STREAMTO: STREAMING CONTENT USING A TAMPER-RESISTANT TOKEN

Jieyin Cheng[1], Cheun Ngen Chong[1], Jeroen M. Doumen[1], Sandro Etalle[1], Pieter H. Hartel[1] and Stefan Nikolaus[2]
[1]*University of Twente, P.O.Box 2100, 7500 AE Enschede, The Netherlands, {chengj,chong,doumen,pieter}@cs.utwente.nl;*

[2]*WIBU-SYSTEMS AG, Rueppurrer Strasse 52-54, 76137 Karlsruhe Germany, Stefan.nikolaus@wibu.com*

Abstract: StreamTo uses a tamper resistant hardware token to generate the key stream needed to decrypt encrypted streaming music. The combination of a hardware token and streaming media effectively brings tried and tested Pay-TV technology to the Internet. We present two prototype implementations with a performance assessment, showing that the system is both effective and efficient.

Key words: streaming; content protection; tamper-resistant hardware.

1. INTRODUCTION

To enforce usage rights and to prevent copyright violations, digital content needs to be protected. As shown in Fig. 1, content protection has three objectives (Judge and Ammar, 2003): (1) *protected distribution*, which protects content when it is accessed online by a content renderer, e.g. a streaming mechanism; (2) *protected storage*, which protects content while being stored locally, e.g. safe disc; and (3) *protected output*, which protects content after it is being rendered by a content renderer at the content output (say a sound card), e.g. Microsoft Secure Audio Path (SAP).

Content protection is difficult on a personal computer (PC) because most of the PC components (i.e., content renderer and content output) are open (i.e. programmable) and thus not trustworthy. When protected content is being used locally on a PC, an attacker might be able to retrieve the actual content by circumventing the protection mechanism (Greene, 2001).

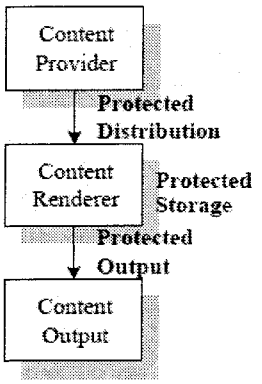

Figure 1. Three phases of content protection.

However, if content is stored on a server while being used via a streaming mechanism (SM), the security of the content can be guaranteed to a certain extent because the *entire* content is not sent to the user's PC directly but only piecemeal as a stream of packets (Holankar and Stamp, 2004). This stream of packets is interpreted and rendered at the user's PC as they arrive. Therefore, SM helps to achieve *protected distribution* of content, provided that the stream cannot be captured easily.

Compared to a PC, a consumer electronic (CE) device is relatively more trustworthy because its components can be manufactured compliant and non-programmable (Eskicioglu and Delp, 2001). Therefore, it is more difficult to circumvent the protection mechanisms applied to CE devices. A common example of such a content protection mechanism is the Conditional Access System (CAS) (Kravitz and Goldschlag, 1999). A Pay-TV system (Jain et al., 2002) applies CAS to control users' access to broadcast TV. Similar to SM, CAS is able to achieve *protected distribution* of the content.

In this paper, we propose StreamTo, which combines aspects of CAS and SM to design a content protection approach, supported by a tamper-resistant hardware token, e.g. a USB dongle. A tamper-resistant hardware token can also provide *protected storage* for the content.

In addition, StreamTo has the following benefits:

- It allows using content without an active Internet connection or when the user does not have sufficient bandwidth.
- It allows flexible sharing of content between users. The provider can control access to different parts of the content by different users. This is useful for business-to-business (B2B) and business-to-consumer (B2C), for instance, when paying users can enjoy full content access at near CD quality, while non-paying users can only listen to clips.

StreamTo is able to solve some of the security threats faced by CAS and SM. This will be discussed later in section 4. Like most streaming mechanisms, StreamTo is not easily scalable. Scalability could be achieved by using Broadcast Encryption techniques as pioneered by Fiat and Naor

(1994). However, this is beyond the scope of the present paper. Here, we show that StreamTo is applicable, practical and secure (within limits).

The remainder of the paper: Section 2 briefly explains CAS and SM, which inspired StreamTo. Section 3 describes StreamTo in detail. Section 4 implements a prototype on a CM-Stick and an iButton. Section 5 assesses the performance of the prototype. The last section concludes and presents future work.

2. CAS AND SM

A Conditional Access System (CAS) is a smart-card-based technology (Guillou, 1984), which is used in Pay-TV systems. The smart-card stores subscription information and a secret key. A set-top-box (STB) is required to interface with the smart-card and the television (TV).

The provider encrypts a TV program using a content key (which is the same for all users) and broadcasts the encrypted TV program, as shown in Fig. 2.

The key management system (KMS), which is responsible for billing, subscriber and key management, transmits the universal content key to the authorized subscribers. A content key is encrypted with the unique secret key stored on a smart-card (Macq and Quisquater, 1995). The smart-card decrypts and stores the content key received from the provider (via the STB). The STB decrypts the encrypted TV program with the content key, and displays the program on the TV.

The provider updates the content key used to encrypt the TV program/channel on a frequent basis (normally each 5 to 20 seconds). Once the key is updated, the KMS must retransmit the updated content key to the subscribers within seconds.

In a streaming mechanism (SM), as shown in Fig. 3, the provider encrypts the content with different content keys for different users. The content is encrypted and transmitted to the user.

A user has a renderer, which is a software application that establishes a secure channel with the provider. The content key is transmitted to the renderer when a secure streaming session is established. The renderer then decrypts the content packet by packet with the content key and renders it, as it is received, leaving behind no residual copy of the content at the renderer (assuming that the renderer is not hacked).

The characteristics of the content key of CAS, SM and StreamTo differ as shown in Table 1. We list the two most important characteristics of a content key: (1) uniqueness (whether the key is unique for different content and users), and (2) update (whether the key is updated on a regular basis).

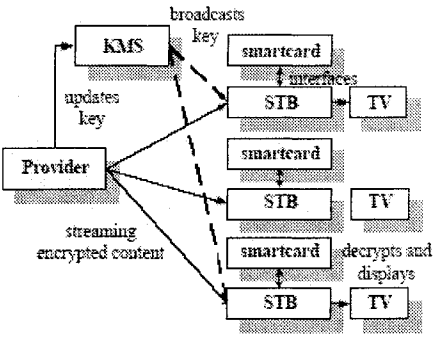

Figure 2. An abstract view of a conditional access system (CAS).

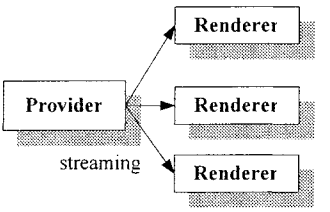

Figure 3. An abstract view of a streaming mechanism.

Table 1. Comparison of CAS, SM and StreamTo with respect to the characteristics of the content key.

	Uniqueness	Update
CAS	A content key is shared among all authorized users.	The content key is updated frequently.
SM	A unique content key is assigned to a user.	The content key is not updated in a streaming session.
StreamTo	A unique content key is assigned to a user.	The content key is updated frequently.

3. STREAMTO

In this section, we discuss StreamTo as outlined in Fig. 4.

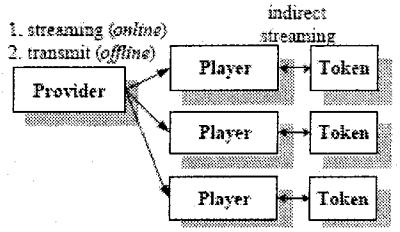

Figure 4. An abstract view of StreamTo.

We use a *token*, which has a cryptographic co-processor and tamper-resistant storage. The token is dispatched physically by the *provider* to a user in the same way as a Pay-TV smart-card. The provider also serves encrypted content. A user has a customized *player* (a software application) that interfaces with the token, and which can play encrypted content. The player depends on the token for providing the key stream necessary to decrypt the content stream.

StreamTo can handle two methods of rendering the content, as shown in Fig. 4: *online* and *offline*. For online rendering, the provider streams the content to the player; whereas for offline rendering, the provider transmits the entire encrypted content to the player. For both access methods, the

player plays the content *piecemeal*, waiting for each subsequent block of the key stream from the token. We call this *indirect streaming*.

StreamTo has the characteristics of both the CAS and SM:

- The provider generates a unique content key for a user (SM provider).
- The player decrypts and plays the content piecemeal (SM renderer).
- The token stores a unique secret key (CAS smart-card).
- The content key is updated frequently (CAS provider).
- The token transmits the updated key for decryption to the player (CAS KMS and smart-card).

To explain the StreamTo protocols in more detail, we use the notation listed in Table 2.

Table 2. The notation of the StreamTo protocols.

Notation	Meaning
SecK	A secret key shared between the token and the provider.
K_i	The content key for the i^{th} content frame.
S_i	The key stream for the i^{th} content frame.
P_i	The i^{th} frame of content (plaintext).
C_i	The corresponding i^{th} frame of encrypted content (ciphertext).

3.1 Keys

We use three different key types: a secret key, a content key and a key stream.

- A secret key (*SecK*) is a secret shared between the provider and the token. We assume that an attacker cannot read, modify or access this key stored on the token; it never leaves the token, and is preloaded on the token in a secure environment of the provider.
- A content key (K_i) is used for generating the key stream. The first content key K_0 is generated randomly by the provider and sent encrypted (with the secret key) to the user along with the encrypted content.
- A key stream (S_i) is used to en/decrypt the content. The key stream is derived from the content key and the content.

The size of the content key is short (e.g. 128 bits) so that a provider can send it to a player efficiently. The size of the key stream is equal to the size of the content so that stealing the key stream is inconvenient.

As a refinement, the provider could partition the content, using a different K_0 for each partition. This would allow for example free use of trailers but paid for use of the remaining content. In this paper, for simplicity, we only use one content key to explain StreamTo in the subsequent sections.

3.2 Encryption Process

Streaming content, e.g. an MPEG audio/video has a special structure: it is composed of multiple frames, each of which has a descriptive header. This header contains the particular information for the corresponding frame, e.g. bit-rate, sample-rate, etc. StreamTo exploits this special feature of streaming content as follows:

$$S_i = \textbf{generate}(K_i, SecK) \tag{1}$$

$$K_{i+1} = \textbf{transform}(K_i) \tag{2}$$

$$Ci = Pi \oplus Si \tag{3}$$

The encryption process, as shown in Fig. 5 is performed by the provider. The provider generates a first content key K_0 randomly. The generate function (Eq. 1) takes the content key K_i and the secret key to produce a block of key stream for the current frame (P_i). The encrypted frame (C_i) is then XORed with the block of key stream, as shown in Eq. 3. Finally, the

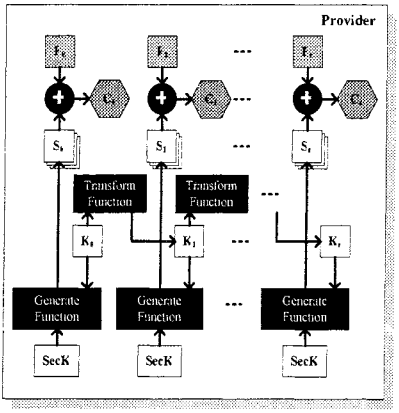

Figure5. Encryption of streaming content, frame by frame, at the provider with a key stream that is generated from an initial content key.

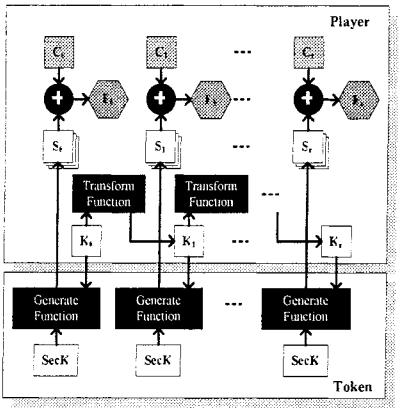

Figure 6. Decryption of encrypted streaming content, frame by frame, with the regenerated key stream using the regenerated content key.

next content key is calculated by the transform function. If the output of the generate function is shorter than the frame size, it is repeated to form the required length.

The encrypted frames ($C_0,\ldots,C_n$) are written to a new content file, preceded by a header. The header contains the first content key (K_0) (encrypted with the secret key *SecK* of the token), padding, and information about the **generate** and **transform** functions.

3.3 Decryption Process

The player receives an encrypted content file from the provider. When the player renders the encrypted content, the decryption process is executed as shown in Fig. 6.

The player interprets the header information of the encrypted content to retrieve the encrypted first content key K_0 and other information. The player then asks for a valid token. Authentication can be achieved between the player and the token with standard methods (Kelsey and Schneier, 1999); this falls outside the scope of this paper. The player then feeds the token with the encrypted first content key (K_0), so that the token can decrypt it.

The token uses the generate function (Eq. 1) to re-generate the key stream, and sends it to the player. The player retrieves a frame C_i from the encrypted content, and decrypts it (by XOR-ing) with key stream S_i generated by the token, as shown in Eq. (4).

$$P_i = C_i \oplus S_i \tag{4}$$

The player then updates the content key using the transform function (Eq. 2), sends it to the token to generate the next block of the key stream, and the next frame is decrypted with this key stream block. At the same time, the player plays the previously decrypted frame P_{i-1}. The decrypted frames will be overwritten by newly decrypted frames after they are played (again, assuming the player has not been hacked). Ideally, the token would perform the transform function itself, but our hardware (the CM-stick) is not capable of doing this.

4. PROTOTYPES

In this section, we discuss the implementation of our prototype. We use streaming audio (MP3) in our prototype because it is less demanding on resources than video. If StreamTo can be applied practically to protect

streaming audio, we can investigate if StreamTo can support other streaming content as well.

The architectural overview of our prototype is given in Fig. 7. The Provider and the Player are the two applications we have created by using Windows Media Format SDK, iB-IDE, CM-Stick SDK (WIBU, 2003), and JavaZoom JLayer SDK.

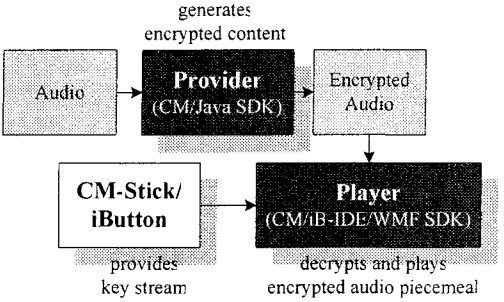

Figure 7. Architectural Overview of the prototype.

The Provider executes the encryption process discussed in section 3.2. It takes as input an MP3 audio file and produces an encrypted audio file as output. The Player performs the decryption process discussed in section 3.3. It asks the token (i.e., CM-Stick or iButton) continuously for blocks of key stream to decrypt the audio.

The hardware token we use in our prototype are a CodeMeter Stick (CM-Stick) and an iButton (as shown in Table 3).

Table 3. Comparison of the iButton and the CM-Stick.

	CM-Stick	iButton
Manufacturer	WiBu-Systems AG, Germany	Dallas Semiconductor,America
Processor Speed	24 MHz	10–20 MHz (Kingpin, 2002)
Non-volatile memory	128 kBytes	134 kBytes
Cryptographic algorithms	AES, Triple-DES (for communication), ECC, SHA-256	DES, Triple-DES, RSA and SHA-1
Interface	USB connection	Serial/Parallel and USB connection

We use the standard Counter-mode (CTR-mode) symmetric encryption (Lipmaa and Rogaway, 2000) to implement StreamTo by virtue of the simplicity, efficiency and proven security of CTR-mode encryption.

The content key (K_n) is the counter of CTR-mode encryption, which is initialized to a random n-bit string. The implementation of the **transform** function of the content key (Eq. 2) is simple:

$$K_{i+1} = K_i + 1$$

The generation of the key stream (S_n) (Eq. 1) is the encryption of the counter in CTR-mode encryption. We use AES encryption, which is the only

symmetric encryption supported by the CM-Stick; and DES encryption on the iButton as the **generate** function (Eq. 1).

In our prototypes, each time a new frame is decrypted a click is audible. This allows us to point out during demonstrations when decryption happens.

5. PERFORMANCE ASSESSMENT

To justify the practicality of StreamTo, we assess the performance of our prototype. Our prototype is built on a platform with an Intel Pentium 4, 1.4 GHz, 512 MBytes RAM, 20 GBytes hard disk space, running Windows XP. We use a 1-minute 192 kbps MP3 audio as the sample for our performance assessment. The sample has 2300 frames, each of which contains 623 bytes.

In our prototype, we use a CM-Stick, which is attached with a USB interface; and an iButton, with two different interfaces to the platform, namely a serial port connection (with the adapter DS9097U) and USB connection (with the USB iButton holder DS9490B).

5.1 Content Key Size

The key stream is generated on the tokens by using the firmware symmetric encryption algorithm. Therefore, to determine if the content key size influences the performance of our prototype, we assess the performance of symmetric encryption on the iButton and the CM-Stick.

From our previous experience, we know that the cryptographic operations on the iButton are relatively slow (Chong et al., 2003). DES encryption of 128 bytes on the iButton takes roughly 200 ms (Chong et al., 2004).

We also need to measure the time required by the CM-Stick to perform AES encryption, which we use to generate the key stream. We use an LSQ-fit equation to summarize the result of 10 measurements as follows:

$$t = (0.5 \pm 0.002) \times d + (36 \pm 26) \text{ ms}$$

Here, t is the time required in milliseconds and d is the data size in bytes. Thus, it takes approximately 100 ± 25 ms to encrypt 128-byte of data on the CM-Stick, making the CM-Stick about twice as fast as the iButton. This is consistent with the cryptographic co-processor speed (Table 3).

If we use 128 bytes of content key, i.e., 1024 bits, the iButton requires approximately $2300 \pm 0.2 = 460$ seconds to generate the key stream, whereas the CM-Stick needs roughly 230 seconds. For a 1-minute MP3 this is too long, hence, we must sacrifice security for performance by (1) using a

smaller content key size; and (2) en/decrypting every n-th frame of the audio sample only.

In our prototypes, we choose a content key of size 8 bytes (64 bits) for the iButton and 32 bytes (256 bits) for the CM-Stick. On the CM-Stick, it takes approximately 40 ± 26 ms to generate a block of key stream. However, for the iButton, it takes approximately 70 ms due to the slower co-processor. Therefore, we also use the second tradeoff on the iButton prototype, as will be discussed in next section.

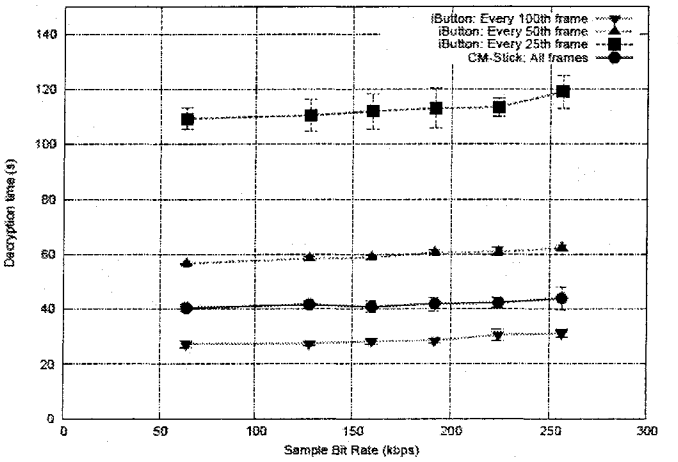

Figure 8. The time required to decrypt the encrypted audio sample frame by frame with the iButton and the CM-Stick.

5.2 Sample Bit Rate

The MP3 sample bit rate refers to the transfer bit rate for which an audio file is encoded. The sampling frequency refers to the number of samples of an audio taken per unit time, i.e., the rate at which audio signals are sampled into digital form.

The frame size depends on the sample bit rate and sampling frequency according to the MPEG-3 standard. We use 6 different sample bit rates (with the same sampling frequency of 44.1 KHz) of our experiment, which include 64 kbps, 128 kbps, 160 kbps, 192 kbps, 224 kbps and 256 kbps. Each frame has standard constant time length of 26 ms. A 1-minute MP3 has approximately 2300 frames.

It takes roughly 80 ms to generate a block of key stream. Therefore, for decrypting the audio sample 192 kbps of 2300 frames (1 minute of play time)

by using a content key of 64 bits, theoretically the iButton needs approximately $2300 \times (0.08 + 0.2 \times 2) = 1104$ seconds in total to generate and transmit the key stream to the player. We have rerun the test using the USB iButton holder. However, there is no obvious improvement of the speed due to the slow cryptographic operations on the iButton.

To overcome this problem, we choose appropriate values of n, en/decrypting every n-th frame of the audio sample. Fig. 8 shows the measurement for n = 25, 50 and 100. We report the average of 10 measurements. The decryption time measured includes the time required to upload the updated content key; to generate and transmit a block of the key stream; and XOR-ing of the encrypted frame. The actual play time of the audio sample is 60 seconds (i.e., $y = 60$).

As can be seen in Fig. 8, the graphs of the iButton are slightly slant, indicating that the time required to decrypt the audio frames increases with faster sample bit rate. This is caused by the preprocessing of the encrypted audio file, i.e. reading the audio frames from an encrypted audio file.

When n = 100, the iButton is able to handle the key stream generation and audio frames decryption comfortably in real time. When n = 50, the decryption time is marginally parallel with the play time of the audio sample. This means that real time playback is possible but only when every n-th frame is encrypted and $n \geq 50$.

On the other hand, the CM-Stick, due to its faster cryptographic co-processor, has better performance than the iButton, as shown in Fig. 8. In conclusions, the CM-Stick is able to provide real time playback at $n = 1$.

6. CONCLUSIONS AND FUTURE WORK

We propose a streaming content protection approach, namely StreamTo, which combines the technology of the Internet streaming mechanism (SM), Pay-TV Conditional Access System (CAS) and a tamper-resistant hardware token.

We implement StreamTo on two commercial tokens, namely the iButton and the CM-Stick, by using the CTR-mode of symmetric encryption. Thus, we show the applicability of StreamTo. We also evaluate the performance of the implementation to justify the practicality of StreamTo. The CM-Stick has a better performance than the iButton due to its faster cryptographic coprocessor.

REFERENCE

Buchheit, M. and Kgler, R. (2004). Secure music content standard – content protection with codemeter. In *4th Open Workshop of Interactive Music Network Multimedia MUSICNETWORK*, page Paper 10.

Chong, C. N., Peng, Z., and Hartel, P. H. (2003). Secure audit logging with tamper-resistant hardware. In Gritzalis, D., di Vimercati, S. D. C., Samarati, P., and Katsikas, S. K., editors, *18th IFIP International Information Security Conference (IFIPSEC)*, volume 250 of *IFIP Conference Proceedings*, pages 73–84. Kluwer Academic Publishers.

Chong, C. N., Ren, B., Doumen, J., Etalle, S., Hartel, P. H., and Corin, R. (2004). License protection with a tamper-resistant token. In Lim, C. H. and Yung, M., editors, *5th Workshop on Information Security Applications (WISA 2004)*, volume 3325 of *LNCS*, pages 224–238. Springer-Verlag.

Eskicioglu, A. M. and Delp, E. J. (2001). An overview of multimedia content protection in consumer electronics devices. *Signal Processing: Image Communication*, 16:681–699.

Fiat, A. and Naor, M. (1994). Broadcast encryption. In *Advances in Cryptology (CRYPTO'03 Proceedings*, volume 773 of *LNCS*, pages 480–491. Springer-Verlag.

Greene, T. C. (2001). MS digital rights management scheme cracked. *TheRegister.co.uk.*

Guillou, L. C. (1984). Smart cards and conditional access. In *Advances in Cryptology EUROCRYPT 84)*, volume 209 of *LNCS*, pages 480–485. Springer-Verlag.

Holankar, D. and Stamp, M. (2004). Secure streaming media and digital rights management. In *Proceedings of the 2004 Hawaii International Conference on Computer Science*, pages 85–96. ACM Press.

Jain, P. C., Joshi, S., and Mitra, V. (2002). Conditional access in digital television. In *The 8th National Conference Communications (NCC) 2002*, Technical Session paper 30.

Judge, P. and Ammar, M. (2003). The benefits and challenges of providing content protection in peer-to-peer systems. In *Int. Workshop for Technology, Economy, Social and Legal spects of Virtual Goods*, paper 12, Ilmenau. Germany.

Kelsey, J. and Schneier, B. (1999). Authenticating secure tokens using slow memory access (extended abstract). In *USENIXWorkshop on Smart Card Technology*, pages 101–106. USENIX Press.

Kingpin (2002). A practical introduction to the dallas semiconductor ibutton. Technical report,@Stake, Inc.

Kravitz, D. W. and Goldschlag, D. M. (1999). Conditional access concepts and principles. In *Proceedings of the 3rd International Conference on Financial Cryptography*, volume 1648 of *LNCS*, pages 158–172. Springer-Verlag.

Lipmaa, H. and Rogaway, P. (2000). Comments to NIST concerning AES-modes of operations: CTR-mode encryption. In *Symmetric Key Block Cipher Modes of Operation Workshop*, Electronic Proceedings.

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

WIBU (2003). *CodeMeter Developer's Guide.* WIBU-SYSTMES AG, Rueppurrer Str.53-54 76137 Karlsruhe, Germany, 1.0 edition.