STREAMTO: STREAMING CONTENT USING A TAMPER-RESISTANT TOKEN

Jieyin Cheng[1], Cheun Ngen Chong[1], Jeroen Doumen[1], Sandro Etalle[1], Pieter H Hartel[1] and Stefan Nikolaus[2]

[1] *University of Twente*
P.O.Box 2100, 7500 AE Enschede 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 tamper resistant hardware tokens to generate the key stream needed to decrypt encrypted streaming music. The combination of a hardware token and steaming media effectively brings tried and tested PayTV technology to the Internet. We provide a security analysis and present two prototype implementations with a performance assessment, showing that the system is both effective and efficient.

Keywords: 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 Figure 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. 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 a content output (e.g. 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].

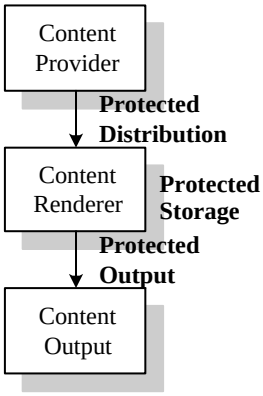

Figure 1. Three phases of content protection.

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 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. We believe that in the foreseeable future, the hardware token will become common due to the convenience it provides to users and its reasonable price. 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 i.e., *offline*, or when the user does not have sufficient bandwidth. Similarly, the content provider does not have to worry about the overloaded servers when there is a large number of users demanding online access.
- 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 listen to clips only.

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 (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). To conclude, our contributions in this paper are:

- We propose StreamTo, which is able to provide a measure of protection for streaming content. The approach is inspired by concepts of CAS and SM. We analyze the security of the approach corresponding to common threats.
- We implement StreamTo on two commercial tokens, namely the CodeMeter Stick (CM-Stick) Buchheit and Kgler [2004] (`http://www.wibu.com`) and the Java iButton (`http://www.ibutton.com`). This shows the applicability of StreamTo.
- We assess the performance of the prototype to justify the practicality of StreamTo.

The remainder of the paper: Section 2 briefly explains CAS and SM, which inspired StreamTo. Section 3 describes StreamTo in detail. Section 4 analyzes the security of StreamTo, referring to some known security threats. Section 5 implements a prototype on a CM-Stick and an iButton. Section 6 assesses the performance of the prototype. Section 7 discusses related work. 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).

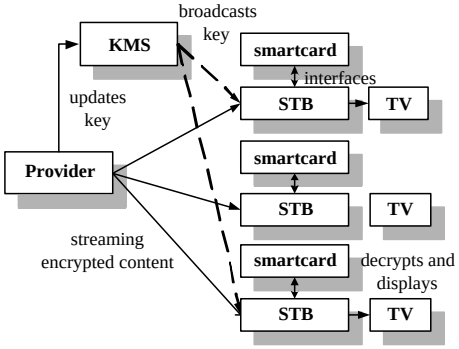

Figure 2. An abstract view of a conditional access system (CAS).

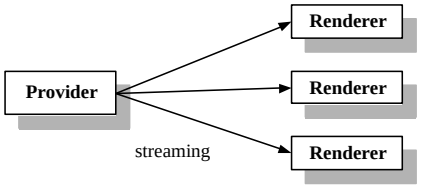

Figure 3. An abstract view of a streaming mechanism.

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 Figure 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 STB). The STB decrypts the encrypted TV program with the content key, and displays the program on a 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 Figure 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 (packets) at the PC (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 user), and (2) update (whether the key is updated on a regular basis).

	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.

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

3. StreamTo

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

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.

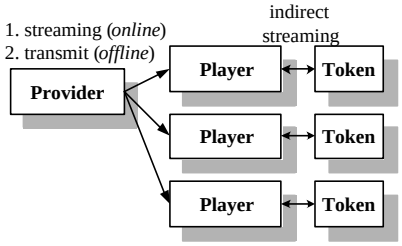

Figure 4. An abstract view of StreamTo.

StreamTo can handle two methods of rendering the content, as shown in Figure 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 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.

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

Table 2. The notation of the StreamTo protocols.

We explain the types of key that we use in StreamTo in section 3.1. We describe the encryption and decryption process in section 3.2 and section 3.3, respectively.

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 to 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. We can also enforce different usage rights on different parts of content by using our LicenseScript license protection scheme [Chong et al., 2004].

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:

$$S_i = \mathbf{generate}(K_i, SecK) \quad (1)$$

$$K_{i+1} = \mathbf{transform}(K_i) \quad (2)$$

$$C_i = P_i \oplus S_i \quad (3)$$

Here, i is an integer, $0 \leq i < \mathrm{last\_frame\_number}$. The provider uses two functions: (1) a *generate function* (Equation 1), e.g. a pseudo-random number generator, which generates a key stream from a content key; and (2) a *transform function* (Equation 2), e.g. a keyed hash function, which updates a content

key. These functions must be supported by the token to perform the decryption process, as will be discussed in section 3.3.

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

We use the secret key $SecK$ in the generate function to ensure that the encrypted content is bound to the token. In addition, we want to assure that an attacker cannot derive a valid content key or key stream without the correct secret key.

The encrypted frames ($C_0, \cdots, C_n$) 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, information about the generate and transform functions.

3.3 Decryption Process

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

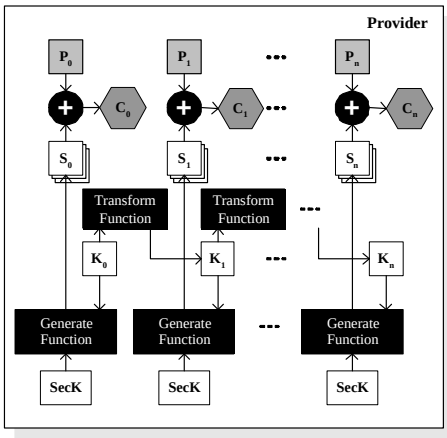

Figure 5. Encryption of streaming content, frame by frame, at the provider with a key stream that is generated from an initial content key K_0.

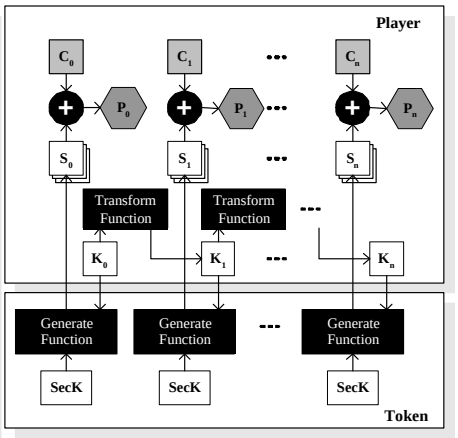

Figure 6. Decryption of encrypted streaming content, frame by frame, with the regenerated key stream using the regenerated content key.

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 (Equation 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 Equation 4.

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

The player then updates the content key using the transform function (Equation 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).

4. Security Analysis

In this section, we analyze the security of StreamTo. We consider several known security threats: the analog hole, digital hole, streaming theft, jugular attack, and cloning.

4.1 Analog and Digital Hole

The main security threat to Pay-TV and streaming is the widespread proliferation of content-capturing tools [Wald, 2002]. This threat is known as the "analog hole" [Holankar and Stamp, 2004], and is considered beyond the boundary of normal content protection approach. Therefore, StreamTo is not concerned with this security threat either.

The digital hole is another severe threat to common content protection approach, especially for the PC. The digital hole opens between the moment of decrypting and the moment of rendering the content. An attacker might be able to retrieve the plaintext content from the memory (especially on the PC) because the memory is always susceptible to manipulation and observation.

On a PC not much can be done to close the digital and analog holes; on CE devices, the digital hole is more or less plugged, but the analog hole is still present. Watermarking is then the only viable technique, which is able to provide some measure of protection [Haitsma and Kalker, 2002].

4.2 Streaming Theft

An attacker can steal streaming content simply by copying the source target, e.g. URL of the streaming content, and publish it on, say, another Web page on another server. Therefore, users can access the streaming content stored on the provider's server through the attacker's Web page.

In essence, the attacker is not only stealing the streaming content but is also stealing the original provider server's bandwidth. This can induce costs to the provider and can be difficult to track down.

In StreamTo, only authorized users, who possess tokens dispatched by the original provider, can access the streaming content. Therefore, it is not possible for other unauthorized users to decrypt the streaming content through another server.

4.3 Jugular Attack

In a Pay-TV system, an attacker is able to obtain descrambling keys by watching the output of the smart-card. The attacker can then redistribute these keys in near real-time to other users, allowing others to view the encrypted broadcast. The attacker can charge these users a cheaper fee. This attack exploits the vulnerability of using a universal content key for a TV program/channel for all users, and is known as the jugular attack [McCormac, 1996].

Online StreamTo is safe from this attack because we use different content keys for different users. Therefore, the attacker's keys are useless to other users, even if she is able to obtain the keys and redistribute them in near real-time.

However, for offline usage, a jugular attack poses a severe threat because the encrypted content file is stored on a user's PC. Assuming a hacked renderer, an attacker can obtain the keys by playing the encrypted content just once: XOR the ciphertext and the plaintext provides the key stream; or XOR the ciphertext and the key stream stolen provides the plaintext.

In any case, StreamTo is reasonably secure for short-lived content, i.e., if the value of he content is reduced after a short period of time. For instance, a breaking news video broadcast, a series of stock quote prices, etc. Therefore, once the content has been decrypted and presumably saved in the clear, we do not insist on communication with the token anymore, i.e., the offline jugular attack and plaintext key stream do not pose a severe threat to StreamTo anymore.

4.4 Cloning

Cloning [Rao et al., 2002] is a physical security threat, from which Pay-TV systems suffer. A valid smart-card of a set-top-box can be cloned and

distributed illegally. This is achieved for example by using side-channel attacks [Anderson and Kuhn, 1997] to steal and copy the secret key of a smart-card. Thereby, home users can purchase these smart-card clones for a lower price from an illegal distributor to access Pay-TV programs.

A hardware token provides a higher physical security than a smart-card because the chip is protected by a more physical means, e.g. stainless steel casing. Additionally, it is more expensive to clone a hardware token than a smart-card.

In conclusion, we have argued that StreamTo is more secure than a typical smart-card-based PayTV system and a streaming system. Other content protection approaches have been proposed attempt to resolve some of the aforementioned security threats, which will be discussed briefly in section 7.

5. Prototype

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.

Section 5.1 presents an architectural overview of our prototype using the iButton and the CM-Stick, and introduces the software tools that we have used. Section 5.2 elaborates the implementation of StreamTo.

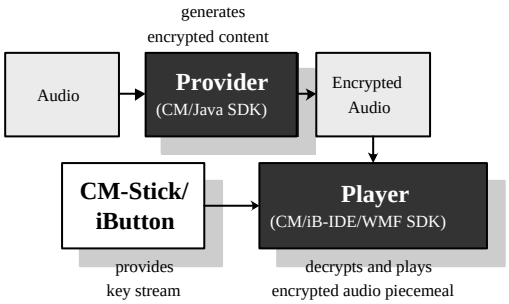

Figure 7. Architectural Overview of the prototype.

5.1 Architecture

The architectural overview of our prototype is given in Figure 7. The Provider and the Player are the two applications we have created. 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.

We use five software development kits (SDK) for the Provider and the Player with the CM-Stick and the iButton: (1) Windows Media Format (WMF) SDK (`http://msdn.microsoft.com/av/`), which supports different types of streaming audio standards, such as WMA and MP3; (2) CodeMeter (CM) SDK [WIBU,

2003], which supports interfacing with a CM-Stick; (3) Java Cryptography Extension (JCE) (`http://java.sun.com/products/jce/`), which provides a standard cryptography library; (4) Javazoom JLayer (`http://www.javazoom.net/javalayer/javalayer.html`), which supports MP3 audio; and (5) iB-IDE SDK (`http://www.ibutton.com/iB-IDE/`), which supports interfacing with an iButton.

The hardware token we use in our prototype are a CodeMeter Stick (CM-Stick) and an iButton. Their features are illustrated in Table 3. Note that the speed of the iButton is not constant and cannot be determined by external means [Kingpin, 2002].

	CM-Stick	iButton
Manufacturer	WiBu-Systems AG, Germany	Dallas Semiconductor, America
Processor Speed	24 MHz	10–20 MHz
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

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

5.2 Implementation

In this section, we give the implementation details of our prototype for StreamTo. 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 update function of the content key (Equation 2) is simple:

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

The generation of the key stream (S_n) (Equation 1) is the encryption of the counter in CTR-mode encryption:

$$\begin{aligned} S_i &= \mathbf{AES\_CBC}(K_i, SecK) \quad (\mathrm{CM-Stick}) \\ S_i &= \mathbf{DES\_ECB}(K_i, SecK) \quad (\mathrm{iButton}) \end{aligned}$$

Here, $\mathbf{AES\_CBC}$ is AES encryption in CBC mode, which is the only symmetric encryption supported by the CM-Stick; whereas $\mathbf{DES\_ECB}$ is DES encryption in ECB mode, which is supported by the iButton.

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.

In next section, we assess the performance of our prototype to justify the practicality of StreamTo.

6. 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).

6.1 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 (ECB mode) 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 (CBC mode) encryption, which we use to generate the key stream. We use an LSQ-fit equation to summarize the result of 10 measurements as follows:

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

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 $2300 \times 0.2 = 460$ 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.

6.2 Sample Bit Rate

The MP3 sample bit rate refers to the transfer bit rate for which an audio file is encoded, e.g. an MP3 file encoded at "at a bit rate of 128 kbps" is compressed such that it can be streamed continuously through a transfer link providing a transfer rate of 128 thousand bits per second. The lower the bit rate, the more the audio file is compressed, and the worse the playback sound quality becomes.

Sample Bit Rate (kbps)	File Size (MBytes)	Frame Size (bytes)
64	0.46	205
128	0.92	414
160	1.14	518
192	1.37	623
224	1.60	727
256	1.83	832

Table 4. Different sample bit rate, with different audio file size and average frame size.

On the other hand, 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. A higher sampling frequency implies a higher-quality of the digital audio.

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, as shown in Table 4. Each frame has standard constant time length of 26 ms. Therefore, the number of frames is determined by the duration of the entire MP3 audio. Therefore, for 1-minute MP3, it has approximately 2300 frames.

In addition, 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 re-run 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. Figure 8 shows the measurement for $n = 25,\ 50, \text{and } 100$. 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., $y = 60$).

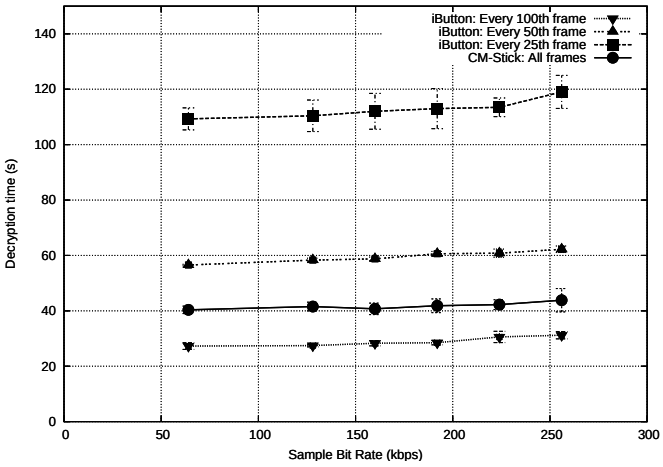

Figure 8. The time required to decrypt the encrypted audio sample frame by frame with the iButton and the CM-Stick.

As can be seen in Figure 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: reading the audio frames from an encrypted audio file.

When $n = 100$, the iButton StreamTo 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 just when $n \geq 50$.

On the other hand, CM-Stick, due to its faster cryptographic co-processor, has far better performance than the iButton, as shown in Figure 8. The time required to decrypt all the audio frames with the CM-Stick is shorter than the actual play time of the audio sample. In conclusions, the CM-Stick is able to provide real time playback at $n = 1$.

7. Related Work

In this section, we discuss related work.

7.1 Content Protection Approach

Several content protection approaches have been proposed. As some of them are proprietary, we are only able to report on publicly available material.

Microsoft Secure Audio Path (SAP) [Microsoft, 2001] is a content protection approach specifically for audio. SAP tries to plug the “digital hole”. When

a user attempts to play an audio file, which is encrypted by an SAP-based method, the audio file is sent to a player where it is decrypted. Some audio noise is added to the decrypted audio signal and is removed only when the content output device (e.g. sound card) has been authenticated to the operating system. Therefore, even if an attacker is able to intercept the decrypted audio signal, she will hear noise.

The Digital Transmission Content Protection (DTCP) [DTCP, 1998] is proposed by Hitachi, Intel, Matsuhita, Sony and Toshiba. DTCP proposes a full-fledged content protection architecture for CE devices, which consists of compliant components. Several communication protocols for these components have been proposed to achieve key management, authentication, encrypted content transmission, content en/decryption etc., so that the content is protected throughout its cycle of usage.

DTCP is a hardware-based approach. Similar to SAP, DTCP attempts to patch the "digital hole" by protecting the transmission of the actual content between devices. For using the DTCP, a certain amount of royalty fees is required.

Both the SAP and DTCP rely on the authentication and trustworthiness of the device components. Therefore, a trusted computing platform (TCP) [Pearson et al., 2003] is a critical foundation to their approach. At this moment, to plug the "digital hole", TCP is deemed to be one of the potential solutions, which is discussed in the subsequent section.

7.2 Trusted Platform

A group of organizations, including Intel, IBM, and HP have formed the Trusted Computing Group (TCG). TCP tries to build a trusted and tamper-resistant system platform, e.g. a PC, a personal device assistant (PDA) etc., by adding tamper-resistant chips on the motherboard. These chips, the so-called Trusted Platform Module (TPM) and Core Root of Trust Module (CRTM) are used to store sensitive information of users, e.g. private keys, and the hash value of the current status of the platform, including installed software.

Before any further processing on the platform is done, the current status of the platform is validated with the hash value stored on the chip. If a difference is detected, it might imply potentially unauthorized manipulation of the platform.

However, TCP technologies may cause severe inconvenience to users, e.g. the protected audio is only playable on a TCP-enabled PC but not other portable devices. This implies that the protected content is tightly coupled to the TCP platform because it is difficult to move the secret key securely (once the key leaves the hardware, the physical security of the key cannot be guaranteed).

Our method of using a hardware token provides a more user-friendly content usage: users can move and use their content “anyway and anywhere”.

8. 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.

StreamTo is able to bring tried and tested Pay-TV technology to the personal computer (PC). In addition, StreamTo is able to offer protection while accessing the streaming content *offline*, which benefits the user (she can access the content without Internet connection), as well as the provider (she does not need to worry about overloaded servers when there is a large number of users). We analyze the security of StreamTo with respect to common security threats. We conclude that StreamTo is more secure than either SM or CAS.

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 co-processor.

Our future work is: (1) To integrate StreamTo with a standard streaming algorithm, such as the Real Time Streaming Protocol (RTSP). Thereby, we can use StreamTo for more efficient Internet streaming; and (2) To evaluate the performance of StreamTo on the network, by taking into account some Internet parameters, such as the bandwidth latency and delay, etc.

Bibliography

R. Anderson and M. Kuhn. Low cost attacks on tamper-resistant devices. In B. Christianson, B. Crispo, T. Mark A. Lomas, and M. Roe, editors, *Security Protocols, 5th Internation Workshop Proceedings*, volume 1361 of *LNCS*, pages 125–136. Springer-Verlag, April 1997.

M. Buchheit and R. Kgler. Secure music content standard – content protection with codemeter. In *4th Open Workshop of Interactive Music Network Multimedia MUSICNETWORK*, page Paper 10, September 2004. `http://www.interactivemusicnetwork.org/events/Fourth_OpenWorkshop_2004/%musicnetwork-xxxc.pdf`.

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

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

DTCP. 5C digital transmission content protection – white paper. Technical report, Hitachi, Intel, Matsushita, Sony and Toshiba, July, 14 1998. `http://www.dtcp.com/`.

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

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

T. C. Greene. MS digital rights management scheme cracked. *TheRegister.co.uk*, October 2001. URL `http://www.theregister.co.uk/content/4/22354.html`.

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

J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *3rd International Conference on Music Information Retrieval (ISMIR)*, pages 107–115, 2002.

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

P. C. Jain, S. Joshi, and V. Mitra. Conditional access in digital television. In *The 8th National Conference Communications (NCC) 2002*, page Technical Session paper 30, January 2002. `http://www.ee.iitb.ac.in/uma/~ncc2002/proc/NCC-2002/`.

P. Judge and M. Ammar. The benefits and challenges of providing content protection in peer-to-peer systems. In *Int. Workshop for Technology, Economy, Social and Legal Aspects of Vir-*

tual Goods, page 12 pages, Ilmenau, Germany, May 2003. URL http://virtualgoods.tu-ilmenau.de/2003/.

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

Kingpin. A practical introduction to the dallas semiconductor ibutton. Technical report, @Stake, Inc., 2002. URL http://www.atstake.com/research/reports/acrobat/practical_introduction_%to_ibutton.pdf.

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

H. Lipmaa and P. Rogaway. Comments to NIST concerning AES-modes of operations: CTR-mode encryption. In *Symmetric Key Block Cipher Modes of Operation Workshop*, page Electronic Proceedings, October 2000. http://www.tcs.hut.fi/~helger/papers/lrw00/.

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

J. McCormac. *European Scrambling Systems*. Waterford University Press, 1996.

Microsoft. Understanding secure audio path. Technical report, Microsoft, 2001. http://www.microsoft.com/windows/windowsmedia/drm/whitepapers.aspx.

S. Pearson, B. Balacheff, L. Chen, D. Plaqui, and G. Proudler. *Trusted Computing Platforms – TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, New Jersey 07458 United States, 2003.

J. R. Rao, P. Rohatgi, H. Scherzer, and S. Tinguely. Partitioning attacks: Or how to rapidly clone some gsm cards. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 31–42. IEEE Computer Society, May 2002.

S. Wald. Secure media consumption in a Ubicomp World. In *Workshop on Security in Ubiquitous Computing (UBICOMP'02)*, page Paper 10, September 2002. http://www.teco.edu/~philip/ubicomp2002ws/.

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