy its own information content, rendering the attack useless. A decision by a group of trustees can allow such a vault--or even a physically destroyed vault--to be re-established safely.

Networks of vaults, in which some active vaults are necessary to re-establish the network, have two advantages over single vault systems: (1) information that is no longer needed can be permanently destroyed, and (2) abuse of the trustees' power can be detected in advance. Each of some mutually suspicious groups can supply part of a vault, in such a way that each group need only trust its part in order to be able to trust the entire vault. Another approach to construction is based on public selection of a system's component parts at random from a large store of equivalent parts. The practicality and ramifications of the ideas presented are also considered.

Introduction

Concern over the trustworthiness of computer systems is growing as the use of computers becomes more pervasive. It is not enough that the organization maintaining a computer system trusts it; many individuals and organizations may need to trust a particular computer system.

For example, consider a computer that maintains the checking account balances of a bank. The bank is concerned, among other things, about possible loss of balance records. The Federal Reserve Bank must know the total of these balances, to ensure that the legally required percentage of the balances is on deposit with it. The Internal Revenue Service requires the ability to check the balance of an individual's account. Individuals, or a consumer organization acting on their behalf, may wish to ensure that disclosures are made known to those involved, and that inquiries can never be made on information that is more than a few years old.

The thesis of this paper is that such widely-trusted computer systems can be provided, if a workable design is agreed on. The cryptographic techniques which form the basis of the approach are introduced in the first section. They make such systems practical by reducing the mechanism upon which reliability and security depend. This

* This work was partially supported by the National Science Foundation under NSF Grant MCS75-23739.
Author's address: Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, CA 94720. (415) 642-1024.

mechanism--the processor and its high speed store--will be called a *vault*. A vault will be physically secured by shielding it within a small safe like container.

Begining in the second section, the paper departs substantially from the literature, and presents techniques that allow mutually suspicious groups to maintain a vault. In each of the third and fourth sections, an advantage of systems comprised of multiple vaults is presented. The fifth and final section removes the assumptions of perfect cryptographic and physical security, and also describes practical approaches to constructing and certifying these systems.

1. Cryptography

Information is encrypted to allow it to pass safely through a hostile environment. Traditionally, concern has centered on providing the secrecy of communications. Consequently, cryptographic techniques were devised to make it very difficult (in some cases impossible) to transform encrypted information back to its unencrypted form without possession of a secret piece of information, called a *key*. Two correspondents who were the sole possessors of a key could use it to maintain the secrecy of their correspondences. Note that the cryptographic algorithms are assumed to be public knowledge; only the key need be kept secret.

Ultimately, all cryptographic algorithms can be thought of as transforming symbols into other symbols. With a Captain Midnight decoder badge, the badge is the key, and letters are mapped into other letters. The un-breakable Vernam cipher maps only single bits into other bits, by adding each bit modulo two with a different key bit [Kahn 67]. On the other extreme, block cryptographic algorithms map large strings of bits, called blocks, into other blocks. The National Data Encryption Standard, for example, maps 64 bit blocks into 64 bit blocks, using a 56 bit key [NBS 77]. Many blocks can be "chained" together during encryption, effectively forming a single large block [Feistel 70].

1.1. Application of Cryptography

We will be interested in block schemes, like the Data Encryption Standard, which make it very difficult to modify part of an encrypted block of information without causing drastic changes to the entire decrypted block. In such systems, a large serial number can be added to a block before encryption. Its presence after decryption indicates that the block has not been altered. Furthermore, it becomes extremely difficult for someone without a key to create a block that will contain a desired serial number when it is decrypted by a keyholder.

Two communicants with a common key can converse using encrypted blocks of data, checking the serial number of each received block to ensure that it has arrived in the proper sequence, and to ensure that it has not been altered [Feistel, Notz and Smith 75]. One communicant might be a vault, and the other might be a terminal with a cryptographic capability. In a similar way, encrypted blocks of data stored in memory devices outside the vault are checked on their return to verify that the correct blocks were returned unaltered.

All information leaving, or returning to a vault is in encrypted form; the unencrypted form is only present within the vault, or after being decrypted by communicants. Thus, data being stored or communicated is secured against tampering and eavesdropping. However, it still must be secured against obstruction or destruction. To these ends, stored information might be duplicated at remote sites, and redundant or broadcast style communication channels used. Because the material is safely encrypted, the proliferation of copies poses no threat to security.

1.2. Establishing a System

There is nothing secret about a vault until it is sealed within its protective shielding. An unsealed vault can, therefore, be freely inspected by any interested party. As we shall see in section five, this is a prerequisite to certification of the vault's mechanism by mutually suspicious certifiers. Assume, for now, that mutually suspicious groups can know that a vault operates correctly, as a result of some certification procedure.

Such certifiable vaults can be built with the use of some relatively new cryptographic techniques. Those considered so far have the unfortunate property that a common key must be distributed to the communicants, while it is kept secret from everyone else. In contrast, consider a fundamentally different sort of cryptographic algorithm independently proposed by Diffie and Hellman [76a], and Merkle [78]. To use these algorithms, each participant creates a *private key,* that is never revealed to anyone else. A corresponding *public key* is made known to everyone. We will be concerned with public key cryptographic algorithms (like that of Rivest, Shamir and Adleman [78]) where the two keys are inverses of one another, in the sense that that a block encrypted with one can be decrypted only with the other.

A message is encrypted with the recipient's public key before being sent. Only the intended recipient can decrypt the received message, because the corresponding private key must be used to decrypt it. (A key-sized random serial number should be included in the message, because otherwise a guessed message would be verified if it proved identical to the original when it too was encrypted with the public key.) Someone signs a message by encrypting it with their private key. If a serial number of all zeros is included in the message before it is signed, its presence after decryption with the corresponding public key would verify the signature.

When the vault is sealed, a suitable public key and its inverse private key are chosen by a mechanism within the vault's shielding, using a physically random process [Knuth 69]. The public key is then displayed outside the vault, on a special device certified for this purpose. As far as the world outside the vault is concerned, the possessor of the vault's private key is the vault: it can read confidential messages sent to the vault, and it can make the vault's signature.

2. What if Something Goes Wrong?

If a vault were totally destroyed, computation would be safely halted--no secret information would be revealed, and the vault would not have taken any improper action. Other conditions might require an equally safe halt to computation. If an alarm device detects an attempt to penetrate the vault's shielding, or a fail-safe mechanism determines that the vault's contents can no longer be counted on to operate correctly, then the information stored in the vault, including the vault's private key, must be erased.

This information will be encrypted in a special way, and saved outside the vault, so that a safe recovery can be provided. The encryption of the vault's contents, which includes its private key, is called a *checkpoint,* and is detailed below. At suitable intervals, checkpoints are formed, and then stored outside the vault. In some cases, there may be time to issue un-scheduled checkpoints before an emergency requires the vault's contents to be erased.

The primary consideration behind the design of an encryption method for checkpoints is that there exists a means to decrypt them, but only at the appropriate time and place. The decision that some newly sealed vault can, and should, be given the ability to decrypt a checkpoint is necessarily a human one. Assume, for now, that the decision is to be made by unanimous consent of a set of *trustees.* A checkpoint is successively re-encrypted with each of a set of keys, one key for each trustee. Conventional as opposed to public key cryptography will be used for this. The keys used to encrypt the checkpoint are called *partial keys,* as they must all be obtained to decrypt the checkpoint. The partial keys are randomly generated within the vault.

Public key cryptography will be used to distribute the partial keys to the trustees in a secure manner. Before it is sealed, the vault is supplied with a public key issued by each trustee. Then, the vault ensures the confidentiality of the the partial key it sends each trustee by encrypting it with the trustee's public key. Each trustee now has two keys to keep secret: a private key used to decrypt messages received, and a partial key that will be used in connection with decrypting checkpoints.

2.1. Restarts

A *restart* is the process by which a freshly sealed vault resumes the computation whose state has been saved in a checkpoint. After a replacement vault is certified and sealed, it forms a *temporary* public key and its inverse private key from a random seed, and then displays the temporary public key, as the permanent public key was displayed in the original start-up. Then the restarting vault receives partial keys from the trustees. A trustee provides the secrecy of its partial key while it is in transit to the vault by encrypting it with the displayed temporary public key.

Having received and decrypted the partial keys, the computation within the replacement vault decrypts the checkpoint. It then bootstraps itself into the state saved in the checkpoint. Thus, the original public key found in the checkpoint is reinstated, and the computation withing the replacing vault becomes an exact copy of the original computation. The restarted vault could be safely brought back up to date by re-playing all the messages sent it since the checkpoint was made.

2.2. More Flexible Decision Making

We have assumed that a consensus of the trustees is required for a restart, but a more flexible arrangement may be more useful in practice. One approach would be to create multiple copies of a checkpoint, each encrypted with a different subset of the partial keys, so that a consensus of the trustees holding the partial keys in any one of the subsets could authorize a restart. For example, the subsets that would be used to allow a simple majority of the trustees to cause a restart are all the distinct subsets of the partial keys that contain just enough partial keys to constitute a simple majority.

Forming many multiply encrypted checkpoints is clumsy at best, and is undesirable because it should be possible to prepare checkpoints quickly in an emergency. It is possible to achieve the same effect, however, while encrypting each checkpoint only once--with a single key. Copies of this key are encrypted with the subsets of the partial keys, just as the copies of the checkpoint itself were encrypted in the previous scheme. These multiply encrypted copies of the key used to encrypt the checkpoint are generated when the vault is originally established.* They can only be used by a restarting vault, once it has received a valid subset of partial keys.

3. Multiple Vaults

Up until now we have been concerned with the operation of a single vault. In this section, systems that use multiple active vaults are shown to have a major advantage over single vault systems. They can make it impossible for anyone, including the trustees, to reconstruct obsolete information.

We have seen how a disabled vault can be replaced by a freshly sealed vault, during a restart that uses a checkpoint issued by a disabled vault. The new vault is identified by the same permanent public key and will embody the same computation as its disabled predecessor. It is possible that a succession of vaults will be formed at a particular site, each restarted from a checkpoint issued by the preceding vault, and all embodying the same computation. It will often be useful to refer to this computation itself, without regard to a particular active vault. The computation will be called a *node*.

This section is based on systems which involve multiple nodes. The node at a particular site will be identified by its unique permanent public key. It will be convenient to refer to a set of nodes collectively as a *network*. When things are running smoothly, the nodes of a network will each have a currently active vault; when a node's active vault is forced to destroy its information content, however, the node will lack an

* Provision for a majority vote of 24 trustees would require 2,496,144 copies of the key. These could be generated by well known algorithms and placed, for example, on a single magnetic tape. This approach becomes impractical with many more trustees. A cryptographic analogue to the Electoral College, in which intermediate keys serve the purpose of electors, could be used for much larger numbers of trustees.

active vault until it is restarted.

3.1. Destruction of Information

In a single vault system, the partial keys held by the trustees will always be sufficient to decrypt any previous checkpoint. Thus, the trustees will have access to all information, no matter how old the information is. In contrast, information that is no longer needed in a system with multiple active vaults (i.e. a network) can be made permanently inaccessible by destroying the keys needed to decrypt it. In order to limit access to obsolete checkpoints, at least some of the partial keys must be destroyed. The network will be able to ensure that sufficient partial keys are destroyed, because it alone will maintain some critical partial keys.

When a node forms its partial keys, it does so in such a way that one additional partial key is always necessary to decrypt checkpoints. This partial key is distributed to the other nodes of the network, and not to the human trustees. When the current vault of a node dies and the node must be restarted, the human trustees will provide their partial keys as before, but the restart can not continue until the missing partial key is supplied by one of the other nodes. This node might require a signed request from each human trustee of the restarting node before it releases the partial key it holds. These requests would include the temporary public key displayed during the restart, so that the vault trustee can send its partial key directly to the restarting vault.

Each node of the network might have partial keys sufficient to allow it to assist the human trustees of any other node in need of restart. Then, even if every active vault in the network but one were destroyed, the surviving vault and the human trustees of the other nodes could rebuild the network. If every node were in need of a restart at once, however, the network would be permanently disabled. This possibility could be made acceptably remote by the inclusion of at least some fault-tolerant nodes, possibly at highly secured locations.

Periodically, say once a year, all keys in the network that could be used to decrypt information that is no longer needed, including the permanent private keys, would be changed. Each node will issue a new public key, signed with its old private key. New partial keys, however, need only be issued to the vault trustees; the partial keys issued to the human trustees need not be changed. Checkpoints of previous years could no longer be decrypted, because this would require partial keys that were erased once the entire key change had been completed successfully.

4. Further Limits on the Trustees' Power

In the previous section, we have been able to force the trustees to request certain partial keys of the network in order to obtain enough partial keys to decrypt a checkpoint. We have seen that if the network changes the keys used to form checkpoints, and the partial keys it releases, it can keep obsolete checkpoints from being decrypted. This section will be devoted to an approach to allowing the network to keep a comprehensive record of the partial keys it releases. Such a record is very useful because it can ensure that only certified vaults have decrypted checkpoints, and that they have done so only during certified restarts.

One possible approach to providing such a record is presented in the following subsection. The next subsection augments this approach to cover the possibility of stolen vaults. The utility of mandatory cooling-off periods for restarts is explored in the third subsection, and the fourth and final subsection of this section looks at the effort required to subvert the record maintained by a network.

4.1. Rosters

A network must issue its partial keys to a restarting vault on the trustees' request, but the network itself will maintain a record of the recipients of all the partial keys it has released. Since the only legitimate recipient of these partial keys is a restart, the recipient will be identified by its temporary key. Periodically, say once a day, the network will be able to issue a *roster* of all the temporary keys with which partial keys have been released.

If each node has sufficient partial keys to allow the trustees to restart any other node, then accurate rosters can not be guaranteed in general. To see this, consider the following scenario: the trustees usurp a node's current vault and deny it contact with the rest of the network, so that it thinks it is the only node not in need of restart. Then, they request it to supply its partial keys to fictitious restarts, whose temporary keys they have fabricated. Once it has divulged the partial keys, they destroy it, along with any record it tried to maintain of the restarts; so that no subsequent roster can contain the bogus temporary public keys.

One solution is to appoint a particular highly protected node as the giver of partial keys and issuer of rosters. If it were destroyed, rosters would no longer be issued, and restarts would no longer be possible. Instead of a single node, a fixed subset of nodes might be used. We will describe a scheme in which only a majority of the nodes of such a subset will be required to distribute partial keys and to issue rosters.*

For simplicity and concreteness we consider the case in which the subset comprises the entire network. This implies that at least a majority of the nodes of the network must be active at all times, otherwise restarts would no longer be possible. Partial keys created with the techniques that allowed arbitrary subsets of partial keys to be sufficient will be used in such a way that only a simple majority of the vault trustees, along with a consensus of the human trustees, will be required for a restart. Each node maintains a *list* of the temporary keys that it has used to transmit partial keys. Signed and dated copies of a node's list are made public daily. A collection of such lists, made up of lists from a majority of the nodes, will be a roster.

The temporary keys of all previous restarts must be represented in every succeeding roster. To see this, notice that there is at least one node in any succeeding roster that has participated in any previous restart, since the majority of nodes whos lists are represented in the roster must overlap in at least one node with the majority that allowed a restart. The list of a restarted node must be brought up to date to reflect any partial keys released after the creation of the checkpoint used in the restart. One need not rely on a restarted node's having been brought back up to date (by re-playing messages sent it, as described earlier) to ensure that its list is up to date; the same effect can be achieved by requiring a restarted vault to update its list from a roster that was issued after the restart.

The system could be subverted if the trustees were able to request enough partial keys to allow them to fabricate rosters. Such fabrication will be prevented by having vaults sign lists with their temporary private key as well as their permanent private key. This is a solution because the temporary private key will never leave the vault--even in a checkpoint--and will therefore be unavailable to the trustees.

4.2. Phantoms

The previous subsection addressed the problem of maintaining a complete roster in spite of efforts which might isolate and destroy a node's current vault. An additional threat is that a node's current vault might be kidnapped, and then kept alive. The kidnapping might even be undetected if a mock vault, which has presumably destroyed its own information content, is substituted for the kidnaped vault. Once the node of the kidnapped vault is brought back to life by the replacement of the kidnaped vault with a new vault, the kidnaped vault becomes a *phantom*. Phantoms can cause problems because they essentially split a node into two parts (the replacement and the phantom), each of whose lists may be incomplete with respect to restarts allowed after the split. We will first show how lists contributed by phantoms can be detected in rosters, and then how such detection can be used to keep phantoms from releasing their partial keys.

Phantoms' lists will be revealed by including additional information in each list entry. Each temporary key in a list will be

*A slight generalization of the argument in the preceding paragraph shows why less than a majority is insufficient in a homogeneous subset. A decision not requiring unanimous consent of a subset (like majority voting) allows a subset that has lost some members, but not enough to stop it from making decisions, to change its members and even its size.

augmented by both the date on which it was passed, and the permanent public key that identifies the node to which it was passed. Consider the set of vaults that contribute lists to a particular roster, where one or more of these vaults are phantoms. One of the phantoms became a phantom first, when a majority of nodes alowed its replacement.* At least one of the nodes of this majority must have contributed to the roster under consideration, because rosters contain lists from a majority of nodes, and pairs of majorities overlap. This node's list will contain an entry with the same permanent public key as the phantom, but with a different temporary key. The phantom will be revealed because this entry will be preceded by another one in the roster with an earlier date, and the phantom's temporary key.

Each vault will be required to check a current roster to make sure that it has not become a phantom, before it issues any partial keys. One way to provide current rosters might be to issue lists at the begining of each day. In a single day, however, a vault might give out a partial key based on its inspection of the day's roster, and the vault might be replaced (and thus become a phantom) based on inspection of the same roster by other nodes.

A solution is to require that requests for partial keys be made at least one day in advance. When a node receives a request for a restart, it adds an entry for it to its list. This entry contains the date on which the partial key will actually be released. A vault can now release partial keys on a particular day, in confidence that it will not become a phantom on that day, if that day's roster does not contain any proposals for the vault's own replacement.

4.3. Cooling-off Periods

An interesting ramification of these post-dated entries is that a cooling-off period is provided, during which those who are dissatisfied with the certification of a proposed restart might have some recourse. Provision might be made so that forces outside the system could cause the trustees to request the network to abort a proposed restart, for which the partial keys had not yet been released.

*The argument easily generalizes to the case when multiple nodes became phantoms at once. All the participants in the roster could not have become phantoms at once because no majority could have existed to allow their replacement.

A single restart might involve two non-overlapping cooling-off periods. In the first, the certification process would be announced by list entries with the appropriate permanent public key, the end date of this cooling off period, and a blank entry for the temporary public key. The second period is used to ensure that the trustees correctly transmit the displayed temporary key to the network. When the trustees provide a temporary public key, it will replace the previously blank part of the list entries, and their end dates will be advanced appropriately. Partial keys will be released only on or after the end date of the second period, and only if the restart has not been aborted.

4.4. Corruption

We have been assuming that once a vault is sealed, the secret keys it contains can not be discovered. If a sealed vault has somehow been opened without triggering the alarm, we say that it has been *corrupted.* Here, we will determine how many vaults must be corrupted to allow the checkpoints of every node in the network to be secretly decrypted. Without the partial keys of the trustees, every node would have to be corrupted. If the trustees participate (and it is easier to destroy a vault than to corrupt one), then the easiest way to obtain sufficient keys to decrypt the checkpoints of every node, is to corrupt enough nodes so that the rest of the network is divided in to two parts--each part just big enough to form a majority when combined with the corrupted nodes.

The two parts are isolated from each other so that the corrupted nodes can deceive each part into believing that the corrupted nodes and it are the remaining majority. Then, the trustees destroy one part after it supplies the partial keys required to restart the other part. These partial keys, along with those obtained from the corrupted nodes, are sufficient to allow the trustees to decrypt the checkpoints of the surviving part. These checkpoints will in turn provide sufficient partial keys (when combined with those of the corrupted nodes) to decrypt the checkpoints of the destroyed part. The surviving part will rebuild the network and replace the corrupted nodes, leaving a suspiciously large number of restarts as the only trace of the networks complete subversion.

With the simple majority scheme considered so far, the number of nodes that must be corrupted is one or two, depending on whether the network has an odd or an even number of nodes, respectively. The number of nodes that must be corrupted can be increased, at the expense of the number of nodes whos simultaneous death would disable the network. The number of nodes above a simple majority that is required for restarts, is the same as the number additional nodes that must be corrupted to subvert a network.

5. Real World Considerations

We have made two assumptions about vaults. Except for the subsection on corruption, we have assumed that the security provided by the cryptographic techniques, the vault and its alarm are invincible. Secondly, we have assumed that mutually suspicious groups can know that a vault operates correctly, as a result of some certification procedure. We will consider the real world approximations to each of these assumptions, in turn.

Security is a relative thing. Bank vaults and their burglar alarms, for example, seem to provide a sufficient probability of detection relative to the potential pay-off. Similarly, the data encryption standard, although implemented by a single chip, may require a million chips to break [Diffie and Hellman 77], and would thus be an adequate deterrent for some applications. Other cryptographic algorithms may be nearly as easy to implement, but might require more computational power than could possibly be amassed, assuming that certain aspects of our present understanding of physics remain the same [Diffie and Hellman 76b].

5.1. Certification

Absolute certainty about the physical world, however, is unobtainable in general. Effort must be expended to increase certainty that a particular system has some desired properties. Today, organizations spend considerable sums to periodically audit the state of their computer systems. The techniques presented in this paper do not have many of the inherent and practical limitations of audits. Before a vault is sealed, however, it must still be certified by people, whom others will have to trust.

The certifiers of a vault must be convinced of two things: (1) that the plan for the vault has the properties the vault will be certified to have, and (2) that the vault conforms to the plan. The first is beyond the scope of this paper (but see [De Millo, Lipton and Perlis 76]). The second will occupy the remainder of the paper. Since much of the plan is information that the vault is required to use (e.g. programs, trustees' public keys, and a roster of a particular date) it need only be signed by a sealed vault that is otherwise certified.

There are a variety of approaches to the remainder of the second problem--determining whether a physical device conforms to the plan--each yielding some degree of trust for a given technology. Since different technologies may be used in a particular system (e.g. printed circuit boards, and integrated circuits), and different parts may be constructed by different groups, a number of different techniques may be combined for a particular system.

One approach to certifying a device is simply to inspect it. (This might be adequate to determine the interconnections provided by a printed circuit board, e.g) Higher degrees of certainty and some technologies may require that the manufacturing process be observed as well. If a device is publicly chosen at random from a large collection of identical devices (integrated circuits, e.g.), then interested parties could choose additional devices and destructively test them.

5.2. Multiple Constructors

There is an alternative approach in which each certifying organization can produce a subset of the *modules* which comprise a system. The modules are combined so that if the modules supplied by any one organization conform to the plan, the whole system will behave as if each module conformed to the plan. This approach differs from the previous techniques in that a certifier can render the vault inoperative by providing a malicious module.

One way of combining a set of random number generator modules, each supplied by a different certifier, is by adding their outputs bit-wise modulo two. Redundant isolation modules for the power and input/output lines, which pass through the vault's shielding, would be arranged serially within the vault. Multiple alarm modules within the vault, possibly of different types, would make for a more secure vault. Reliability, in the form of immunity to false alarms, can be traded for security by a mechanism that requires more than one alarm be tripped before the memory content is erased.

Each member of a set of redundant processor modules must have access to all of the vault's inputs. If the output of one particular processor module is used as the output for the entire vault, the other processors must be able to compare their output to its output, and have time to stop the output on its way through the isolation devices, before it leaves the vault's shielding. If, instead, the outputs of the processors were routed through a voting device, the system could be made more reliable at the expense of the security lost by allowing one or more processors to be ignored.

If memory devices which will hold keys and other important information are supplied by mutually suspicious certifiers, each certifier may have to be concerned that some device supplied by another certifier will improperly retain this information in an emergency. This problem can be solved if thermal pyrotechnic devices, like those employed for similar purposes by the military, are placed within the vault. Each certifier might then supply a pyrotechnic module.

When the modules are installed in the vault, it should be possible to certify their interconnections by inspection. Certifiers' jobs would not be completed until they had witnessed the sealing of the vault and the display of the public key.

Summary and Conclusion

The techniques described here provide secure computer systems that can be trusted by mutually suspicious groups. To the extent that such systems prove practical, those providing information to, or relying on the output of a computer system, will also insist on being able to trust it.

Acknowledgements

I would like to express my gratitude to the following people who have carefully read and commented on various drafts of this paper: Mike Clancy, Doug Cooper, Bob Fabry, Richard Fateman, Chip Martel, Bernard Mont-Reynaud, Annie Moose, Carlo Sequin, Michael Turner, Stu Wecker, and Tim Winkler.

References

De Millo, R.A., Lipton, R.J. and Perlis, A.J., "Social Processes and Proofs of Theorems," Report, Georgia Institute of Technology, Atlanta, Nov. 1976.

Diffie, W. and Hellman, M.E., "Multiuser Cryptographic Techniques," Nat. Comp. Conf. 1976a, pp. 109-112.

Diffie, W. and Hellman, M.E., "New Directions in Cryptography," IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976b, pp. 644-654.

Diffie, W. and Hellman, M.E., "Exhaustive Cryptanalysis of the N.B.S. Data Encryption Standard," IEEE Computer, vol. 10, no. 6, June 1977, pp. 74-84.

Feistel, H., "Cryptographic Coding for Data-Bank Privacy," IBM research report RC-2827, Yorktown Heights, N.Y., March 1970.

Feistel, H., Notz, W. and Smith, J., "Some Cryptographic Techniques for Machine-to-Machine Data Communications," Proceedings of IEEE, vol. 63, no. 11, Nov. 1975, pp. 1545-1554.

Kahn, D., "The Code Breakers, The Story of Secret Writing," Macmillan Co., 1967, p. 395.

Knuth, D., "The Art of Computer Programming, Vol. 2, Semi-Numerical Algorithms," Addison-Wesley, Reading, Mass., 1969, pp. 2-3.

Merkle, R.C., "Secure Communications over Insecure Channels," Comm. ACM, vol. 21, no. 4, April 1978, pp. 294-299.

National Bureau of Standards, "Data Encryption Standard," F.I.P.S. Pub. 46, Jan. 1977.

Rivest, R., Shamir, A. and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Comm. ACM, vol. 21, no. 2, Feb. 1978, pp. 120-126.