Broadcasting Classified Information[*]

Hans-Georg Stork & Wolffried Stucky
(Institut für angewandte Informatik und formale Beschreibungsverfahren
der Universität Karlsruhe)

1 Introduction

The progress in communication satellite technology has made it feasible to distribute data and textual information via broadcast at relatively low costs to many receivers simultaneously. The technical equipment necessary for setting up satellite links is now fairly standard and can well be afforded by organizations who have an economically motivated interest in obtaining fast and up-to-date information about the markets they are depending on. Trading for instance with bonds, stocks, foreign currencies or commodities will only be successful if all decisions can be based on correct and timely information.

In the present paper we envisage the situation of an information provider who regularly supplies data items to his customers on the basis of individual contracts. He would certainly not be willing to entrust valuable data to communication channels that are – at least in principle – open to eavesdropping by everybody if these data could not be suitably protected. He must be sure that nobody except his customers has access to the broadcast information.

Moreover, the customers should be allowed to read only those data items or messages which they are willing to pay for. All other information should be illegible to them.

We show how these requirements can be met on broadcast networks. In addition, the scheme we are proposing is flexible enough to cope with frequent changes of existing contracts and with changes of the total amount of information a provider offers. Clearly, this property is of importance in a commercial environment.

It goes without saying that cryptographic methods must be at the heart of the matter. We shall in fact employ both classical methods and modern public key cryptosystems.

The requirements set out above imply that the hardware as well as the software of the receiving terminals must be immune against malevolent or negligent forgery. We take care that this immunity will be fully guaranteed.

The presentation is split up in two parts. In section 2 a simple model of the situation will be developed and the basic problems and requirements will

[*] IFIP 1989, San Fancisco

be stated in terms of this model. Solutions will be outlined in an abstract fashion. Section 3 briefly describes the software that implements the model and the proposed solutions.

2 The Model

Our "real world" consists of a central system run by some information provider and a set of remote terminals owned by information consumers (or subscribers). In general, data are transmitted to the terminals via satellite links, with the uplink going from the central system to the satellite and the downlinks connecting the satellite to the terminals. However, we assume that it is also possible for a terminal to establish a two-way end-to-end connection with the broadcast center via some (public or private) terrestrial network for instance.

2.1 Classes and Contracts

The crucial concept underlying our model is that of an information class. We take it for granted that the set of data items being offered at a given time can be partitioned into a finite number of classes each of which is uniquely identifiable. Let $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be the set of class identifiers.

A contract is simply a subset of I. Contracts will be denoted by $c \subseteq I$. There is a one-to-one correspondence between terminals and contracts.

It should be noted that neither I nor a given contract c nor the set of all contracts must be considered fixed. Rather, new information classes may be introduced and existing ones may be removed at any time. Also, a contract may be changed by excluding or including existing classes. Finally, new contracts may be solicited and existing ones may be cancelled. Obviously, performing these operations is subject to certain conditions. These conditions however, are not relevant for the subsequent expositions.

A specific terminal should be able to filter out of the incoming data stream those data items whose classes belong to its current contract. Hence each data item must be accompanied by its class identifier. Let d be some data item of class $i \in I$. The pair $e = (i, d)$ will be called an information element.

It follows that the data stream may be looked upon as a sequence of information elements e. Let t be a terminal and let c_t be its contract. The fundamental requirement that a customer owning t be allowed to read exactly those data items whose classes are contained in c_t now translates into the following statement:

"t intelligibly displays data d of an information element $e = (i, d)$ if and only if $i \in c_t$."

The "only-if" part of this statement may be expressed slightly more succinctly:

"If $i \notin c_t$ then t has no way to render d readable."

The basic strategy to make these statements true is rather straightforward: Any d wrapped up in some information element $e = (i, d)$ will be a data item that has been enciphered at the broadcast center using some class dependent key $k(i)$. Suppose that $k(i)$ also serves for deciphering. Then, in order to satisfy the above statements, one has to provide terminal t with the set of keys $K_t = \{k(i)|i \in c_t\}$ (and no other keys, of course).

As trivial as this might seem there are still some questions related to this approach that deserve attention:

(i) Where and how are the keys to be generated and how will they be distributed to the subscriber terminals?

(ii) Which method would in this case be a good choice for enciphering/deciphering data items?

(iii) How can sender and receiver be kept synchronized?

(iv) Which measures may be taken in order to protect a subscriber terminal against fraudulent manipulations?

Each of these questions will be discussed in turn in the remaining subsections.

2.2 Key Generation and Distribution

The generation of class-specific keys takes place at the broadcast center. A true random number generator may be employed for this purpose. Clearly, all contracts are known at the central system. In order to get its keys a terminal t sets up a two-way connection with the broadcast center and requests the transmission of K_t. This request may be part of a daily login procedure. The user's and the terminal's authorization have to be checked of course. This can be done using "classical" methods: user identifications, passwords and terminal identifications (to be stored in read-only memory) for instance. It is important however, that the keys are well protected against eavesdroppers on their way to the terminal. Among the best available methods achieving a high degree of security are public-key cryptosystems (cf. for example [3], [4]). The use of public-key cryptosystems entails the additional advantage of being able to authenticate the identifications of users and terminals. A protocol involving public-key encryption and decryption of the *broadcast-keys* and authentication as well will in fact be part of the software to be described in section 3.

Besides the keys proper other data will be sent over the bidirectional link to the terminals. The role of these data will be explained in section 3.

The contract-specific keys will be stored within the terminal such that upon receipt of an information element (i, d)

– it is easy to check whether the corresponding key $k(i)$ is present or not and – in case this check comes out in the affirmative –

– the key can be retrieved quickly.

In order to enhance the security of the system the current set of class-specific keys should be completely or partially changed on a regular basis. It will be expounded in sections 2.5 and 3 how this can be done during the ongoing operation.

2.3 Enciphering and Deciphering

Besides the usual demands on cryptosystems (concerning their resistance to all kinds of "attacks", cf. [3]) our choice is constrained by two additional requirements that are due to specific characteristics of the application:

– Since we are dealing with a real-time environment the processes of enciphering and deciphering must be executable at high speed;

– secondly, the selected procedure must be sufficiently insensitive to transmission errors, i.e. deciphering the data of a correctly received information element must not be impeded by any previous garblings.

These requirements will be met best by the well known technique of synchronous stream ciphering (cf. [3]). In our case cipher streams d' may be generated by bitwise XORing the individual data items d (considered as bit sequences) and bit sequences s, with s being a function of the class i of d but not of d itself. To put it more precisely, the mechanism that produces s must take into account the key $k(i)$. A pseudo-random number generator using one or several seed values at the outset may be underlying such a mechanism. The seed or one of the seeds can be $k(i)$ itself or some number depending on $k(i)$.

2.4 Synchronization

Obviously, upon receipt of an information element (i, d') the same sequence s that has been used at the broadcast center to transform d into d' must also be produced by terminals which are entitled to accept data items of class i in order to perform the inverse transformation. The receiver must work in unison - synchronously - with the sender.

A straightforward preliminary solution of this problem might look as follows: Whenever the sender is going to put a data item d of class i on the air he initializes the mechanism producing the enciphering sequence s with some value that only (!) depends on $k(i)$. The resulting d' can be deciphered by the receiver if he possesses that key and if he has the same mechanism at his disposal.

Unfortunately, this solution, albeit simple, lends itself rather easily to attacks (by eavesdropping) since items of class i will invariably be enciphered by the same sequence s. To avoid this regularity a somewhat trickier approach has to be taken. We assume that any number of information elements $(i_1, d_1), \ldots, (i_m, d_m)$ can be combined to form an object called *frame* (cf. Figure 1). The number m of elements in a frame need not be fixed and will in general be part of that structure. The crucial constituent of a frame however, will be an arbitrary, randomly generated value a_0, termed *frame synchronization value*. Frames will from now on be the units of transmission.

a_0	m	(i_1, d_1)	$\cdots$	(i_m, d_m)

Figure 1: Structure of a frame

Let C be the mechanism for producing the enciphering/deciphering sequences s. We postulate another mechanism, F, also common to both sender and receiver and which may be just another pseudo-random number generator. In order to emit a frame the sender first (arbitrarily) determines the frame synchronization value a_0. He then successively computes $a_1 = F(a_0), a_2 = F(a_1), \ldots, a_m = F(a_{m-1})$ and enciphers data item d_j $(j = 1, \ldots, m)$ using C initialized with a_j and $k(i_j)$. Hence the enciphering sequence corresponding to d_j no longer depends on i_j alone but also on the particular frame that carries d_j. It follows that each time a data item of a given class is sent off it will in general be enciphered by a different sequence. Upon receipt of a frame the same sequence $a_1, \ldots, a_m$ will be generated in the terminal irrespective of whether it is in possession of the keys for accessing all the information elements contained in that frame. Whenever it encounters an element, say (i_j, d_j), for which it owns $k(i_j)$ it will start C on a_j and $k(i_j)$ and bitwise recover the plaintext from d_j. Otherwise it will simply ignore that element. (For a more detailed exposition of this procedure see the description of ProcessFrame in section 3.)

We have thus obtained a solution of the synchronization problem which – as a byproduct – also enhances the quality of data protection on the broadcast channel. It is similar to a scheme suggested by [5].

2.5 Prevention of Fraud

It should have become quite clear from the above that stealing information by eavesdropping on the channel will be a rather toilsome business. (Although we are still owing a formal proof of that claim.) The measures however, that have been taken so far do not preclude manipulations of the terminal software which may aim at getting more data than the terminal is entitled to accept. The most vulnerable part of course is the key store. Any (sensible) change of its contents must be made impossible. (We assume that the initial loading of the key store, described in section 2.2, presents no difficulty as far as security considerations are concerned.)

To make sure that no unwanted manipulation of neither the key set nor the terminal's system-software (!) occurs a piece of telesoftware will be broadcast time and again to the terminals. The functions performed by this routine will be explained in the next section.

It should be noted that as far as transmission is concerned the bits and bytes of telesoftware are treated in the same way as any other data item. They just belong to special classes some of which may be automatically included in every contract.

3 The System

The software implementing the ideas set out above will be conceived as a distributed system which splits up into separate subsystems that communicate by data flow only. There is no transfer of control from one subsystem to another subsystem. However, all subsystems of a system may share certain constants and data types. A distributed system may be represented by a directed graph whose nodes are the subsystems and whose edges are labeled with the type of data flowing between the respective subsystems. Figure 2 displays the system SecuredBroadcast as composed of the subsystems BroadcastCenter and BroadcastTerminal.

The types assumed to be known throughout the system include the following:

- RnType, the range of (pseudo-)random numbers,
- IdType, the range of class identifiers,
- IeType, the (RECORD-)structure of information elements with components
  id (of IdType),
  length (of type CARDINAL) and
  info (of type BitString[length]) (cf. 2.1),
- FrameType, the (RECORD-)structure of frames with components synchval (of RnType), count (of type CARDINAL) and contents (of type ARRAY [1... count] OF IeType) (cf. 2.4),
- KeyStoreType = MAPPING IdType TO RnType.

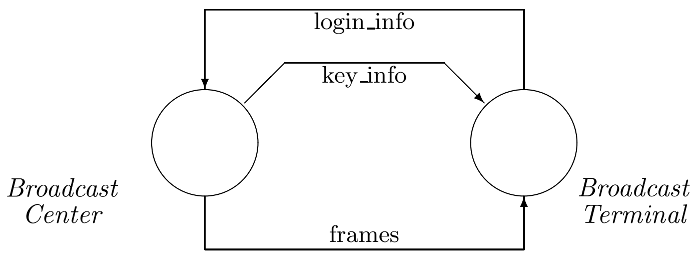

Figure 2: The system *SecuredBroadcast*

A subsystem consists of stores (or *variables*), processes, channels, procedures and, in general, other subsystems. A process is a sequence of activities which goes on concurrently with other processes. Channels serve for transferring data from one process to another. There is a distinguished procedure that starts up the subsystem by spawning its processes. In general, however, procedures are subordinated to processes inasmuch as they execute some or all activities of a process. Again, constants and types may be shared by all the objects contained in a subsystem. Since there is a variety of types of objects and of relations between these objects the graphical representation of a subsystem tends to look slightly more complicated than that of an entire system. Figure 3 displays the software-structure of BroadcastTerminal.

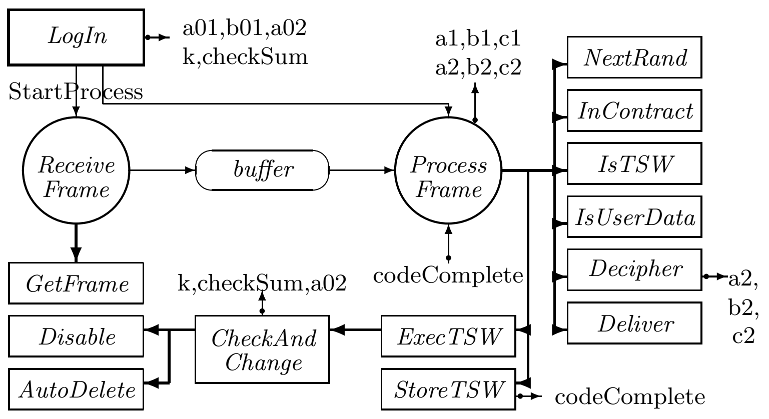

Figure 3: Subsystem *BroadcastTerminal*

Circles denote processes, procedures are represented by rectangular boxes and channels by oval boxes. A fat arrow means procedure call. Thin arrows either mean channel access (the respective operations are Deposit and Fetch) or starting a process. Depending on their directions, dotted arrows either mean read- or write-access to variables.

The following variables are global with respect to the subsystem BroadcastTerminal:

– a01,b01: RnType – store the first two initial values for generating the frame synchronizing pseudo-random numbers,

– a02: RnType – stores the first initial value for generating the deciphering pseudo-random numbers,

– k: KeyStoreType – stores the contract-specific keys,

– checkSum: RnType – stores a checksum computed over all keys in k.

The values to be stored in a01, b01, a02, k and checkSum are transmitted online at login-time, using public-key encryption and authentication. Frames arriving on the broadcast-channel are accepted by the process ReceiveFrame and – via buffer – forwarded to ProcessFrame. The following variables are global with respect to this latter process:

– f: FrameType – stores the current frame,

– ie: IeType – stores the current information element,

– a1,b1,c1: RnType – store the three most recent synchronizing pseudo-random numbers,

– a2,b2,c2: RnType – store the three most recent deciphering pseudo-random numbers.

The code controlling ProcessFrame is shown below:

```
BEGIN ProcessFrame
  FOREVER DO
    buffer.Fetch (f);
    a1:=a01; b1:=b01; c1:=f.synchval;
    FOR i:=1 TO f.count DO
      NextRand (a1, b1, c1);
      ie:=f.contents[i];
      IF InContract (ie.id) THEN
        a2:=a02; b2:=k[ie.id]; c2:=a1;
        Decipher (ie);
        IF IsUserData (ie.id) THEN
          Deliver
        ELSIF IsTeleSoftware (ie.id) THEN
          StoreTSW
        END {IF}
      END {IF}
    END {FOR};
    IF codeComplete THEN ExecTSW END {IF}
  END {FOREVER}
END ProcessFrame;
```

The random number generator NextRand(VAR a,b,c: RnType) acts as mechanism F (cf. 2.4) when called on (a1,b1,c1). The procedure Decipher(VAR ie:IeType) yields plaintext in ie.info. The deciphering bits are produced by calling NextRand on (a2,b2,c2) (mechanism $C!$). Upon exit a1 and a2 will contain the respective next pseudo-random number whereas b1,c1 and b2,c2 will contain the respective predecessors. Observe that the initialization involves f.synchval or k[ie.id] depending on whether NextRand is used as F or C.

If an information element contains telesoftware it will be stored by StoreTSW and – upon completion – executed by ExecTSW. The only piece of telesoftware of interest in the present context is the procedure CheckAndChange. It is crucial for ensuring the safety of the broadcast system as far as the terminals are concerned. It regularly checks the terminal's current state (key-store and critical software parts) and changes the class-specific keys and some or all initial values for pseudo-random number generation. Thus any manipulation of a terminal's code and data will at best entail a malfunction of the deciphering procedure. CheckAndChange is based on data transmitted along with the code proper. In general, different data will be included each time the procedure is broadcast. These data are:

– a set Alpha $= \{$Alpha[i] $|$ i $\in$ IdType$\} \subset$ RnType which will be used for checking the current keys;

– a set KeyUpdates of triples (i, keydelta[i], csdelta[i]) where
  - i ranges over a subset of IdType,
  - keydelta[i] (of RnType) updates k[i], the current key for class i,
  - csdelta[i] (of RnType) updates the current checksum following an update of k[i].

  The csdelta[i] are computed at the broadcast center with respect to the set Alpha that will be included with the next transmission of the CheckAndChange procedure.

– Finally, there is a set Locations containing addresses (selected at random by the broadcast center) within critical parts of the terminal's software (other than CheckAndChange); these parts are assumed to be identical in every terminal. Locations is used to update a02, one of the values initializing the deciphering pseudo-random number generator.

The operations performed by the CheckAndChange procedure must be explained in connection with some of the functions of the BroadcastCenter software, the basic structure of which is shown in Figure 4.

Of course, the subsystem BroadcastCenter "knows" all contracts. This knowledge may be represented as a table contracts: TerminalIdType $\times$ IdType $\rightarrow$ BOOLEAN. contracts[i,j] is TRUE if and only if terminal i is entitled to receive information elements of class j.

Furthermore, BroadcastCenter has five variables of type KeyStoreType at its disposal: k, alphaNext, alpha, keyDelta and csDelta.

k contains all currently used class-specific keys. It is initially filled by the procedure GenerateInitKeys.

GenerateInitKeys also initializes alphaNext whose contents is used by the LoginHandler to compute checksums over a terminal's set of class-specific keys. alphaNext will be (randomly) modified by the CheckAndChange process. Prior to each modification its contents is assigned to alpha which forms part of the CheckAndChange telesoftware to be broadcast to the terminals.

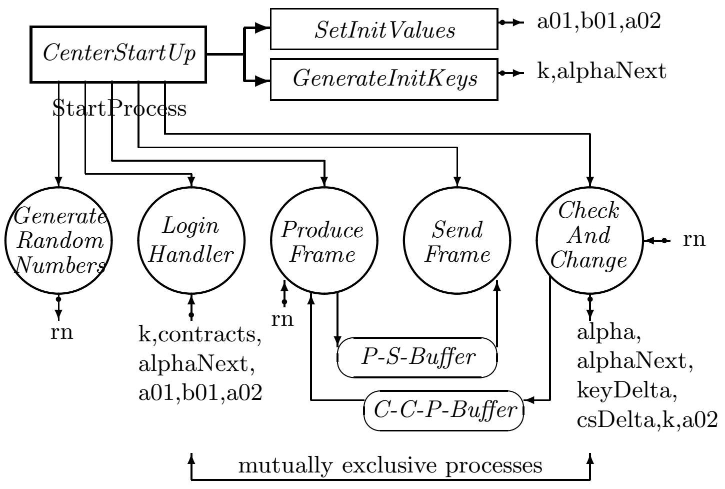

Figure 4: Subsystem *BroadcastCenter*

Likewise, the CheckAndChange process modifies k taking into account the randomly generated contents of keyDelta. The values needed to update the corresponding checksums will be computed and stored in csDelta, the latter being a function of keyDelta and alphaNext. keyDelta and csDelta make up the set KeyUpdates described above.

The values initializing the mechanisms F and C (cf. 2.4) are stored in global variables also named a01,b01 and a02 respectively. They are sent to the terminals (at login-time) by the process LoginHandler. Process CheckAndChange repeatedly modifies the value of a02, using the set Locations defined above. At the terminals these modifications are made accordingly by the procedure CheckAndChange.

All random numbers used by GenerateInitKeys and the CheckAndChange process are *true* random numbers produced continuously by the process GenerateRandomNumbers and made public in the variable rn.

The interworking of the CheckAndChange facilities at both the center and the terminals should become clear from Figure 5. cd(i) and cs(i) denote the contractual data of terminal i and the corresponding checksum respectively. The checksum is computed by summing over the products of alphaNext- (or Alpha-)elements and k-elements. (In general, the functions f1, ..., f5 involve simple arithmetics modulo two to the size (in bits) of RnType.)

Whenever the checksum computed at a terminal differs from the value stored in checkSum the CheckAndChange procedure invokes the procedure Disable which renders the terminal inoperational (by erasing the key-store for instance) as far as its receiving capabilities are concerned. Furthermore, the CheckAndChange procedure deletes itself (by calling AutoDelete) upon completion of its job.

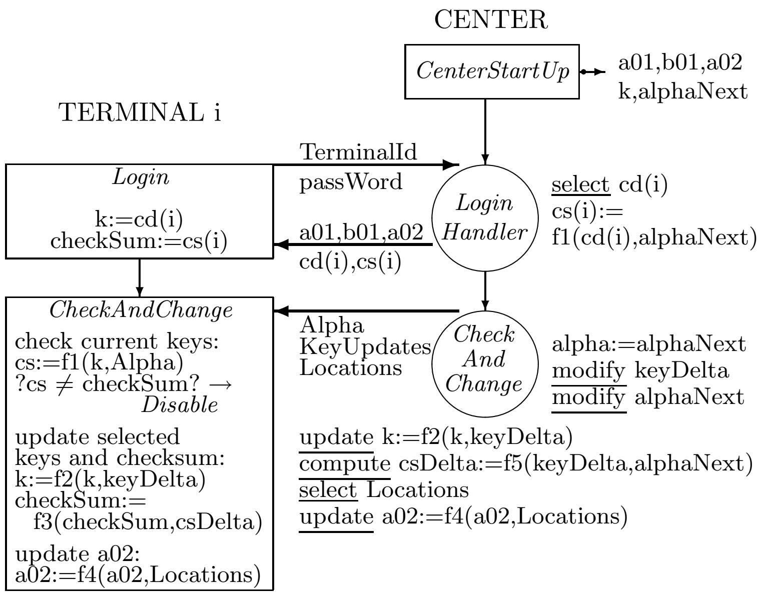

Figure 5: Interworking of CheckAndChange facilities

3.1 Summary and Final Remarks

The requirements set out in section 1 may be summarized as follows:

(1) Access to information that belongs to classes a client is subscribing to must be guaranteed.

(2) Access to information that belongs to classes a client is not subscribing to must be denied.

(3) The proposed system must be able to cope with frequent changes of contracts.

(4) It must be possible to change (and/or extend) the (amount of) information offered without disrupting the dissemination of information that is not affected by these changes.

Keeping in mind the model outlined in sections 2.1 – 2.4 it is quite obvious that the requirements (1) and (2) can be satisfied without much ado if a user does not manipulate his terminal. This assumption however, may not take into account the well-known fallibilities that are so characteristic of human nature. As mentioned above (2.5) a manipulation that would certainly make sense consists in changing the set of class-specific keys, by adding or replacing keys for instance. It is this kind of forgery that we claim to be rendered impossible (or useless!) by the check-and-change facilities described in section 3.

It has already been pointed out in section 2.1 that the requirements (3) and (4) are automatically satisfied due to the concept of information classes. Changing a contract amounts to changing the set of class-specific keys that is sent to a terminal at login-time. Hence, no on-site modifications are necessary. Also, the total number of information classes, being a parameter of the system, can be increased or decreased at short notice.

The cryptological quality of the system hinges on the methods that are used for enciphering/deciphering the messages proper and for protecting the enciphering/deciphering keys. Strictly speaking, we are dealing with a three-leveled key-hierarchy. On the "bottom-level" there is the sequence of bits that are XORed with the plaintext and ciphertext bits respectively. This sequence may be termed *key-stream*. A specific key-stream is generated for each individual information element by initializing a suitable device (algorithm) producing pseudo-random numbers with the *class-specific key* (a "level-2-key") and some value that depends on the frame which accomodates the given element. (Note that the third value, stored in a02, serves as a safeguard against unwanted manipulations of the terminal software. This explains why NextRand must take into account the three most recently produced pseudo-random numbers!) Intuitively, the probability of using some key-stream twice is in fact very small. Thus the scheme we are proposing comes close to the well-known technique of *one-time pads* (cf. [3]). Although generators that are based on the linear congruence method are known to be cryptographically insecure (cf. [1]) we conjecture that generators of this type are in fact sufficient in the present context, due to the frequent change of initial values.

Finally, on the third level, so to speak, there are the *terminal-specific public keys* which are used for the online-transfer of contract-specific class-keys to the subscribers. Results concerning the security and practicability of public-key cryptosystems are widely publicized (cf. [3]) and need not be restated here.

A discussion of security issues related to the central station is not within the scope of the present paper. (Fundamental questions concerning this matter are treated for instance in [2].)

References

[1] J. Boyar: Inferring sequences produced by pseudo-random number generators. *J. ACM, Vol.36, Jan. 1989, pp. 129-141*

[2] R.A. DeMillo et al.: Applied Cryptology, Cryptographic Protocols and Computer Security Models. *Proceedings of Symposia in Applied Mathematics, Vol.29, AMS Short Course Lecture Notes, American Mathematical Society, Providence (Rhode Island), 1983*

[3] D.E.Robling Denning: Cryptography and Data Security. *Addison-Wesley Publishing Company, Reading (Massachusetts), 1983*

[4] R.L.Rivest,A.Shamir,L.Adleman: A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, Vol.21, 1978, pp.120-126*

[5] S.C.Serpell,C.P.Brookson: Encryption and key management for the ECS satellite service. *Advances in Cryptology, Proceedings of EUROCRYPT 84, Springer Verlag, Berlin, 1985 (Lecture Notes in Computer Science, Vol. 209, pp. 426-436)*