Key Management for Encrypted Broadcast

AVISHAI WOOL
Lucent Technologies

We consider broadcast applications where the transmissions need to be encrypted, such as direct broadcast 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 service provider 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 two novel schemes to manage the encryption keys for these applications. The schemes are highly flexible, and understandable to users, yet require very few keys to be stored in the STTs' secure memory. The computational power required of the STTs is very low. The security of these schemes is as good or better than that offered by current technology.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Smartcards*; E.3 [**Data**]: Data Encryption

General Terms: Security

Additional Key Words and Phrases: Conditional access, pay-per-view

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 direct broadcast 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-[Macq and Quisquater 1995].

A preliminary version of this paper appeared in the 5th ACM Conference on Computer and Communication Security (1998).
Author's address: Bell Laboratories, Lucent Technologies, 700 Mountain Ave., Murray Hill, NJ 07974; email: yash@research.bell-labs.com.

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, either directly or on a smart-card, 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 Kilobyte [Gem 1998]. 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. Another aspect of a scheme's flexibility is the ability to make the packaging simple and understandable in user terms.

(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" of the STTs. This is necessary 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.

Before presenting the new schemes, we start by describing a simple "straw-man" scheme called ExtHeader. This is a variant of the well known "encrypt the session key with the user key" technique. Despite its simplicity, the scheme is apparently not in current use by pay-TV providers. The most commonly used schemes seem to be significantly less secure than the ExtHeader scheme. 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.

The contributions of this paper are the Vspace scheme and the Tree scheme. Both schemes 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., in the Vspace scheme packages can only be of 1, 3, 7, 15, . . . , $2^n - 1$ programs). Among the two schemes, Vspace is the speedier one: In a typical configuration the STT can compute a Vspace key by 32 XOR operations of 128-bit words. The Tree scheme, which is also very efficient (but requires somewhat more CPU power than Vspace) is more secure: It has the added advantage that it is *provably* secure if we make some assumptions about its central component, which is a cryptographic hash function.

Note that if a pirate is able to pry open an STT which is entitled to all the programs and to extract the secret information from it, then our schemes offer the same level of security offered by existing schemes. However, our schemes are stronger in that they do not allow the pirate to "upgrade" an STT from a cheap package to an expensive one by low cost chip rewriting attacks (cf. Anderson and Kuhn [1997]). In our schemes, if an STT is not entitled to decode a certain program, then the decryption key does not exist in the STT—it is not merely "masked off", as is the case in existing schemes. In other words, we prevent the type of tampering recently publicized in the Internet browser arena, in which a few bits in the executable files of browsers with "international-grade" security were patched to activate dormant "US-grade" security code [Fort 1998], thus bypassing US export restrictions. Moreover, unlike current schemes, breaking into an STT that is not fully entitled does not compromise our scheme completely.

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 Tree scheme is presented in Section 6. The security of all three schemes is discussed in Section 7. In Section 8, we analyze the addressing power of the Vspace and Tree schemes, and we conclude with some directions for future work in Section 9.

2. RELATED WORK

2.1 The Bit-Vector Scheme

This is the most popular access control scheme in current use. It is used by most of the analog European satellite TV systems such as the Sky VideoCrypt systems [McCormac 1996], and also by the US digital DirectTV system [Dir 1998; McCormac 1996].

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 additional data, which we can abstractly view as a *bit-vector* **b** which has an entry for each program. The STT decrypts a program **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 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. Pirates have reportedly taken advantage of this weakness in existing systems [McCormac 1996]. Secondly, the scheme needs to store the bit-vector in secure memory. Therefore, the size of the bit-vector may become prohibitive when the number of programs is large.

2.2 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 [McCormac 1996, pp. 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 reencrypted with multiple block keys, and retransmitted 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.

2.3 Types of Keys and User Revocation

Pay-TV systems typically have multiple types of keys. The key types differ according to the length of time that the key material needs to be used. The three main key types are: Establishment keys (that can only be refreshed by replacing the smartcard), periodic keys (that control access during a billing period, say a month), and program keys (that encrypt a single program) [Quisquater 1998].

The longest-lived keys are the *establishment keys*. These are keys that are burned into the STT, and need to remain valid and secure for the life of the STT (or, more commonly, for the life of the smartcard). Replacing such keys is expensive to the provider, so their lifetime is measured in many

months or even years. Thus the allocation of these keys needs to give the provider the ability to revoke or exclude users, and collusions of users, without the need to replace paying customers' smartcards. A large body of literature deals with methods of allocating these establishment keys to users, starting with the seminal work of Fiat and Naor [1994], and extended in Blundo and Cresti [1994]; Blundo et al. [1998]; Luby and Staddon [1998]; Abdalla et al. [2000]; Garay et al. [2000], and others.

Between the long lived establishment keys and the individual program keys we have *periodic keys*. These are keys that control the access to the content that is broadcast in a particular billing period (say a month). Access to the actual program keys is only available to users with the current periodic key. The periodic keys are distributed to paying customers using the long-lived establishment keys. Revoking non-paying users is done by simply not giving them next month's key. This revocation method does not provide backward security, and is not immediate (a user can only be revoked at the end of the month), however, it is very cheap to implement. The remaining, paying, customers are not impacted at all when users are revoked. Recently, there has been significant work on schemes that provide much tighter revocation capabilities, which are appropriate for small user groups (cf. Moyer et al. [1999]). However, the simpler scheme based on periodic keys is much better suited for the huge subscriber bases of pay-TV systems (see Briscoe [1999] for more details). In this paper we deal primarily with the organization and usage of the periodic keys.

2.4 How to Transfer Keys to the STTs?

A related question is how to transfer the periodic 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 unidirectional broadcast. This model is accurate for existing European systems. Under this model, Fiat and Naor [1994] 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. The new periodic key is encrypted using a subset of the establishment keys, that legitimate paying users have, and is broadcast to all the users. However, excluded users will not be able to decrypt the new periodic key since they lack the necessary establishment keys.

In the second model, the STTs also have an uplink capability (e.g., via a phone line or cable-modem). This is realistic since most new U.S.-based systems have such a "callback" feature. In the design described in Cohen et al. [1995], the head-end runs a secure and authenticated protocol with every STT each billing period. 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 unidirectional communication in that it limits a pirate's ability to obtain secret keys: They are

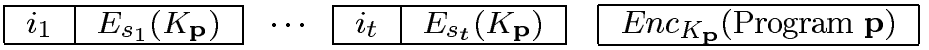

Fig. 1. The headers of a program $\mathbf{p}$ which belongs to packages $i_1, \ldots, i_t$.

never broadcast. The architecture also allows additional security checks to be piggybacked onto the key download, such as tracking the location of the STT [Gabber and Wool 1999].

2.5 Alternative Approaches

Key management schemes which have full flexibility (i.e., allowing users to access any set of programs they wish) can be found in Chick and Tavares [1990] and Halevi and Petrank [1995]. However, their techniques rely on RSA public-key cryptography [Rivest et al. 1978], and therefore are not considered to be applicable in our scenario. This is mostly since the recommended key lengths of public keys, i.e., 1024 bits (cf. Schneier [1996]), would severely tax the limited secure memory of the STT (typically 2–8KB for commercially available smart-card based boxes [Gem 1998]), and since RSA computations are typically not fast enough to control the video access on slow smartcard CPUs.

Protocols for conference key distribution [Chiou and Chen 1989; Berkovits 1991; Gong 1994] 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 (see Figure 1). 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 **p** belongs to t packages, then the scheme requires $t \times (|H.ID| + |H.Key|)$ header bits to be transmitted along with **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 **p** the STT needs to wait until **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.

Moreover, bandwidth allocated to cryptographic headers should not be compared to the bandwidth allocated to programs but rather to that allocated to other types of control information. Direct broadcasting operators are usually reluctant to add more overhead bandwidth since it cannot be billed for, and may potentially decrease the number of TV channels that can be packed into a single satellite transponder [Shamir 1998]. Thus commercial considerations may make this scheme unattractive in its basic form.

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 **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 **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 several seconds.

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 this 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 **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 **p**" or a "program with CID **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

[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.

the ECM [Entitlement Control Message] field defined in the MPEG-2 standard [MPEG2 1994]. 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 Birkhoff and MacLane [1977]. We use boldface letters to denote program CIDs, which we view as n-bit vectors over $GF(2)$[2] We use capital letters to denote matrices over $GF(2)$.

4.2 How it Works

For readers that are comfortable with linear algebra, we provide a succinct description of the scheme's construction. The head-end has a secret random binary $k \times n$ matrix M called the *master matrix*. The keys for encrypting the programs are generated from the master matrix as follows. A program whose CID is $\mathbf{p}$ is encrypted using the key

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

Program packages are constrained to be linear subspaces of program CIDs. A user that buys such a package receives as her entitlement a (personalized, lower-rank) key matrix K, which is computed from the master matrix M by projecting it onto the package's subspace.

In the following sections we shall go over the details of the scheme, explain why it works, and prove that a user holding the matrix K can only decrypt programs whose CIDs fall within the purchased linear subspace.

4.3 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. A program whose CID is $\mathbf{p}$ is encrypted using the key $Key(\mathbf{p})$ of equation (1).

This method of generating keys as linear combinations of the master keys is similar to the techniques used in Impagliazzo and Naor [1989]; Aiello et al. [1995]; Fischer and Stern [1996] to construct provably secure pseudorandom generators. However, a pseudorandom 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. An all-zero key is considered a weak key in some encryption

[2]$GF(2)$ is the single-bit field over $\{0, 1\}$ with bitwise-XOR as the addition operation and bitwise-AND as the multiplication operation.

algorithms (e.g., DES [1977]), and some encryption algorithms (notably shift-register based ones) actually 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

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

For $n = 32$ and $k = 64$ this value is $\approx 1 - 10^{-10}$. Note, also, that it is easy to check whether the master keys are linearly independent.

4.4 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 $Key(\mathbf{p}_1) \oplus \mathbf{Key}(\mathbf{p_2})$, 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.

4.5 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

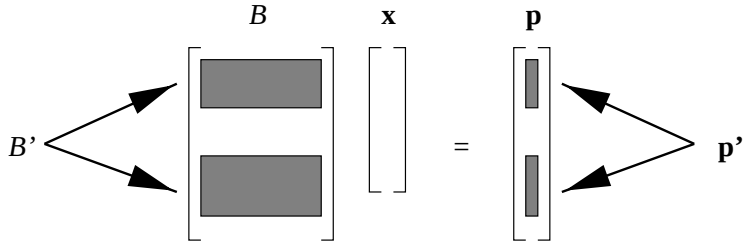

Fig. 2. The linear equation system $B\mathbf{x} = \mathbf{p}$. The regular $r \times r$ submatrix B' and the corresponding $\mathbf{p}'$ are shaded.

head-end needs to generate an entitlement which will allow Alice to decrypt any program $\mathbf{p} \in U$ but nothing else. In Section 4.5.1, 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 decryptable space U without going through the whole decryption procedure. The check matrix is described in Section 4.5.2.

4.5.1 *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) (see Figure 2).

Definition 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 1. *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).

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.

Fig. 3. The *Decrypt-V* procedure.

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 2. Let K be the $k \times r$ matrix

$$K = MB.$$

In order to generate the decryption part of the entitlement, the head-end needs to calculate the matrix K of Definition 4.5.1, the active indices $i_1, \ldots, i_r$ of Definition 4.5.1 and the inverted matrix $(B')^{-1}$ of Proposition 1. 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 3.

PROPOSITION 2. *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 2 and Proposition 1, 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 $Dec \leftarrow D\mathbf{p}$.

4.5.2 *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 [MacWilliams and Sloane 1977] such a matrix is known as a *parity check matrix*.

Definition 3. 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

$$C = BB'' - I.$$

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

PROPOSITION 3. $C\mathbf{p} = \mathbf{0}$ *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 3 always succeeds, as expected.

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 $2^\ell \geq |ST(v)|$.
for each $u \in ST(v)$ **do**
    $label(u) \leftarrow \ell$-bit value in the range $[0, 2^\ell - 1]$.
    $mask(u) \leftarrow mask(v) \,||\, label(u)$.
    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.

Fig. 4. 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. 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.

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 4(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 4(ii).

Remarks.

—A label may be longer than the minimal number of bits required for labeling all the subtopics. For instance, consider a topic X at the lowest level of the tree, whose subtopics 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 subtopic (i.e., program) labels, even if there are fewer than 2^{n-m} such programs.

—The labels given to subtopics need not be consecutive. It may sometimes be beneficial to leave some labels unused.

—The top level topic with label **0**, which is called "Bonus" in Figure 4(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 4. 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 $\mathbf{0} \notin U_\mu$. 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. 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*.

Definition 6 gives an explicit construction of a basis B_μ for a bonus-extended package $\hat{U}_\mu$. Then, in Proposition 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 6 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 6. 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, ie., $\mathbf{z} = \mu \| \mathbf{0}_{n-m}$. Define the basis B_μ to be

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

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 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 4(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-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 6. Then we build a new basis B by repeatedly including the next vector from $B_{\mu_1} \cup \ldots \cup B_{\mu_t}$ 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 [Birkhoff and MacLane 1977]. 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 TREE SCHEME

The next scheme we present, called the Tree scheme, was suggested by Daniel Bleichenbacher [Bleichenbacher 1998]. Its basic structure is that of the construction of a pseudorandom function due to Goldreich et al. [1986]. An arbitrary pseudorandom function would not necessarily allow a service provider to flexibly package programs using very small entitlements. However, we show that the particular construction of Goldreich et al. [1986] gives us precisely the properties we need—with the added benefit of security proofs.

As in the Vspace scheme, the scheme only requires attaching a single n-bit CID to each program' broadcast. The encryption key for a program is a function of its CID and of secret information stored at the head-end. Each STT stores secret information which allows it to compute the keys to all the programs it is entitled to, and to nothing else. And as in the Vspace scheme, the Tree scheme requires that CIDs be assigned carefully so that meaningful packages will have small entitlements.

6.1 The Basic Scheme

To make this paper self-contained, we include a description of the [Goldreich et al. 1986] construction, using the terminology of the application we have in mind. The main building block of the scheme is a cryptographically secure, length-doubling, hash function

$$H : \{0, 1\}^k \rightarrow \{0, 1\}^{2k},$$

where, as before, k is the encryption key length. For convenience, we consider the image of H to be a pair of k-bit values, which we denote by $H(\mathbf{x}) = \langle H_0(\mathbf{x}), H_1(\mathrm{x})\rangle$. We call H_0 the *left* half of H, and call H_1 the *right* half. For instance, when $k = 160$, H could be defined by using SHA-1 [SHA 1995] as $H(x) = \langle \text{SHA-1}(\mathbf{x}||\mathbf{0}),\ \text{SHA-1}(\mathbf{x}||\mathbf{1})\rangle$, where $\mathbf{0}$ and $\mathbf{1}$ are all-zero and all-one bit strings, respectively.

The scheme has a single k-bit master key, denoted by $\mathbf{m}$. Let the bits of CID $\mathbf{p}$ be denoted by $\mathbf{p} = (p_1, \ldots, p_n)$. Then the encryption key for a program with CID $\mathbf{p}$ is defined to be

$$Key(\mathbf{p}) = H_{p_n}(\ldots H_{p_2}(H_{p_1}(\mathbf{m}))\ldots). \tag{3}$$

Alternatively, we can describe this computation in terms of a full n-level binary tree T, which we call the *key tree*. A left edge in the key tree is labeled by 0 and a right edge is labeled by 1, and the programs correspond to the leaves. We say that the label of a node $\mathbf{u} \in T$ is the concatenation of the labels on the edges on the path from the root to u. Thus we can identify the nodes' labels with CIDs. We use $T(u)$ to denote the subtree rooted at node u, or equivalently, to denote the set of program CIDs corresponding to leaves in u's subtree.

We now place the master key m at the root of the tree. We compute the keys for all the other nodes recursively by using the left or right half of H, depending on whether we follow a left or right edge. In particular, this process computes all the program keys as in (3). However, it also allows us to talk about the key at an internal tree node $\mathbf{u}$, which we denote by $Key(\mathbf{u})$.

In Goldreich et al. [1986] the authors prove that if the function H is a pseudorandom bit generator, then the mapping $Key(\mathbf{p}) : \{0, 1\}^n \rightarrow \{0, 1\}^k$, parameterized by the master key $\mathbf{m}$, is a pseudorandom function. See the original paper, or the book [Luby 1996], for precise definitions and proofs.

6.2 Packages and Entitlements in the Tree Scheme

To be useful for our purposes, we need to show how to create small entitlements for many sets of programs, both small and large, using the Tree scheme.

Observe that given $Key(\mathbf{u})$ of an internal node $\mathbf{u}$ at depth $\mathbf{u}$, with a (partial) CID $(u_1, \ldots, u_r)$, it is easy to compute the keys of any program $\mathbf{p}$

in $\mathbf{u}$'s subtree $T(\mathbf{u})$, i.e., any program with a CID of the form $\mathbf{p} = \mathbf{u}\|*$. This is done by activating $H\ n - r$ times and taking the left or right half each time, as dictated by the low order bits of $\mathbf{p}$. Thus $Key(\mathbf{u})$ can serve as an entitlement for all the programs in $T(\mathbf{u})$.

From this observation, we can see how the program packaging is done. Let S be a collection of programs (the *target set*) that needs to be packaged. We first find a minimal set of tree nodes whose subtrees precisely cover the target S. Formally,

$$\mathcal{T}(S) = Z \subseteq T \text{ such that } \bigcup_{\mathbf{u} \in Z} T(\mathbf{u}) = S, \text{ and } |Z| \text{ is minimal.} \qquad (4)$$

The entitlement for the package S is the set of keys held at the nodes of $\mathcal{T}(S)$. As we observed above, this set of keys allows the STT to decrypt exactly the programs in S but nothing else.

Note that, in principle, the Tree scheme can create an entitlement for any arbitrary target set S. This is in contrast to the Vspace scheme, where only packages of specific sizes were at all possible. Nevertheless, if CIDs are assigned arbitrarily then the Tree entitlements may become prohibitively large for the limited secure memory of the STTs.

6.3 Computing the Entitlements

There is an simple algorithm which computes the cover $\mathcal{T}(S)$ of (4) for any given target set S. The algorithm first decomposes the set S into maximal disjoint *intervals* of consecutive CIDs,[3] and then finds a cover for each interval. Let $\mathcal{I}(S)$ denote the number of intervals in S. It is easy to see that computing a cover for a single interval of CIDs requires visiting $O(n)$ tree nodes in a key tree of depth n, so the algorithm's time complexity is $O(\mathcal{I}(S)n)$.

The size of the entitlement for a package similarly depends on the number of intervals of consecutive CIDs in S, and on the tree depth n. This is since the number of nodes in the minimal cover for a single interval is also $O(n)$ (in fact $2n$ nodes always suffice). We summarize this discussion by following result, which we state without proof:

PROPOSITION 5. *Let T be a key tree of depth n. Then the minimal cover $\mathcal{T}(S)$ can be computed in time $O(\mathcal{I}(S)n)$, and its size is $O(\mathcal{I}(S)n)$*

The algorithm is efficient only as long as $\mathcal{T}(S)$ is polynomial in n. Thus a package of "all CIDs with least significant bit=1" would need an entitlement of 2^{n-1} keys, which is quite unrealistic to compute, let alone to store in an STT, unless n is very small. In a realistic scenario, the entitlement size would be limited to some small number of keys, and hence the service

[3] We say that CIDs $\mathbf{p_1}$ and $\mathbf{p_2}$ are consecutive if the integers whose binary representations are $\mathbf{p_1}$ and $\mathbf{p_2}$ are consecutive.

provider would not be able to package every arbitrary subset of programs. We quantify this observation in Section 8.

Therefore, as in the Vspace scheme, care must be taken so programs with related content are assigned CIDs that allow them to be packaged efficiently. Conveniently, the Tree scheme perfectly matches the topic hierarchy and CID assignment we discussed in Section 5.1, where basic packages are of the form "all CIDs with a bit prefix μ." An entitlement for such a single-topic package is a single key in the Tree scheme. Moreover, multitopic packages can be assembled with no side-effects: The entitlement is simply the set of keys for the individual topics that comprise the multitopic package. And we can do away with the restrictions of the "bonus" hierarchy: Unlike Vspace packages, a Tree package defined by a prefix μ does *not* allow the STT to decrypt programs with a **0** prefix of the same length.

6.4 Comparison with the Vspace Scheme

In many ways, the Tree scheme is superior to the Vspace scheme:

—Like Vspace, it only uses a single CID field in the broadcast.

—The Tree scheme is at least as flexible than the Vspace scheme (see Section 8), and the possible packages are easier to understand since there are no "side effects."

—The Tree scheme does not have any vulnerable linear structure.

—Security properties can be proven for it under suitable cryptographic assumptions.

In fact, the only aspect in which the Tree scheme is inferior to the Vspace scheme is in its CPU efficiency: For n-bit CIDs and k-bit keys, an STT using the Vspace scheme needs $O(n)$ XOR operations of k-bit words to compute a key, whereas an STT using the Tree scheme needs to perform $O(n)$ hash function computations on k-bit values to compute the same key. This may or may not be a substantial drawback of the Tree scheme, depending on the processing power of the STT, the function H, and the value of n.

6.5 Variants and Remarks

—The basic tree scheme uses a binary tree, with left and right edges labeled by individual bits of the CID. A possible variation is to use wider trees, with edge selection governed by several bits at once. Instead of the basic construction of Goldreich et al. [1986], we could use other pseudorandom-function constructions, such as Hall et al. [1998]. For instance, a node could have 2^c children, and instead of the two hash functions H_0 and H_1 we would use 2^c hash functions, which would be indexed by blocks of c consecutive bits in the CID. In fact, the number of children a node has can vary with the tree level at which it is located, e.g., top levels of the tree would be binary, and the lower levels would be wider. The

advantage of wider trees is that they are also shallower, thus reducing the number of hash function calls necessary for the computation of a program key. The disadvantage of this approach is that it limits the flexibility of the scheme: In the extreme, a single-level tree with 2^n children implies that the user can either buy a package of all programs, or packages made up of individual program keys, which defeats the whole purpose of a package. Thus there is a tradeoff between the efficiency of key computation, and the flexibility of the scheme.

—A construction similar to the Tree scheme has recently been suggested for applications in which the provider sells access to a multicast for predefined periods of time [Briscoe 1999]. From our perspective, the suggested construction is a special case of the general Tree scheme, in which a package is constrained to be a range of consecutive CIDs.

7. THE SECURITY OF THE SCHEMES

When we consider the security of our 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 7.1.

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 McCormac [1996] 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 7.2 we discuss the consequences of the pirates' ability to break into the secure chip.

7.1 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 unencrypted versions of programs, from sources such as video libraries or TV reruns. 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.

7.1.1 *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 an 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. A candidate for the covering algorithm may be Triple-DES [ANSI 1985].

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.

7.1.2 *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 3. 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. Golomb [1967]) may not be good choices. Algorithms based on the discrete log problem (cf. Odlyzko [1985]) 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 cryptographically strong hash function that destroys linearity. Practical choices may be MD5 [Rivest 1992] or SHA-1 [SHA 1995]. 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).

7.1.3 *The Tree Scheme*. If we assume that the hash function H is a pseudorandom bit generator, then $Key(\mathbf{p})$ is provably a pseudorandom function [Goldreich et al. 1986]. So if the actual function H is cryptographically strong, then the encryption keys would be unpredictable. Thus, if the

pirate only has access to the encrypted program broadcast, the knowledge that the keys were generated using the Tree scheme does not seem to help in breaking the encryption. So essentially the only concern we have is to ensure that the video encryption algorithm is able to withstand known plaintext attacks.

7.2 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.[4] Our specific concern here is to ensure that the pirates will not be able to decrypt programs that Alice is *not* entitled to.

7.2.1 *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 Section 7.1.1, we also need to ensure that the covering algorithm is resistant to known-plaintext attacks.

7.2.2 *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 4.3), the size of the enumeration space for $Key(\mathbf{p})$ is slightly smaller: $2^k - 2^r$. 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

[4]Several measures are suggested in the Lucent IVES™ architecture [Cohen et al. 1995] 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.

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.

7.2.3 *The Tree Scheme*. As before, if H is assumed to be pseudorandom, then a pirate who knows $Key(\mathbf{p})$ provably has a negligible probability of successfully computing a key for any program outside $\mathbf{u}$'s subtree. However, for a concrete realization of H, we need to be concerned about two properties.

One obviously required property is that it is hard to compute the input $\mathbf{x}$ given half of the image $H_0(\mathbf{x})$ or $H_1(\mathbf{x})$. This certainly holds for any cryptographic hash H, which is hard to invert even when *both* halves of the image are known.

The second property H needs to have is that it is hard to compute $H_0(\mathbf{x})$ even when $H_1(\mathbf{x})$ is known, and vice versa. In principle, it may be easier to complete a missing half-key when the other half is known, even if the function H is hard to invert. If this is the case, then a pirate who knows $Key(\mathbf{u})$ for some node $\mathbf{u}$ may be able to compute the key to $\mathbf{u}$'s sibling $\mathbf{v}$, and then to all the programs in $\mathbf{v}$'s subtree $T(\mathbf{v})$.

An advantage of the Tree scheme is that it makes merging pirated entitlements inefficient. Consider a pair of sibling programs $\mathbf{p_1}$ and $\mathbf{p_2}$, and their parent $\mathbf{u}$. Suppose that the pirate knows both $Key(\mathbf{p}_1)$ and $Key(\mathbf{p}_2)$, which are the two halves of $H(Key(\mathbf{u}))$. The pirate still cannot invert H and compute $Key(\mathbf{u})$ since H is a cryptographic hash function. Thus, the merged pirated entitlement would have to contain both $Key(\mathbf{p}_1)$ and $Key(\mathbf{p}_2)$, rather than the more compact $Key(\mathbf{u})$. So breaking into multiple STTs with cheap but different entitlements is not a good strategy for the pirate. The combined entitlement will be very large.

8. THE ADDRESSING POWER OF THE SCHEMES

The addressing power ($\mathcal{AP}$) of a key management scheme is the number of different packages of programs that the service provider 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., $\mathcal{AP}(X) = 2^b$.

8.1 The Vspace Scheme

In the Vspace scheme, the STT stores kn bits, thus we could hope for an addressing power of $\mathcal{AP}(\text{Vspace}) = 2^{kn}$. 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 6. $\mathcal{AP}(\text{Vspace}) = 2^{(1+o(1))n^2/4}$.

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 Goldman and Rota [1969] and MacWilliams and Sloane [1977, pp. 443–445], it is shown that

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

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

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

Since for any $2 \le s \le n$ we have $(2^n-1)/(2^s-1) \le (2^{n-1}-1)/(2^{s-1}-1)$, we obtain that

$$n \begin{bmatrix} n \\ n/2 \end{bmatrix} = n\frac{(2^n-1)(2^{n-1}-1)\ldots(2^{n/2+1}-1)}{(2^{n/2}-1)(2^{n/2-1}-1)\ldots(2-1)}$$

$$\le n(2^{n/2+1}-1)^{n/2} \le 2^{n^2/4+n/2+\log n}.$$

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

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

8.2 The Tree Scheme

In principle, any arbitrary subset of programs can comprise a package in the Tree scheme. However, the entitlement size would be huge. Therefore we are interested in the addressing power of the Tree scheme when entitlements are limited to only d keys, i.e., the STT stores kd bits of secret information. So the best we can hope for is $\mathcal{AP}(Tree) = 2^{kd}$. The next proposition shows that the number of packages a user can choose from is a close $2^{\Theta(nd)}$, which is optimal (up to constants in the exponent) when k and n are of the same order of magnitude.

PROPOSITION 7. $\mathcal{AP}(Tree) = 2^{(1+o(1))nd}$.

PROOF. For a lower bound, note that the Tree scheme can package any subset of d individual programs, so

$$\mathcal{AP}(Tree) \geq \binom{2^n}{d} \geq \left(\frac{2^n}{d}\right)^d = 2^{nd-d\log d+d}.$$

For an upper bound, note that a tree of depth n, with 2^n leaves, has a total of $2^{n+1}-1$ nodes. Any subset of nodes corresponds to a possible entitlement (with some packages counted multiple times). Therefore

$$\mathcal{AP}(Tree) \leq \binom{2^{n+1}}{d} \leq (2^{n+1})^d = 2^{nd+d}. \qquad \square$$

If $d = n$, then we see that $\mathcal{AP}(Tree) = 2^{\Theta(n^2)} = \mathcal{AP}(\text{Vspace})$. Thus, Tree and Vspace have essentially the same addressing power when the entitlement sizes are equal. However, in the Tree scheme, we can have $d > n$, which is meaningless in the Vspace scheme since the dimension of the vector space is n. Thus, Tree has more addressing power since it can utilize larger entitlements.

9. CONCLUSIONS AND FUTURE WORK

We have seen that the "straw-man" ExtHeader scheme has excellent flexibility, at the cost of extra header data that needs to be transmitted. The Vspace scheme requires essentially no additional headers and still enjoys high flexibility and superb efficiency at the STT, however, careful design decisions must be made to ensure the security of the system. The Tree system is more flexible, more secure, and less restrictive in its design, however, it requires somewhat more CPU power from the STT. All the schemes are practical, and offer better capabilities than schemes in current use.

An important aspect of smart-card based systems is their ability to support "impulse buying" of pay-per-view events by storing some digital cash on the smart-card, and deducting from it when the user decides to view a program. We would like to offer this feature without requiring a callback to the head-end to obtain a new key. Is it possible to allow such impulse buying while maintaining the desirable property that keys to which the user is not entitled are not stored in the STT?

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, and suggested the Tree scheme. 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

ABDALLA, M., SHAVITT, Y., AND WOOL, A. 2000. Key management for restricted multicast using broadcast encryption. *IEEE/ACM Trans. Netw. 8*, 4, 443–454.

AIELLO, W., RAJAGOPALAN, S., AND VENKATESAN, R. 1995. Design of practical and provably good random number generators. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, CA, Jan.). ACM Press, New York, NY, 1–9.

ANDERSON, R. AND KUHN, M. 1997. Low cost attacks on tamper resistant devices. In *Proceedings of the International Workshop on Security Protocols* (Cambridge, U.K., Apr.). Springer-Verlag, New York, NY, 125–136.

ANSI. 1985. ANSI X9.17 (revised), American National Standard for Financial Institution Key Management (Wholesale). ANSI, New York, NY.

BERKOVITS, S. 1991. How to broadcast a secret. In *Proceedings of the Conference on Advances in Cryptology: Lecture Notes in Computer Science* (EUROCRYPT'91), D. W. Davies, Ed., vol. 547. Springer-Verlag, New York, NY, 535–541.

BIRKHOFF, G. AND MAC LANE, S. 1977. *A Survey of Modern Algebra*. 4th ed. Macmillan Publishing Co., Inc., Indianapolis, IN.

BLEICHENBACHER, D. 1998. Personal communication.

BLUNDO, C. AND CRESTI, A. 1995. Space requirements for broadcast encryption. In *Advances in Cryptology—EUROCRYPT'94*, A. D. Santis, Ed. Springer-Verlag, New York, NY, 287–298.

BLUNDO, C., FROTA MATTOS, L. A., AND STINSON, D. R. 1998. Generalized Beimal-Chor schemes for broadcast encryption and interactive key distribution. *Theor. Comput. Sci. 200*, 1-2, 313–334.

BRISCOE, B. 1999. MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences. In *Proceedings of 1st International Workshop on Networked Group Communication* (NGC'99, Pisa, Italy, Nov.), L. Rizzo and S. Fdida, Eds. Springer-Verlag, New York, NY.

CHICK, G. C. AND TAVARES, S. E. 1989. Flexible access control with master keys. In *Proceedings of the Conference on Advances in Cryptology* (EUROCRYPT '89), G. Brassard, Ed. Springer-Verlag, New York, NY, 316–322.

CHIOU, G.-H. AND CHEN, W.-T. 1989. Secure broadcasting using the secure lock. *IEEE Trans. Softw. Eng. 15*, 8 (Aug.), 929–934.

COHEN, J., FAUCHER, D. W., ETZEL, M .., AND HEER, D. N. 1995. Security for broadband digital networks. *Commun. Tech.*, 58–69.

DES U.S. DEPARTMENT OF COMMERCE. 1977. Data encryption standard. National Bureau of Standards. NBS FIPS PUB 46.

DIR. 1998. The high-tech behind broadcasting DirecTV. http://www.directv.com/ hardware/ tech.html

FIAT, A. AND NAOR, M. 1994. Broadcast encryption. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology* (CRYPTO '93, Santa Barbara, CA, Aug. 22–26, 1993), D. R. Stinson, Ed. Springer Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 480–491.

FISCHER, J.-B. AND STERN, J. 1996. An efficient pseudo-random generator provably as secure as syndrome decoding. In *Proceedings of the 16th Annual International Conference on Advances in Cryptology* (CRYPTO '96, Santa Barbara, CA, Aug.), N. Koblitz, Ed. Springer-Verlag, New York, NY, 245–255.

FORT. 1998. Fortify for Netscape. http://www.fortify.net.

GABBER, E. AND WOOL, A. 1999. On location-restricted services. *IEEE Network 13*, 6 (Nov/Dec), 44–52.

GARAY, J. A., STADDON, J., AND WOOL, A. 2000. Long-lived broadcast encryption. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology:*

Lecture Notes in Computer Science (CRYPTO '00), vol. 1880. Springer-Verlag, New York, NY, 333–352.

GEM. 1998. Gemplus: Catalog of products and services. http://www.gemplus.com/global_offer/index.htm

GOLDMAN, J. AND ROTA, G. -C. 1969. The number of subspaces of a vector space. In *Recent Progress in Combinatorics*, W. Tuttle, Ed. Academic Press, Inc., New York, NY, 75–83.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *J. ACM 33*, 4 (Oct.), 792–807.

GOLOMB, S. W. 1967. *Shift Register Sequences*. Holden-Day, Inc., San Francisco, CA.

GONG, L. 1994. New protocols for third-party-based authentication and secure broadcast. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (Fairfax, VA, Nov. 2–4), D. Denning, R. Pyle, R. Ganesan, and R. Sandhu, Chairs. ACM Press, New York, NY, 176–183.

HALEVI, S. AND PETRANK, E. 1995. Storing classified files. ftp://theory.lcs.mit.edu/ pub/people/shaih/classify.ps.gz.

HALL, C., WAGNER, D., KELSEY, J., AND SCHNEIER, B. 1998. Building PRFs from PRPs. In *Advances in Cryptology—CRYPTO '98*, H. Krawczyk, Ed. Springer-Verlag, New York, NY, 370–389.

IMPAGLIAZZO, R. AND NAOR, M. 1989. Efficient cryptographic schemes as secure as subset sum. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science* (FOCS '89, Research Triangle Park, NC, Oct. 30–Nov. 1). IEEE Computer Society Press, Los Alamitos, CA, 236–241.

LUBY, M. 1996. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, Princeton, NJ.

LUBY, M. AND STADDON, J. 1998. Combinatorial bounds for broadcast encryption. In *Proceedings of the Workshop on Advances in Cryptology: Lecture Notes in Computer Science* (EUROCRYPT '98, Espoo, Finland), K. Nyberg, Ed., vol. 1403. Springer-Verlag, New York, NY, 512–526.

MACQ, D. M. AND QUISQUATER, J.-J. 1995. Cryptology for digital TV broadcasting. *Proc. IEEE 83*, 6, 944–957.

MACWILLIAMS, F. J. AND SLOANE, N. 1977. *The Theory of Error Correcting Codes*. North-Holland Publishing Co., Amsterdam, The Netherlands.

MCCORMAC, J. 1996. *European Scrambling Systems 5*. Waterford University Press.

MOYER, M. J., RAO, J. R., AND ROHATGI, P. 1999. A survey of security issues in multicast communications. *IEEE Network 13*, 6 (Nov/Dec), 12–23.

MPEG2. 1994. MPEG-2: Coding of moving pictures and associated audio. ISO/IEC CD 13818-1.

ODLYZKO, A. M. 1985. Discrete logarithms in finite fields and their cryptographic significance. In *Proc. of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques* (Paris, France, Apr. 9-11, 1984), T Beth, N Cot, and I Ingemarsson, Eds. Springer-Verlag, New York, NY, 224–314.

QUISQUATER, J.-J. 1998. Personal communication.

RIVEST, R. 1992. The MD5 message-digest algorithm. MIT Laboratory for Computer Science, Cambridge, MA.

RIVEST, R., SHAMIR, A., AND ADELMAN, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb.), 120–126.

SCHNEIER, B. 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd ed. John Wiley and Sons, Inc., New York, NY.

SHA AND NIST. 1995. NIST FIPS PUB 180-1, Secure Hash Standard. National Institute of Standards and Technology, Gaithersburg, MD.

SHAMIR, A. 1998. Personal communication.

Received: January 1999; revised: April 2000; accepted: April 2000