DIMACS Technical Report 98-06
January 1998

Key Management for Encrypted Broadcast

by

Avishai Wool[1]
Bell Laboratories, Lucent Technologies,
700 Mountain Ave., Murray Hill, NJ 07974.
E-mail: `yash@research.bell-labs.com`.

[1]DIMACS member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Research, Bellcore, and Bell Laboratories.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We consider broadcast applications where the transmissions need to be encrypted, such as broadband digital TV networks or Internet multicasts. In these applications the number of encrypted TV programs may be very large, but the secure memory capacity at the set-top terminals (STT) is severely limited due to the need to withstand pirate attacks and hardware tampering. Despite this, we would like to allow the vendor to offer different *packages* of programs to the users. A user who buys a package should be able to view every program belonging to that package, but nothing else. A *flexible* scheme should allow for packages of various sizes to be offered, from a single program up to all the programs.

We suggest a novel scheme to manage the encryption keys for these applications. The scheme is highly flexible, yet requires very few keys to be stored in the STTs' secure memory. The computational power required of the STTs is very low. The security of this scheme is as good or better than that offered by current technology.

1 Introduction

1.1 The Problem

The domain we consider in this paper is that of broadcast applications where the transmissions need to be encrypted. As a primary example we consider a broadband digital TV network, broadcasting either via satellite or via cable, but other applications such as Internet multicasts are similar. The reason encryption is needed is to ensure that only paying customers, who have the required keys, will be able to view the programs [MQ95].

In this context, the *head-end* broadcasts the encrypted TV programs to a large population of users. Each network user has a *set-top terminal* (STT) which receives the encrypted broadcast and decrypts the programs that the user is entitled to. The secret information which is stored inside a user's set-top terminal (keys and access control data) is collectively called an *entitlement*.

Because of extensive piracy, these STTs need to contain a secure chip which includes secure memory for the entitlements. This memory should be non-volatile, writable (since keys are changed every billing period), and tamper-resistant (so the pirates will find it difficult to read its contents). As a result of these requirements, STTs have severely limited secure memory, typically in the range of a few KB. On the other hand, the number of programs that may be broadcast during a billing period can be large, say 200,000 or more. The problem we are faced with is how to manage the keys for such an application. We are trying to achieve the following two goals:

1. Flexibility: We would like to allow the TV network to offer many different *packages* of programs to the users. A user who buys a package should be able to view every program belonging to that package, but nothing else. A flexible scheme should allow for packages of various sizes, from a single program up to all the programs. This flexibility should be obtained under the constraint that the STTs can only store very few keys.

2. Security: Piracy is a major concern, so all the components of the scheme must be made secure. We must certainly ensure that it is infeasible to attack the encryption algorithms directly. In addition, we should not completely trust the "tamper-resistance" the STTs. This is since pirates have been notoriously successful in tampering with these devices. A good scheme is one where if the pirates break into an STT they would not be able to decrypt *more* than what that STT's owner was entitled to.

1.2 Contributions

In this paper we describe key management schemes which achieve the flexibility and security goals we set for ourselves, and which use very little secure memory in the STTs. The computational power required of the STTs by the schemes is very low. The security of these schemes is as good or better than that offered by current technology. Thus they offer practical and applicable solutions.

We start by describing a simple scheme called ExtHeader, which is a variant of the well known "encrypt the session key with the user key" technique. The scheme works by adding header information to each broadcast program. Using it, any predefined set of programs can be a package. Thus the scheme has full flexibility—although the length of the header information grows with the number of packages. If the number of predefined packages is not too large then ExtHeader may be a reasonable choice.

Our main contribution is the Vspace scheme. By using it we can provide a high degree of flexibility in the creation of program packages, essentially *without adding any overhead* to the transmission. Moreover, the packages need not be predefined completely, and to some extent may be customized to fit a user's individual taste. The price we pay for this freedom is that not every arbitrary set of programs may be a package (e.g., packages can only be of $1, 3, 7, 15, \ldots, 2^n - 1$ programs).

The rest of this paper is structured as follows. We review some related work in Section 2. In Section 3 we describe the ExtHeader scheme. The Vspace scheme is presented in Sections 4 and 5. The security of both schemes is discussed in Section 6. In Section 7 we analyze the addressing power of the Vspace scheme, and we conclude with some directions for future work in Section 8.

2 Related Work

The bit-vector scheme. This is an access control scheme which is reportedly used by some analog European satellite TV systems, e.g., the Sky VideoCrypt systems [McC96, p. 4-10]. According to the pirate reports [McC96, p. 9-56], the recent digital DirectTv system (VideoGuard) in the US is similar to VideoCrypt, and thus may also use the bit-vector scheme.

In this scheme, all the programs are encrypted with the same key, which is stored in every STT. To control the user's access to the programs, the STT also stores an additional *bit-vector* $\mathbf{b}$ which has an entry for each program. The STT decrypts a program $\mathbf{p}$ only if $\mathbf{b}[\mathbf{p}] = 1$. Thus the scheme has full flexibility, since the user can buy any set of programs as a package.

However it has two major disadvantages. First, the scheme needs to store the bit-vector in secure memory. Therefore the size of the bit-vector becomes prohibitive when the number of programs is large. Second, the access control is non-cryptographic. Therefore if the pirates can overwrite the bit-vector with an all-1 vector, the STT will decrypt every program and bypass the access control entirely. Alternatively, if the pirates can extract the key from the STT then they can build a decoder that will decrypt every program. This second disadvantage seems to have contributed to the fact that both the VideoCrypt and VideoGuard systems were indeed extensively pirated [McC96].

The block-by-block scheme. In this scheme, the programs are split into n disjoint *blocks*. All the programs belonging to a block are encrypted using the same key. The STT stores the keys for each block that the user buys. According to the pirates' reports, such a system has been used in the D2-MAC EuroCrypt system [McC96, p. 4-11].

The block-by-block scheme is more secure than the bit-vector scheme since the access control is cryptographic. In addition the secure memory requirements are small; at most n keys need to be stored. The main drawback of this scheme is its *poor flexibility* since a user cannot buy less than a large block of programs. Moreover, a program which is required to belong to several blocks incurs a high overhead. Such a program needs to be re-encrypted with multiple block keys, and re-transmitted separately for each block.

A possible remedy is to split the programs into many blocks which contain only a few programs each, or even to have a single program per block. But then the amount of secure memory in the STT limits the maximal number of block keys that a user may buy, and in particular a user may be unable to buy the set of all programs as a package. So the remedy may be worse than the original problem.

How to transfer keys to the STTs? A related question is how to transfer the key information to the set-top terminals at the beginning of each billing period in a secure and efficient way. Two types of solutions can be found in the literature, which differ in the communication model they assume.

In the first model, the only type of communication between the head-end and STTs is the uni-directional broadcast. Under this model, Fiat and Naor [FN94] suggested methods of securely broadcasting key information such that only a selected set of users can decrypt this information while coalitions of up to k other users can learn nothing. Extensions to this basic work can be found in [BC94, BFS96].

In the second model, the STTs also have an uplink capability (e.g., via a phone line or cable-modem). This model fits modern system designs such as the Lucent IVES™ architecture. The solution suggested in [CEFH95] is based on the idea that each billing period the head-end runs a secure and authenticated protocol with every STT. During this protocol key information is downloaded to the STT, and pay-per-view data is uploaded. This architecture offers better security than the one based on uni-directional communication in that it limits a pirate's ability to obtain secret keys—they are never broadcasted.

Alternative approaches. Key management schemes which have full flexibility (i.e., allow users to access any set of programs they wish) can be found in [CT90, HP95]. However their techniques rely on RSA public-key cryptography [RSA78] and therefore are not considered to be applicable in our scenario.

Protocols for conference key distribution [CC89, Ber91, Gon94] have more stringent security requirements than we do here, such as verifying the key's authenticity and repelling replay attacks. Moreover they do not address our main constraint, which is the limited secure storage of the STTs.

3 The Extended-Header Scheme

3.1 The Scheme

We start by describing a simple scheme which we denote by ExtHeader. The main idea in this scheme is that we attach cryptographic header information to each program. By transmitting more data we can overcome the limited secure storage at the STTs. The scheme is a variation of the well known "encrypt the session key using a user key" technique. However instead of having a separate key for each user, we arrange the programs into predefined *packages*, and each package has a key. Each program may belong to many packages, and each user may have the keys to multiple packages.

Suppose that a program $\mathbf{p}$ belongs to t packages. Then the transmission of $\mathbf{p}$ consists of the encrypted program itself, and a list of t header blocks. Each header block H contains two fields:

the package identifier field $H.ID$ and the key field $H.Key$. When the ID field contains the package identifier j, then the Key field contains $E_{s_j}(K_{\mathbf{p}})$, which is the program key $K_{\mathbf{p}}$ encrypted using the package key s_j. We say that the program key $K_{\mathbf{p}}$ is *covered* using the package key s_j.

Note that the encryption used for the programs may be different from the encryption used to cover the keys. The former needs to allow real-time decryption of a video stream, while the latter only encrypts short but potentially more sensitive keys.

The ExtHeader scheme is already a big improvement over the block-by-block scheme in terms of flexibility, since *any* set of programs may constitute a package. Its simplicity is another attractive feature.

3.2 Decreasing the Header Bandwidth

A drawback of the ExtHeader scheme is the amount of additional data that is transmitted. If a program $\mathbf{p}$ belongs to t packages, then the scheme requires $t \times (|H.ID| + |H.Key|)$ header bits to be transmitted along with $\mathbf{p}$. For example, if the size of a header block is 160 bits and assuming a program may belong to 64 packages, this amounts to an extra 1280 bytes per program.

A few extra KB of headers may seem negligible in comparison to the size of a typical video clip. However note that this header information should be transmitted fairly frequently. This is due to the fact that when Alice switches to view program $\mathbf{p}$ the STT needs to wait until $\mathbf{p}$'s next header is received before it can start decrypting. Infrequent header transmission would cause Alice to notice delay when she switches channels. Therefore the scheme has a design tradeoff between the bandwidth allocated to header transmission, and the delay a user would incur when switching channels. Since tolerable delays are measured in fractions of a second, the bandwidth overhead may be significant.

To overcome this problem, we suggest the following mechanism. Assume that the STT is powered up and receiving the broadcast transmission continuously (even when the TV is turned off). Assume further that in addition to its small secure memory, the STT has access to a large *insecure* memory, e.g., banks of low-cost RAM. Then the STT can use this RAM as a *key cache* and thereby reduce the frequency of header transmission.

The STT needs to receive the broadcast continuously and scan all the channels for header blocks. For every transmitted program $\mathbf{p}$, if one of its header blocks belongs to a package that the user is entitled to, the STT copies the block into the cache. We then say that the header block was *captured* by the STT. When Alice switches to program $\mathbf{p}$ the required header block will already be in the cache, provided that the STT was operational long enough to capture it. Therefore a noticeable delay will only occur when the STT is powered up for the first time, or when Alice attempts to switch to a program she is not entitled to. Using this capturing mechanism, we can choose the frequency of header transmissions based on the tolerable power-up delay, which may be on the order of seconds or even minutes.

Note that the headers are copied to the cache without decrypting their Key field. The program keys are uncovered *inside* the secure chip only when Alice switches to a specific program. Thus the only information a pirate can obtain by reading the (insecure) cache is the list of programs that Alice is entitled to. The actual data placed in the cache can just as easily be extracted from the broadcast transmission itself.

An alternative mechanism would be to designate a fixed channel which would repeatedly broadcast the header information of all the programs for the next hour (say). Such a fixed channel, called a Barker channel, may already exist in the system (e.g., serving as a directory listing). If such is the case then the extra header bandwidth may be "for free" in some sense since the Barker channel is not used to broadcast program contents.

4 The Vspace Scheme

4.1 Overview

In the ExtHeader scheme we needed to attach large headers to each program in order to achieve flexibility in packaging the programs. In the Vspace scheme we achieve comparable flexibility essentially without attaching *any* extra headers.

The only piece of information we attach to a program's transmission is a single n-bit *cryptographic identifier*, or CID. However this CID is not chosen arbitrarily. We shall see that imposing a specific structure on these identifiers is crucial to the scheme's usefulness.

The program encryption keys cannot be chosen arbitrarily either. The encryption key of a program $\mathbf{p}$ is a *function of its CID* and of *secret data* stored at the head-end. The STT recomputes the key using the transmitted CID and its own secret data. However the secret data stored in the STT only allows the decryption of programs with certain CIDs, namely those which belong to the package the user buys.

Finally, the Vspace scheme does not allow every arbitrary collection of programs to be a package. One simple restriction is that only collections of $1, 3, 7, \ldots, 2^n - 1$ programs may constitute a package.[1] However we argue that the large variability in package size, from a single program package up to all the programs, gives sufficient flexibility.

In the rest of this paper we identify a program with its CID, and refer interchangeably to either a "program $\mathbf{p}$" or a "program with CID $\mathbf{p}$".

The Vspace scheme is parameterized by two numbers. The length (in bits) of a CID is denoted by n, and the length of an encryption key is denoted by k. We require that $k > n$ for the scheme to work. A convenient value for n is $n = 32$, as this size allows a program's CID to be placed in the ECM field defined in the MPEG-2 standard [MPE94]. For sake of concreteness we can think of using encryption keys of length $k = 64$ or $k = 128$ bits.

The Vspace scheme relies on basic results of linear algebra. A good reference book for the required mathematical background is [BM77]. We use boldface letters to denote program CIDs, which we view as n-bit vectors over $GF(2)$. We use capital letters to denote matrices over $GF(2)$.

[1] More precisely, a package consists of programs with $2^r - 1$ CIDs; note that it is possible to assign the same CID to multiple programs.

4.2 Generating the Encryption Keys

At the beginning of every billing period, the head-end chooses a random binary $k \times n$ matrix M called the *master matrix*. The k-bit columns of the matrix M, denoted by $\mathbf{m}_1, \ldots, \mathbf{m}_n$, are called the *master keys* of the scheme. We require the master keys to be *linearly independent* k-dimensional vectors over $GF(2)$. Therefore the matrix M is n-dimensional.

The keys for encrypting the programs are generated from the master keys as follows. A program whose CID is $\mathbf{p}$ is encrypted using the key

$$\boxed{Key(\mathbf{p}) = M\mathbf{p}.} \tag{1}$$

This method of generating keys as linear combinations of the master keys is similar to the techniques used in [IN89, ARV95, FS96] to construct provably secure pseudo-random generators. However a pseudo-random generator in itself is not sufficient for our purposes; in the next section we show how we capitalize on the linearity of the scheme in the definitions of the possible packages.

Remarks:

- A program which has CID $\mathbf{p} = \mathbf{0}$ would get $Key(\mathbf{p}) = \mathbf{0}$ for any master matrix M. Many popular encryption algorithms produce a ciphertext which is identical to the plaintext when the key is all zeros. Therefore we may as well assign the CID $\mathbf{p} = \mathbf{0}$ to any program that needs to be broadcast in the clear, such as directory listings or the major TV networks' broadcasts. We can then use the value $Key(\mathbf{p}) = \mathbf{0}$ as a flag to bypass the encryption (or decryption) algorithm altogether.

- Since we require the master keys to be linearly independent, it is impossible that a program $\mathbf{p} \neq \mathbf{0}$ would accidentally get a zero key, and thus be broadcast in the clear.

- Since $k > n$ it is always possible to pick n linearly independent k-bit master keys. Indeed if we pick n keys uniformly at random then the probability of their being linearly independent is

$$\mathbb{P}(\text{master keys are linearly independent}) = \prod_{i=0}^{n-1} \left(1 - \frac{1}{2^{k-i}}\right) \geq e^{-1.386/2^{k-n}}.$$

  For $n = 32$ and $k = 64$ this value is $\approx 1 - 10^{-10}$.

4.3 The Linear Subspace Paradigm

At first sight, the key generation scheme proposed in the previous section seems to "leak information". If Alice knows the keys $Key(\mathbf{p}_1)$, $Key(\mathbf{p}_2)$ of programs $\mathbf{p}_1$ and $\mathbf{p}_2$, then she can compute $Key(\mathbf{p}_1) \oplus Key(\mathbf{p}_2)$, which is precisely the key to the program $\mathbf{p}_1 \oplus \mathbf{p}_2$ by the linearity of the scheme.

However this "deficiency" turns out to be the source of the scheme's flexibility. To see this, we introduce the central paradigm of the Vspace scheme (and the reason for its name), which is

A user is only allowed to buy an entitlement for a *linear subspace* of program CIDs.

So in fact the user Alice does not gain any information she is not entitled to. Since she is entitled to decrypt both $\mathbf{p}_1$ and $\mathbf{p}_2$, she must have bought (and paid for) a linear subspace U of CIDs which contains both $\mathbf{p}_1$ and $\mathbf{p}_2$. But U must also contain $\mathbf{p}_1 \oplus \mathbf{p}_2$ by the definition of a linear subspace. Therefore if Alice can decrypt both $\mathbf{p}_1$ and $\mathbf{p}_2$, then the price she paid for the entitlement already reflects her ability to decrypt $\mathbf{p}_1 \oplus \mathbf{p}_2$.

An r-dimensional linear subspace over $GF(2)$ contains 2^r vectors (CIDs), one of which is $\mathbf{0}$ (marking plaintext programs). This is the reason why packages can only contain $1, 3, 7, 15 \ldots, 2^n-1$ programs.

4.4 Creating the Entitlements and Decryption

Assume that Alice has decided to buy a package of programs which is characterized by an r-dimensional linear subspace U of CIDs. Then the head-end needs to generate an entitlement which will allow Alice to decrypt any program $\mathbf{p} \in U$ but nothing else. In Section 4.4.1 we describe what such an entitlement consists of, and how the head-end can generate it.

An optional part of the entitlement is the *check matrix*. If the STT has the check matrix it can determine whether a program $\mathbf{p}$ is in the decrypt-able space U without going through the whole decryption procedure. The check matrix is described in Section 4.4.2.

4.4.1 The Decryption Procedure

The inputs for the head-end procedure that generates the decryption data are the master matrix M, and the r-dimensional subspace U. We assume that U is represented by a *basis*, i.e., an $n \times r$ matrix B.

Consider some program $\mathbf{p} \in U$. Since B is a basis for U we can write $\mathbf{p}$ as a linear combination of the basis vectors, i.e., there exists an r-dimensional vector $\mathbf{x}$ such that

$$B\mathbf{x} = \mathbf{p}. \tag{2}$$

Given B and $\mathbf{p}$, we need to solve (2) for $\mathbf{x}$. Note that if $r < n$ (i.e., U is not the space of all programs) then the equation system (2) is over-defined; it has n equations and r variables. Nevertheless, if $\mathbf{p} \in U$ we know a solution exists. In the next definition we identify the data needed to solve (2).

Definition 4.1 *Let the* active indices $i_1 \ldots, i_r$ *be indices of r rows of B which form a regular $r \times r$ submatrix B'. Let $\mathbf{p}' = (p_{i_1}, \ldots, p_{i_r})$ be the r-dimensional vector of the corresponding entries in $\mathbf{p}$.*

Remark: It is always possible to find r such linearly independent rows since B is r-dimensional.

Proposition 4.2 *Let B' and $\mathbf{p}'$ be as before, and let $(B')^{-1}$ be the $r \times r$ inverse of B'. Then*

$$\mathbf{x} = (B')^{-1}\mathbf{p}'$$

is the unique solution to the linear system of equations (2).

Proof: Clearly $\mathbf{x}$ is the unique solution to the system of equations $B'\mathbf{x} = \mathbf{p}'$ since B' is regular. Thus $\mathbf{x}$ solves the r equations of (2) corresponding to the active indices. However *any* solution to the whole system (2) must solve these r equations in particular. Therefore if (2) has a solution at all it can only be $\mathbf{x}$. But (2) has a solution since $\mathbf{p} \in U$ and B is a basis for U, so $\mathbf{x}$ is the required solution. ▮

The matrix $(B')^{-1}$ and the active indices $i_1, \ldots, i_r$ form one part of the entitlement. Another part, which actually contains the secret key data, is the following matrix K.

Definition 4.3 *Let K be the $k \times r$ matrix*

$$\boxed{K = MB.}$$

In order to generate the decryption part of the entitlement, the head-end needs to calculate the matrix K of Definition 4.3, the active indices $i_1 \ldots, i_r$ of Definition 4.1 and the inverted matrix $(B')^{-1}$ of Proposition 4.2. If these three components are downloaded into the STT then it can decrypt any program $\mathbf{p} \in U$ using the *Decrypt-V* procedure of Figure 1.

Input: the program CID $\mathbf{p}$.
Secret information: K, $(B')^{-1}$, and the active indices $i_1 \ldots, i_r$.

1. $\mathbf{p}' \leftarrow$ the entries $p_{i_1} \ldots, p_{i_r}$ of $\mathbf{p}$.
2. $\mathbf{x} \leftarrow (B')^{-1}\mathbf{p}'$.
3. $Dec \leftarrow K\mathbf{x}$.
4. Decrypt program $\mathbf{p}$ using the key *Dec*.

Figure 1: The *Decrypt-V* procedure.

Proposition 4.4 *Procedure Decrypt-V decrypts a program $\mathbf{p}$ correctly if and only if $\mathbf{p} \in U$.*

Proof: Assume that $\mathbf{p} \in U$. Then using Definition 4.3 and Proposition 4.2, and plugging (2) and (1) we have that

$$Dec = K\mathbf{x} = MB\mathbf{x} = M\mathbf{p} = Key(\mathbf{p}),$$

thus the decryption key is correct.

For the other direction, assume that $\mathbf{p} \notin U$. Then no solution exists for the linear equation system (2), and in particular the $\mathbf{x}$ computed by *Decrypt-V* is not a solution, i.e., $B\mathbf{x} = \mathbf{z} \neq \mathbf{p}$ for this $\mathbf{x}$. Thus *Decrypt-V* would generate the key $Dec = Key(\mathbf{z})$ which is incorrect for program $\mathbf{p}$. ▮

Remarks:

- If $\mathbf{p} \notin U$ then procedure *Decrypt-V* generates a *valid* key $Key(\mathbf{z})$ to some program $\mathbf{z}$ which is different from $\mathbf{p}$. However $\mathbf{z} \in U$ is a linear combination of basis vectors (whereas $\mathbf{p}$ is not) so the procedure does not generate keys that the user is not entitled to.

- In order to make the decryption more efficient, it is possible merge the secret components K, $(B')^{-1}$ and $i_1 \ldots, i_r$ into a single $k \times n$ matrix D. In this representation procedure *Decrypt-V* computes the key by a single matrix multiplication $Dec \leftarrow D\mathbf{p}$.

4.4.2 The Check Matrix

For any program $\mathbf{p}$ the *Decrypt-V* procedure of the previous section generates a key and attempts to decrypt the program. However if $\mathbf{p}$ is outside the entitled subspace U then the key is incorrect and the program will not be decrypted. Thus the STT cannot distinguish between programs that fail to be decrypted due to transmission errors and those that fail because they are outside the subspace U. This may be undesirable since the STT cannot give the user any meaningful feedback regarding the cause of the problem.

To amend this situation, the *check matrix* can be added to the entitlement. With this extra information the STT can easily check if the selected program $\mathbf{p}$ is in the entitled subspace U or not, without attempting to decrypt it. In the field of error correcting codes [MS77] such a matrix is known as a *parity check matrix.*

Definition 4.5 *Let B be a basis matrix for a subspace U. Let B'' be an $r \times n$ matrix whose active index columns $i_1 \ldots, i_r$ contain the columns of $(B')^{-1}$, and is 0 everywhere else. Let I be the n-dimensional unit matrix. Then the $n \times n$* check matrix *C is*

$$\boxed{C = BB'' - I.}$$

The test for the decrypt-ability of a program with a CID $\mathbf{p}$ is based on the following proposition.

Proposition 4.6 $C\mathbf{p} = \mathbf{0}$ *if and only if* $\mathbf{p} \in U$.

Proof: If $\mathbf{p} \in U$ then it is easy to see that $\mathbf{x} = B''\mathbf{p}$ is the solution to the linear system of equations (2). Therefore by linearity we have that

$$C\mathbf{p} = BB''\mathbf{p} - I\mathbf{p} = B\mathbf{x} - \mathbf{p} = \mathbf{0}.$$

Conversely, if $\mathbf{p} \notin U$ then $B\mathbf{x} \neq \mathbf{p}$ for all $\mathbf{x}$, and in particular $B\mathbf{x} - \mathbf{p} \neq \mathbf{0}$ for $\mathbf{x} = B''\mathbf{p}$. ∎

Remark: If the entitled subspace U is the space of *all* CIDs, then any basis matrix B is an n-dimensional regular matrix in itself, and thus $B'' = B^{-1}$ and $BB'' = I$. So the check matrix C becomes all zero and the test of Proposition 4.6 always succeeds, as expected.

5 How to Assign CIDs to the Programs

For the Vspace scheme to be useful, the CIDs cannot be assigned to programs arbitrarily. Care must be taken so that the CIDs of programs with related content fit into low dimensional linear subspaces. In this section we present one systematic method of assigning CIDs to programs which achieves this goal.

```
Input: a topic v in the hierarchy.
----------------------------------------------
Let ST(v) denote the set of sub-topics of v.
Let label(v) be the label assigned to v and let
   mask(v) be its mask.
----------------------------------------------
Choose a label length ℓ such that 2^ℓ ≥ |ST(v)|.
for each u ∈ ST(v) do
      label(u) ← ℓ-bit value in the range [0, 2^ℓ − 1].
      mask(u) ← mask(v) || label(u).
      if u is not a leaf then
            MakeMasks(u).
end-for
```

(i) The *MakeMasks*(v) procedure.

```
[2 bits]
    0: Bonus ...
    1: Movies ...
    2: Sports [5 bits]
        2.0: Baseball ...
        2.1: Football ...
        2.2: Basketball [2 bits]
            2.2.0: College
            2.2.1: Professional
            2.2.2: European
            2.2.3:
        ...
        2.31:
    3: Other ...
```

(ii) An example topic hierarchy.

Figure 2: The *MakeMasks* procedure (i), and the masks it assigns to a topic hierarchy (ii). The decimal numbers next to the topics represent their mask values, and the label lengths appear in brackets. Thus the prefix mask for "2.2.1 Professional Basketball" consists of the nine bits 10 00010 01.

5.1 The Topic Hierarchy

We assume that the programs can be naturally organized in a hierarchy according to attributes such as their subject, language, rating, source, etc. We call this hierarchy (or tree) the *topic hierarchy*.

We assign CIDs to programs in the hierarchy using the notion of *prefix masks*. Procedure *MakeMasks* of Figure 2(i) recursively assigns the prefix masks by labeling the topics from the root towards the leaves. The prefix mask of every topic is its own label concatenated to the mask of its parent, or equivalently, its own label concatenated to the labels of all its ancestors. An example of a part of a topic hierarchy with the masks generated by the *MakeMasks* procedure appears in Figure 2(ii).

Remarks:

- A label may be longer than the minimal number of bits required for labeling all the sub-topics. For instance, consider a topic X at the lowest level of the tree, whose sub-topics are individual programs. If the cumulative length of X's mask is m bits then we may assign all the remaining $n - m$ bits for sub-topic (i.e., program) labels, even if there are fewer than 2^{n-m} such programs.

- The labels given to sub-topics need not be consecutive. It may sometimes be beneficial to leave some labels unused.

- The top level topic with label $\mathbf{0}$, which is called "Bonus" in Figure 2(ii), has a special role. The reason for this will become clear in the next section, when we see how to generate a basis from the prefix masks. Informally, users who buy some package like "all the sports programs" will also be entitled to some of the programs appearing in the Bonus hierarchy.

5.2 Computing a Basis from the Prefix Masks

5.2.1 Single-Topic Packages

We start by addressing the case of a package which contains all the programs in some topic X. Then the labeling done by procedure *MakeMasks* is such that the CIDs of all these programs share the same prefix mask, which is the mask assigned to topic X itself. So our goal here is to show how to generate a basis for a package of CIDs sharing a prefix μ. Formally,

Definition 5.1 *Let μ be an m-bit mask. Then $U_\mu = \{\mu \| *\}$ is the package of all CIDs with the prefix μ.*

Note that in general such a set U_μ of CIDs sharing a prefix μ is *not* a linear subspace. This may be easily seen from the fact $\mathbf{0} \notin U_\mu$ if $\mu \neq \mathbf{0}$. However when we extend U_μ to include some "bonus" programs, then the extended package becomes a linear subspace. So when Bob buys a package of "all the sports programs", his entitlement will also give him access to certain bonus CIDs. For a package with a mask μ of length m the bonus CIDs are those for which the m most significant bits are zero. Formally,

Definition 5.2 *Let $\mu \neq \mathbf{0}$ be an m-bit mask, and let $\mathbf{0}_m$ be a string of m 0-bits. Then the* bonus-extended *package with prefix μ is $\hat{U}_\mu = \{\mu \| *\} \cup \{\mathbf{0}_m \| *\}$, and programs with a CID of the form $\mathbf{0}_m \| *$ are called* bonus programs.

The next Definition 5.3 gives an explicit construction of a basis B_μ for a bonus-extended package $\hat{U}_\mu$. Then in Proposition 5.4 we prove that $Span(B_\mu) = \hat{U}_\mu$. By this we show that $\hat{U}_\mu$ is indeed a linear subspace, and that Definition 5.3 gives us a simple method of constructing a basis for it. Moreover, the proof also shows that the entitlement for a package with a prefix μ gives access precisely to the set U_μ and its bonuses, and to nothing else.

Definition 5.3 *Let $\mu \neq \mathbf{0}$ be an m-bit mask. Let $\mathbf{e}_1, \ldots, \mathbf{e}_n$ denote the standard basis where $\mathbf{e}_i$ has a 1-bit in position i. Define the* enabling vector *$\mathbf{z}$ to have μ as its prefix followed by $(n-m)$ 0-bits, i.e., $\mathbf{z} = \mu \| \mathbf{0}_{n-m}$. Define the basis B_μ to be*

$$\boxed{B_\mu = \{\mathbf{z}, \mathbf{e}_{m+1}, \ldots, \mathbf{e}_n\}.}$$

Remark: In this section we view the basis B_μ as a *set of vectors* rather than as a *matrix*. The corresponding $n \times (n-m+1)$ basis matrix has the vectors $\mathbf{z}, \mathbf{e}_{m+1}, \ldots, \mathbf{e}_n$ as its columns.

Proposition 5.4 *$Span(B_\mu) = \hat{U}_\mu$, thus $\hat{U}_\mu$ is a linear subspace of dimension $n-m+1$ and B_μ is a basis for it.*

Proof: Clearly the linear combinations of the standard vectors $\mathbf{e}_{m+1}, \ldots, \mathbf{e}_n$ span every vector of the form $\mathbf{0}_m \| *$, so B_μ spans all the bonus programs.

Now the enabling vector $\mathbf{z}$ has 1-bits only in the m most significant bits, which comprise the mask μ. Any bit pattern in the lower $n - m$ bits can be written as a linear combination of the standard vectors $\mathbf{e}_{m+1}, \ldots, \mathbf{e}_n$. Hence every CID $\mathbf{p}$ of the form $\mu \| *$ can be written as a combination of $\mathbf{z}$ and some standard vectors. We conclude that $\hat{U}_\mu \subseteq Span(B_\mu)$.

For the other direction, consider some linear combination $\mathbf{p}$ of vectors in B_μ. Then the m-bit prefix of $\mathbf{p}$ is either μ (if the combination includes the enabling vector $\mathbf{z}$) or $\mathbf{0}_m$ (if $\mathbf{z}$ is not included). Thus $\hat{U}_\mu \supseteq Span(B_\mu)$.

The vectors in B_μ are clearly independent, so the dimension of $\hat{U}_\mu$ is $n - m + 1$. ∎

Remarks:

- The mask length of a topic is *not* determined by the position of the rightmost 1-bit in the mask, but rather by the length of the labels assigned to the topic hierarchy. In the example of Figure 2(ii), the "2.2 Basketball" package has a 7-bit mask 10 00010, while the "2.2.0 College Basketball" package has a 9-bit mask 10 00010 00, and both these masks lead to an identical enabling vector. However the bases for the two packages differ in the standard vectors they contain; the "College Basketball" basis is $(n - 8)$-dimensional and does not contain $\mathbf{e}_8$ and $\mathbf{e}_9$.

- Programs placed in the bonus hierarchy are shared by many packages. A package characterized by *any* prefix mask of m bits also contains the bonus CIDs of the form $\mathbf{0}_m \| *$. The bonuses added to a package depend only on the *length* of the package's prefix mask. Therefore we may speak of "m-bit bonuses", which contain all the bonus CIDs whose prefix is at least m 0-bits.

- If there is no need for bonus programs, it is always possible not to use CIDs from the bonus hierarchy at all. Then the only cases in which two packages share some programs (other than plaintext programs with CID $\mathbf{p} = \mathbf{0}$) is if one package contains the other.

5.2.2 Multi-Topic Packages

So far we have seen how to create a basis for a package of the form "all the programs which belong to topic X". The case of a package which is the union of several topics is similar, with some limitations.

Suppose Alice wants to buy all the programs belonging to topics $X_1, \ldots, X_t$ with masks $\mu_1, \ldots, \mu_t$. We start by generating the t corresponding bases $B_{\mu_1}, \ldots, B_{\mu_t}$, as in Definition 5.3. Then we build a new basis B by repeatedly including the next vector from $B_{\mu_1} \cup \cdots \cup B_{\mu_t}$ which is independent of all the vectors already in B. Checking if a vector $\mathbf{p}$ is independent of the vectors currently in B can be done by solving a system of linear equations [BM77]. It is not difficult to see that the basis B generated in this manner does indeed span all the programs belonging to the requested topics.

However, there may be *side-effects* to this procedure. This is since in general the union of linear subspaces is not a linear subspace. The computed basis B is the basis of a linear subspace that contains all the requested topics, and parts of the bonus hierarchy, but it may also contain other (unrequested) parts of the topic hierarchy.

Therefore there can be two different approaches to using the topic hierarchy. The simpler but more restrictive approach is to offer only packages which correspond to a single topic in the hierarchy, and avoid the complexities of possible side-effects. The second approach is to let users choose several topics in the hierarchy, and to have the system compute the subspace of programs that would actually be accessible with all the side-effects. The users then need to be informed of what they would really be getting (and paying for).

6 The Security of the Schemes

When we consider the security of the ExtHeader and Vspace key management schemes, there are two models of attack that need to be addressed. In the more optimistic model, the secret information is stored in a truly tamper-proof chip inside the STT. Therefore a pirate can only mount an attack using the transmission itself. This model of attack is discussed in Section 6.1.

However experience shows that pirates are very successful in breaking into such “tamper-proof” STTs. In fact only a handful of the successful attacks described in [McC96] can be classified as crypt-analytical attacks, and only against analog equivalents of simple substitution ciphers. All the rest exploit breaches in the “secure” chip. Therefore in Section 6.2 we discuss the consequences of the the pirates’ ability to break into the secure chip.

6.1 Tamper-Proof Set-Top Terminals

If the STT contains truly secure memory, then our main concern is to protect against known-plaintext attacks. We argue that chosen-plaintext attacks are less likely since they would require a pirate to cause the encryption and broadcast of specific content, and such activities may be noticeable. On the other hand it is reasonable to assume that a pirate would have access to un-encrypted versions of programs, from sources such as video libraries or TV re-runs. Thus the encryption algorithm used to encrypt the programs must be resistant to known-plaintext attacks. In addition the decryption algorithm must be efficient enough to support real-time video decoding.

6.1.1 The ExtHeader Scheme

Here the pirate can also mount an attack against the covering algorithm, used to encrypt the program keys. The headers data attached to the programs contains: (a) Multiple encryptions of the same program key $K_{\mathbf{p}}$ with different package keys, and (b) multiple encryptions of different program keys with the same package key.

Therefore a stream cipher whose basic operation is a XOR with a pseudo random string seems to be unsuitable as a covering algorithm. This is because the pirate would be able to obtain either XORs of program keys, or XORs of the pseudo random strings by XORing together various encryptions. Candidates for the covering algorithm may be DES [DES77] or Triple-DES [Tri85].

Besides choosing a good covering algorithm, another measure we can take is to pad the program keys with random strings, and then cover the padded key. This would increase the header size in proportion to the length of the padding. However problem (a) then becomes a smaller concern since each header block of a program $\mathbf{p}$ would cover $K_{\mathbf{p}}$ padded with *different* random string.

6.1.2 The Vspace Scheme

Here our concern is that the pirates may exploit the linearity of the scheme. If the program encryption preserves the linearity of the key management, then a pirate may be able to track the influence of each bit of $Key(\mathbf{p})$ on the encrypted program. Then using a known-plaintext attack, the pirate may be able to write linear equations with the key bits as variables. If k such equations are collected then the system of equations can be solved and $Key(\mathbf{p})$ can be found. If r program keys are broken, for programs with linearly independent CIDs, then the pirate can decrypt a subspace of programs of dimension 2^r essentially by using procedure *Decrypt-V* of Figure 1. Thus every program can be decrypted if the keys of n linearly independent programs are broken.

Therefore we must ensure that the encryption algorithm does not preserve linearity. For this reason, algorithms based on linear feedback shift registers (cf. [Gol67]) may not be good choices. Algorithms based on the discrete log problem (cf. [Odl85]) may be inappropriate as well, since in some sense the linearity is preserved in the exponents, namely $g^a g^b = g^{a+b}$.

Engineering practices may be such that the system is designed in a modular way and the choice of video encryption algorithm is made independently from the choice of the key management scheme. If this is the case, the key management scheme should not output $Key(\mathbf{p}) = M\mathbf{p}$ as the key for program $\mathbf{p}$, since the video encryption algorithm may preserve some linearity. Rather, the encryption key should be $h(Key(\mathbf{p}))$ where $h(x)$ is a member of a universal family of hash function (cf. [Lub96, ch. 8]). Practical choices may be MD5 [Riv92] or SHA [SHA93]. By this the linearity is destroyed before the key is used in the fast video encryption algorithm. This is also important in order to prevent pirates who buy a package which is a subspace of dimension r from learning all the keys in the subspace by extracting only r keys from the secure module (i.e., in order to build a pirate decoder they would need to break into the secure memory).

6.2 When the “Tamper-Proof” Chip is Compromised

If the pirates are able to break into the secure memory of an STT, then clearly the security of the system is breached. We may assume that if the pirates break into Alice's STT then they learn all the secrets stored in that STT, and in particular they can decrypt all the programs that Alice is entitled to.[2] Our specific concern here is to ensure that the pirates will not be able to decrypt programs that Alice is *not* entitled to.

[2] Several measures are suggested in the Lucent IVES™ architecture [CEFH95] to qualify this statement. For instance, if the entitlements are stored encrypted, using an encryption function specific to each individual STT, then simply copying the entitlement of one STT to another would be insufficient to clone the first one.

6.2.1 The ExtHeader Scheme

In the ExtHeader scheme, knowledge of some package keys in itself does not help in breaking another package key, since these keys are chosen completely independently of each other. However if the pirates know a key s_i of some package i then they can mount a known-plaintext attack on the covering algorithm. If some program $\mathbf{p}$ belongs to the exposed package i then the pirates can now uncover its key $K_{\mathbf{p}}$. This $K_{\mathbf{p}}$ becomes the plaintext that is covered by the key of every other package that $\mathbf{p}$ belongs to. So in addition to the requirements mentioned in the Section 6.1.1, we also need to ensure that the covering algorithm is resistant to known-plaintext attacks.

6.2.2 The Vspace Scheme

In the Vspace scheme the situation is marginally more delicate when an STT is compromised. If Alice is entitled to a subspace of programs of dimension r, then her key matrix K is of dimension $k \times r$. If the pirates learn this K, this knowledge is equivalent to knowing r of the master keys (r columns of M). Knowing r master keys can help the pirate in obtaining keys to a program $\mathbf{p}$ that Alice is not entitled to, however the pirate's advantage is negligible.

If Alice is not entitled to $\mathbf{p}$, then $Key(\mathbf{p})$ is a linear combination of master keys, at least one of which is unknown to the pirate. However since the master keys are chosen so they are linearly independent (recall Section 4.2), the size of the enumeration space for $Key(\mathbf{p})$ is slightly smaller: $2^k - 2^r$ rather than 2^k. The pirate is in the best situation if $r = n - 1$ (only one master key is missing): Then the enumeration space is of size $2^k - 2^{n-1}$, and if $k \gg n$ (say $k = 2n$) then the 2^{n-1} term is negligible. Thus we see that the system security is essentially intact despite the break-in.

If the pirates are able to break into other STTs that are entitled to different subspaces then they may collect more master keys. Then the subspace of programs they would be able to decrypt would be larger than the union of the broken STTs' entitlements. Note that in any key management scheme, if the pirates manage to break into STTs of users who together are entitled to all the programs then the pirates too can decrypt all the programs. The difference is that in the Vspace scheme the pirates need only to break into STTs of users whose entitlement bases together contain n linearly independent vectors.

7 The Addressing Power of Vspace

The addressing power ($\mathcal{AP}$) of a key management scheme is the number of different packages of programs that the vendor can offer. This number should be parameterized by the number of secret bits stored in the STT (assuming the scheme does not add header information to the broadcast). If a scheme X needs b secret bits stored at the STT, then the best we could hope for is the ability to offer 2^b different packages, i.e., $\mathcal{AP}(\mathrm{X}) = 2^b$.

In the Vspace scheme, the STT stores kn bits thus we could hope for an addressing power of $\mathcal{AP}(\mathrm{Vspace}) = 2^{kn}$. Clearly not every possible setting of the secret bits is valid since the user is only allowed to buy linear subspaces of programs. Nevertheless, we show that the number of packages a user can choose from is close to the maximal value. The next proposition shows that

$\mathcal{AP}(\text{Vspace}) = 2^{\Theta(n^2)}$, which is optimal (up to constants in the exponent) when k and n are of the same order of magnitude.

Proposition 7.1 $\mathcal{AP}(\text{Vspace}) = 2^{(1+o(1))n^2/4}$.

Proof: The number of d-dimensional linear subspaces in an n-dimensional vector space over a field of 2 elements is called the *Gaussian binomial coefficient* and is denoted by $\begin{bmatrix} n \\ d \end{bmatrix}$. In [GR69] and [MS77, pp. 443–445] it is shown that

$$\begin{bmatrix} n \\ d \end{bmatrix} = \frac{(2^n-1)(2^{n-1}-1)\cdots(2^{n-d+1}-1)}{(2^d-1)(2^{d-1}-1)\cdots(2-1)}.$$

For the upper bound, note that the Gaussian coefficient is maximized when $d = n/2$, so

$$\mathcal{AP}(\text{Vspace}) = \sum_{d=1}^{n} \begin{bmatrix} n \\ d \end{bmatrix} \le n \begin{bmatrix} n \\ n/2 \end{bmatrix}.$$

Since for any $2 \le s \le n$ we have $\frac{2^n-1}{2^s-1} \le \frac{2^{n-1}-1}{2^{s-1}-1}$ we obtain that

$$n \begin{bmatrix} n \\ n/2 \end{bmatrix} = n \frac{(2^n-1)(2^{n-1}-1)\cdots(2^{n/2+1}-1)}{(2^{n/2}-1)(2^{n/2-1}-1)\cdots(2-1)} \le n(2^{n/2+1}-1)^{n/2} \le 2^{n^2/4+n/2+\log n}.$$

For the lower bound, note that for $n \ge 2$ and $s < n/2$ we have that $\frac{2^{n-s}-1}{2^{n/2-s}-1} \ge 2^{n/2-1}-1$. Therefore

$$\mathcal{AP}(\text{Vspace}) = \sum_{d=1}^{n} \begin{bmatrix} n \\ d \end{bmatrix} \ge \begin{bmatrix} n \\ n/2 \end{bmatrix} \ge (2^{n/2-1}-1)^{n/2} \ge 2^{n^2/4-n}. \qquad \blacksquare$$

8 Conclusions and Future Work

We have seen that the simple ExtHeader scheme has excellent flexibility, at the cost of extra header data that needs to be transmitted. In comparison, the Vspace scheme requires essentially no additional headers and still enjoys high flexibility, however careful design decisions must be made to ensure the security of the system. Both schemes are practical, and offer better capabilities than schemes in current use. Two directions for future work are the following.

- In Section 6 we suggested some informal guidelines to make the schemes secure against various attacks. An important addition would be to supply a formal *proof* of their security.

- The attractive features of the Vspace scheme rely on the linearity of the scheme. However at the same time we need to be careful that this linearity does not compromise the security of the scheme. Can we design an efficient scheme which has high flexibility and low additional transmission overhead, yet does not rely on linearity?

Acknowledgments

I am grateful to Dan Heer for introducing me to the problems of broadcast encryption, and to Peter Winkler for encouraging me to work on them. Daniel Bleichenbacher made numerous suggestions about the cryptographic properties of these systems. Shimon Even brought Gaussian coefficients to my attention. Remarks made by Avi Silberschatz led to the idea of caching headers in the ExtHeader scheme. I thank Amos Fiat and several anonymous referees whose comments helped me improve the presentation.

References

[ARV95] W. Aiello, S. Rajagopalan, and R. Venkatesan. Design of practical and provably good random number generators. In *Proc. ACM-SIAM Symp. on Discrete Alg. (SODA)*, pages 1–9, 1995.

[BC94] C. Blundo and A. Cresti. Space requirements for broadcast encryption. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT'94, LNCS 950*, pages 287–298. Springer-Verlag, 1994.

[Ber91] S. Berkovits. How to broadcast a secret. In *Advances in Cryptology – EUROCRYPT'91, LNCS 547*, pages 535–541. Springer-Verlag, 1991.

[BFS96] C. Blundo, L. A. Frota Mattos, and D. R. Stinson. Trade-offs between communication and storage in unconditionally secure schemes for broadcast encryption and interactive key distribution. In *Advances in Cryptology – CRYPTO'96, LNCS 1109*, pages 387–400. Springer-Verlag, 1996.

[BM77] G. Birkhoff and S. Mac Lane. *A Survey of Modern Algebra.* Macmillan Publishing Co., 4th edition, 1977.

[CC89] G. H. Chiou and W. T. Chen. Secure broadcasting using secure lock. *IEEE Trans. Software Eng.*, 15(8):929–934, 1989.

[CEFH95] J. L. Cohen, M. H. Etzel, D. W. Faucher, and D. N. Heer. Security for broadband digital networks. *Communications Technology*, pages 58–69, August 1995.

[CT90] G. C. Chick and S. E. Tavares. Flexible access control with master keys. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89, LNCS 435*, pages 316–322. Springer-Verlag, 1990.

[DES77] Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977.

[FN94] A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology – CRYPTO'93, LNCS 773*, pages 480–491. Springer-Verlag, 1994.

[FS96] J.-B. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96, LNCS 1070*, pages 245–255. Springer-Verlag, 1996.

[Gol67] S. W. Golomb. *Shift Register Sequences.* Holden-Day, San Francisco, 1967. Reprinted by Aegean Park Press, 1982.

[Gon94] L. Gong. New protocols for third-party-based authentication and secure broadcast. In *Proc. 2nd ACM Conf. Comp. and Comm. Security*, pages 176–183, 1994.

[GR69] J. Goldman and G.-C. Rota. The number of subspaces of a vector space. In W. Tutte, editor, *Recent Progress in Combinatorics*, pages 75–83. Academic Press, 1969.

[HP95] S. Halevi and E. Petrank. Storing classified files. Manuscript, 1995. Electronic Publication: `ftp://theory.lcs.mit.edu/pub/people/shaih/classify.ps.gz`.

[IN89] R. Impagliazzo and M. Naor. Efficient cryptographic schemes as secure as subset sum. In *Proc. 30th IEEE Symp. Foundations of Comp. Sci. (FOCS)*, pages 236–241, 1989.

[Lub96] M. Luby. *Pseudorandomenss and Cryptographic Applications.* Princeton Universtity Press, Princeton, New Jersey, 1996.

[McC96] J. McCormac. *European Scrambling Systems 5.* Waterford Universtity Press, Waterford, Ireland, 1996.

[MPE94] MPEG-2: Coding of moving pictures and associated audio. ISO/IEC CD 13818-1, 1994.

[MQ95] B. M. Macq and J.-J. Quisquater. Cryptology for digital TV broadcasting. *Proceedings of the IEEE*, 83(6):944–957, 1995.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes.* North Holland, 1977.

[Odl85] A. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Advances in Cryptology – EUROCRYPT'84, LNCS 209*, pages 224–314. Springer-Verlag, 1985.

[Riv92] R. L. Rivest. The MD5 message digest algorithm. RFC 1321, April 1992.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.

[SHA93] Secure hash standard. National Institute of Standards and Technology, NIST FIPS PUB 180, U.S. Department of Commerce, May 1993.

[Tri85] ANSI X9.17 (revised). American National Standard for Financial Institution Key Management (Wholesale), American Bankers Association, 1985.