Trusted Watermarks

Andreas Brett*, Nicolai Kuntze†, Andreas U. Schmidt‡

* **Department of Computer Science, Technische Universität Darmstadt**
Darmstadt, Germany
brett@rbg.tu-darmstadt.de

† **Fraunhofer-Institute for Secure Information Technology SIT**
Rheinstraße 75, 64295 Darmstadt, Germany
nicolai.kuntze@sit.fraunhofer.de

‡ **CREATE-NET Research Center**
Via alla Cascata 56/D, 38100 Trento, Italy
andreas.schmidt@create-net.org

***Abstract*—The Internet becomes more and more integrated into everyday life these days. Reasons may be increasing data transfer rates and decreasing connection fees. Increasing data transfer rates on the client-side cause a widespread usage of peer-to-peer techniques that balance network load between servers and clients. Especially when distributing content to a multitude of customers, usage of peer-to-peer mechanisms comes in handy. Another main impact of the Internet nowadays is an increasing appetite for on-demand content which leads to entertainment devices providing customers with on-demand media from the Internet in their living rooms.**

Furthermore uprising media-deployment results in an arising demand for theft-protection by the holders of copyrights. Nevertheless past showed that copyright protection methods that restrict the user in being able to playback acquired media without difficulty cause decreasing sales figures. Thus customer-friendly copyright-protection methods seem to bridge the gap between customer's and supplier's needs.

Moreover one major problem in the growing usage of software implemented algorithms in embedded systems, like on-demand entertainment-devices, is the ability of customers to modify their devices in order to reveal secret data or to bypass copyright-protection providing methods. Integrity-proving mechanisms that ensure devices are being unaltered are required accordingly.

This paper presents a concept for a so called Trusted Set-Top-Box (TSTB) that will make use of peer-to-peer file sharing mechanisms to provide the customer with real-time and on-demand video streaming. To protect copyrighted video material from theft in a customer-friendly manner, novel watermarking-technology will be used. This allows for unproblematic playback of acquired media on distinct playback devices. The TSTB is referenced to be trusted since TPM-Technology will serve as security anchor, providing methods for attesting the devices' integrity. Through this unauthorized modification of the device will be detected.

I. Introduction

Peer-to-peer (hereinafter referred to as P2P) turned out to be the most successful content distribution method for larger amounts of content in the past years since network load is naturally balanced to every participant. By using P2P there is no more need for expensive server infrastructures that provide methods and techniques for network load balancing. This uprising success of P2P however leads to difficulties concerning security and customer experiences.

Concerning customer experiences there is no quality of service that can be guaranteed as P2P performance is technically unforeseeable so no SLAs can be given. Furthermore speaking of distributed content, security (particularly integrity of content) is one major issue as content is not only spread by content providers but the customers themselves. Content providers are no longer able to ensure what content appears at the customer's side. Using P2P opens the door for customers taking over parts of the P2P network in order to deploy their own content instead of the supplier's one.

As the former problem is to some degree negligible concerning inflating transfer rates and is already addressed by studies about Quality of Service (QoS) capable P2P strategies, we will show a way to solve the latter problem by using modern integrity proving methods. Using TPM-Devices inside content-distributing devices enables content providers to state devices to be unaltered and thus trusted. Only data distributed by trusted devices will be accepted, preventing malicious users from deploying their own content instead of the providers'.

Another security aspect to be well considered is privacy. Distributing copyrighted content in a customer-friendly manner leads to marking content with customer-specific data instead of wrapping it in proprietary formats that can only be played back on supported devices. Content hence is no longer bound to a given license that a device needs to verify in order to allow for decoding the copyrighted material into playable content. Instead every device is able to playback copyrighted material without the need of decoding it. Additionally devices may read and interpret the embedded customer-specific data for any purpose.

This kind of copyright-protection conflicts with the fundamentals of P2P, namely sharing equal content among many customers, as each customer obtains its personalized content. Therefore more sophisticated customer-friendly copyright-protection techniques need to be developed to combine the

advantages of both, copyright-protection and P2P.

We will show how to solve this issue and further shift load from the server to the client side by transferring unmarked material via P2P and thereupon embedding watermarks right on the client's device. This provides a) *customer-bound material* that is customer-friendly in respect to playability and privacy, b) *compatibility to P2P* and c) *reduced server load*. Since unmarked media is sent to the customer via P2P, this vulnerable transmission path has to be strongly secured in particular. We therefore propose to strongly encrypt unmarked content before distribution with one unique key for all P2P participants. Furthermore this key should be cryptographically bound to the customer's Set-Top-Box before handing it over. This is done by utilizing the bind feature provided by TPM-Devices thus unfolding the key only to the TPM-Device itself. Additionally watermarks to be embedded into shared media should be passed to the Set-Top-Boxes in the same manner.

By ensuring system integrity of each device before sharing data, *unmarked content*, the *encryption key* and the *watermark information* securely stay inside the Set-Top-Box that is only revealing watermarked media to the customer. Devices detected to be modified in any untrusted way, are banned from sharing media hence disabling attackers to infiltrate the device in order to gain access to shared media in an unmarked state.

II. Trusted Computing Principles

Trusted Computing technology as defined by the Trusted Computing Group [TCG06] is a technology implementing consistently behaving computer systems. This consistent behavior is enforced by providing methods for reliably checking a system's integrity and identifying anomalous and/or unwanted characteristics. These methods depict a trusted system's base of trust and thus are implemented in hardware, as it is less susceptible for attacks than software pendants.

To successfully realize stringently reliable modules, several cryptographic mechanisms are implemented on a hardware chip, namely Trusted Platform Module (TPM). This chip incorporates strong asymmetric key cryptography, cryptographic hash functions and a random number generator, that is capable of producing true random numbers instead of pseudo random ones. Additionally each trusted system is equipped with a unique key pair whose private key is securely and irrevocably stored inside the chip. The chip itself is the only entity to read and use this key for e.g. signing or encryption. This concept builds a powerful basement for approving and establishing system integrity since it allows to truly trustworthy let a trusted system sign data and to securely encrypt data for one specific trusted system. This is commonly used to measure system integrity and to ensure a system is and remains in a predictable and trustworthy state that produces only accurate results.

A. Trust for Measurement

The key concept of Trusted Computing is the establishing and extension of trust from an initially trusted security anchor up to further used components of a system while boot-up. Each component loaded while booting up the system is measured before execution by computing a SHA-1 digest value of it. The first component of this cycle acts as security anchor and has to be initially trusted, since it's integrity is not measured. This anchor is called *Core Root of Trust for Measurement* (CRTM) and is implemented as BIOS extension. It is executed after the very start of a system before any other BIOS code thus enabling to measure the BIOS and the platform's firmware. Each subsequent component involved in the boot-up process thereupon measures its successive component. Each measurement is stored in *Platform Configuration Registers* (PCR) on the TPM chip. These 160-bit registers are in the volatile storage on the chip and can exclusively be updated by calling the TPM command `TPM_EXTEND`. This command includes the old value of a register in the calculation of its new value thus preventing manipulation of registers.

$$\text{PCR}_i = \text{SHA-1}(\text{PCR}_i \mid \text{new value}) \tag{1}$$

This basically implements a non-commutative one-way function preventing from deleting and/or overwriting digest values in a PCR and enabling tracking of the chronological sequence values were applied to the register. This allows to analyze a system's state and furthermore prove its integrity by verifying integrity of any component loaded upon boot-up. This type of boot-up is also called *Trusted Boot Process*. Moreover the successive process of extending trust with each measure is commonly referenced to build up a *Chain of Trust*. To reproduce and verify a platform register's value in hindsight, every `TPM_EXTEND` command executed has to be tracked in a log. In the case of runtime measurement this has to be done by the operating system resulting in a log called *Stored Measurement Log* (SML). Since PCRs are located in the volatile storage of the TPM chip, each PCR is initialized with zeros upon system start and from there on is filled with measured data.

B. Trust for Reporting

Another main concept of Trusted Computing is Remote Attestation, a process to prove trustworthiness of a Trusted Platform to an external party. To verify a platform's integrity, a subset of PCRs together with the above-mentioned SML is sent to the external party. The PCRs values are then recalculated using the chronological order of measured components logged in the SML and a software implementation of the `TPM_EXTEND` command. In order to ensure integrity of the submitted PCR subset, it is signed by a unique TPM key pair. This key pair - namely *Endorsement Key* (EK) - is created on the chip at manufacture time. It is strictly unmodifiable and stored in the non-volatile storage on the chip. It is generated utilizing the `TPM_CreateEndorsementKey` command that creates the key pair *inside* the chip. Due to its specified uniqueness, once an EK is generated it can't be overwritten. Further calls of the `TPM_CreateEndorsementKey` command hence fail to ensure each chip holds one unique key pair. Its private key is furthermore secured from being read from outside the chip thus representing the so-called *Root of*

Trust for Reporting (RTR) - another security anchor based on hardware.

C. Trust for Storage

TPM chips are equipped with several cryptographic modules providing access to en-/decryption, hashing and key generation. This allows to securely generate cryptographic keys inside the TPM. Therefore a *RSA key pair generator* makes use of a *True Random Number Generator* that is capable of producing true random numbers hence generating true random RSA key pairs. These are thereupon stored outside the chip in a shielded storage. This storage is protected using a hierarchical encryption structure. Each private key of a generated key pair is encrypted with a parent key. The root of this tree-like key structure is represented by the last of three security anchors, the *Storage Root Key* (SRK). The SRK is a 2048-bit RSA key pair, that is created on the chip while setting up the TPM for a new owner. This is done using the `TPM_TakeOwnership` command. Like the EK it is unmodifiable and stored in the non-volatile storage on the chip, restricting the private key from being read from outside the chip. The SRK represents the *Root of Trust for Storage* (RTS) since it is used to securely store data and other keys outside the chip.

D. AIK Certification

Since each TPM is globally unique and thus identifiable and traceable, privacy issues arise when attesting a platform's state to external parties using Remote Attestation. In order to avoid this security issue, TPM chips provide for pseudonymity by allowing to generate temporary keys for attestation. These *Attestation Identity Keys* (AIK) can be created at any time using the `TPM_MakeIdentity` command and may be certified by a *Trusted Third Party* (TTP) to allow external parties to verify, that an AIK belongs to a TCG conform platform. AIKs can only be associated to their platform's EK by the TTP thus providing the platform with pseudonymity towards other entities. To issue an AIK credential, the platform has to send the EK-signed public key of a generated AIK key pair together with several credentials declaring the platform's TCG conformance to the TTP. After successful verification of the AIK and the platform's credentials, a particular data structure is sent to the platform. This structure contains the AIK credential and can be securely loaded only into the TPM that signed the initial request using the `TPM_ActivateIdentity` command.

III. Trusted Demonstration Environment

For the purpose of developing ideas and concepts of Trusted Computing, we set up a Trusted Demonstration Environment that was aimed to serve as TC playground that could be even used on systems lacking of physical TPM presence. This was realized by combining virtualization and TPM emulation. Since we were able to inspect debugging messages of the software emulated TPM device, emulation furthermore allowed us to gain deeper insight to TPM functions and to debug a TC framework we developed for convenient high-level access to common TPM processes and protocols.

TPM Emulator developed by M. Strasser [Str04] was used to emulate a TPM device by software. To furthermore replicate a system being in possession of a TPM even on boot-up, we decided to virtualize a guest system and pass the emulated TPM to it. This was done by utilizing a customized instance of QEMU [Bel08] that was enhanced with the ability of passing a TPM to the virtualized system.

A. TPM Emulator

To emulate a TPM *TPM Emulator 0.5.1* developed by Mario Strasser at the Swiss Federal Institute of Technology (ETH) Zurich and documented in [Str04] was used. It is a software based emulation of a TPM for Unix based systems implementing almost every function specified. At this point functions needed for demonstrating the concept of this thesis are fully implemented by TPM Emulator. Missing functions besides are perpetually added.

TPM Emulator furthermore provides an entire implementation of a TCG Device Driver Library (TDDL) allowing applications to pass commands to the emulated TPM. It implements three different components:

1) **tpmd:** a user-space daemon implementing a TPM with almost all functions, components and mechanisms specified by the [TCG06]
2) **tddl:** a device driver library for Unix acting as generic interface for accessing the emulated TPM
3) **tpmd_dev:** a kernel module simulating a physical TPM by providing the character device `/dev/tpm` forwarding all commands to the user-space daemon tpmd

Applications may access the emulated TPM by either passing commands directly to the user-space daemon (tpmd) or by handing commands over to the device driver library (tddl) or the kernel module (tpmd_dev). TPM Emulator thus acts transparently for applications providing realistic emulation of a physical present TPM device.

Unfortunately TPM Emulator won't operate before being loaded by the operating system hence ignoring the important boot process of a system. By omitting this part of the boot chain, the *Chain of Trust* is broken as trust has to anchor in a component being initially trusted. This is put into practice by basing trust on hardware and a BIOS extension that is loaded and executed right before any other software may compromise the system. TPM Emulator as an emulator executed by the operating system itself can't establish such a strong Chain of Trust because the operating system can't reliably be determined as trusted system. To overcome this issue and to demonstrate the abilities of a trusted system implementing a *Trusted Boot Process*, the emulated TPM is passed to a virtual machine thus becoming a physical device for the underlying system.

B. Virtualization of a TPM-equipped System

QEMU [Bel08] was used to virtualize a system fully equipped with a TPM device. In order to actually be able to pass a TPM device to QEMU we needed to modify it by applying a patch [Ble07] to the current version of QEMU

0.9.1. This patch allows QEMU to connect to a TPM via UNIX socket just like the one provided by TPM Emulator's tpmd.

To furthermore allow the virtualized system (we chose a standard Debian Etch distribution) we configured the kernel to support TPM devices (`CONFIG_TCG_TPM = y` and `CONFIG_TCG_ATMEL = y`). Additionally we enabled IMA, an implementation of the principle of Integrity Measurement for Linux operating systems that was introduced by IBM Secure Systems Department (`CONFIG_IMA_MEASURE = y` and `CONFIG_IMA_MEASURE_PCR_IDX = 10`). Integrity Measurement enables measurement and logging of executed software thus representing the operating system's mechanism for extending the Chain of Trust. Each system process and program is being measured before execution, added to PCR-10 and additionally logged to the SML.

Building on that it is possible to implement protocols for AIK Certification and Remote Attestation. We set up an infrastructure of several QEMU virtualized TPM-equipped clients inside a virtual network, which allowed us to demonstrate a complete client/server infrastructure of trusted platforms, trusted third parties certifying TP's AIKs and external parties attesting TP's system integrity.

C. Implementation of a client/server infrastructure

In order to pass TPM commands to the TPM in a convenient high-level way, we developed a framework based on jTSS [IAI08] in Java. jTSS is a Trusted Software Stack as defined by the TCG [TCG06]. Written in Java, jTSS is developed at IAIK Graz focusing on implementing a fully TCG-compliant Trusted Software Stack. One main feature is the ability to develop coexisting C/C++ and Java applications on a *TrouSerS* [Tro06] based system by letting Java applications use *jTSS Wrapper*, a Java wrapper for TrouSerS. It is concurrently updated and builds the base for the demonstration framework used in this work.

Our demonstration framework is able to create AIKs that can easily be certified by an implementation of a Privacy CA acting as TTP. This PCA is also part of our framework. According to [GR06] there is a serious security weakness in the AIK Certification process defined by the TCG. We thus implemented this process as proposed in [GR06] by inserting a handshake in the protocol. This ensures that the PCA communicates with the TPM that generated the submitted AIK. This prevents attackers from requesting credentials for arbitrary RSA keys from the PCA. This can easily be done by submitting intercepted data (public key of an EK and several credentials) to the PCA, which thereupon issues a certificate due to the false impression of actually communicating with a TPM. Even if the attacker won't be able to decrypt the delivered credential due to it being encrypted for the TPM owning the EK submitted, it allows for severe DoS attacks on the PCA.

We additionally implemented the Remote Attestation protocol as defined by the TCG. An instantiation of a Remote Attestation Server waits for clients to send a Stored Measurement Log together with PCR-10 signed by the AIK certified by the PCA in the previous step. The signature is thereupon verified by the server and the PCR-10 is re-computed using the given SML and a software implementation of the `TPM_EXTEND` command. If both PCR values, the one submitted and the one re-computed, are equal, the submitted SML has not been modified. The SML is then checked against a database of hashes for programs known to be trusted. If the SML contains any hash unknown to the server, attestation fails. This ensures that - until the moment of attestation - only trusted software was run on the TPM client. Modified software, malware or even viruses or trojans that would result in unpredictable system behavior thus can be detect. After successful attestation, the client receives an Attestation Certificate. This certificate can then be used to authorize against another service provider.

To further improve the framework and implement the concept presented in the following section, we added high-level access to more TPM functions like *binding*, *sealing*, *creating keys* and *certifying* them.

Unfortunately we were not able to access the emulated TPM device through the QEMU BIOS. This breaks the previously mentioned Chain of Trust, preventing the system to base trust in the CRTM. As soon as QEMU BIOS supports TPM functionalities, we will be able to establish the Chain of Trust from the very beginning of a system's boot-up process and thus anchor trust again on hardware.

IV. Concept

We will examine three different security aspects in our concept: a) *device and content integrity*, b) *customer privacy* and c) *copyright protection*.

A. Device and Content Integrity

As the Set-Top-Boxes are being hand over to customers, we have to ensure, that these devices operate in an unaltered trusted state. We establish device integrity using Trusted Computing concepts. TPM-Devices acting as a hardware trust anchor provide Remote Attestation, that reliably attests a devices' system state. Modifications of the devices' software and/or configuration will therefore be detected and state such devices being untrusted.

Each requesting device in the P2P network will have to prove itself as trusted, otherwise it will not receive any response. Remote Attestation Servers will receive TPM-signed system states from each device, check them against a set of system states being known as trusted and issue a certificate. These certificates will be valid for an arbitrary amount of time and are being requested and verified by each communication partner before sharing any content over the P2P network. This automatically ensures content integrity as only content intended by the content-provider is distributed over the P2P network.

B. Customer Privacy

Another issue to face is customer privacy. Serving as security anchor, each TPM-Device contains one global unique key pair for public-key cryptography called Endorsement Key. As

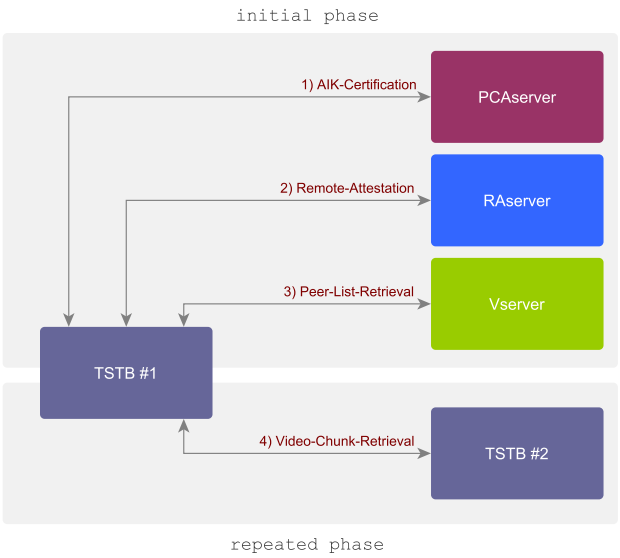

Fig. 1. Data Flow for retrieving videos

the before mentioned attestation certificate belongs to such a key-pair, it is possible to trace and track consumers' behaviors shortening the customers' privacy. To prevent this, we will make use of Attestation Identity Keys providing pseudonymity to some degree.

AIKs are derived from the EK with the private portion only available to the TPM, thus establishing a security chain with the TPM still being the anchor of trust. These key-pairs will be created periodically and are being used for Remote Attestation.

C. Copyright Protection

To protect copyrighted material from theft in a customer-friendly manner, we will make use of client-based digital watermarking methods. This reduces load on the provider side and balances it towards each Set-Top-Box participating in the media sharing process via P2P. Since media is watermarked on the client side after transmission over P2P, unmarked content is shared and thus has to be strongly encrypted using one unique key per shared medium before distribution.

This key and a customer-specific set of watermark information is cryptographically bound and transferred to each Set-Top-Box individually. Binding data is implemented using the TPM provided bind operation. This ensures that bound data can only be decrypted by and inside the TPM. Malicious users getting hold of both the encrypted content and the bound key/watermark information thus are prevented from getting access to unmarked content and watermark parameters. This on the one hand protects media from being stolen in an unmarked state and on the other hand prevents malicious users from wrongly accusing the victim of theft of copyright violation.

V. Trusted Watermarks

This section briefly describes the implementation of the concept illustrated in the preceding section.

A. Protocol Flow

As figure 1 shows, the retrieval of media consists of three processes along an *initial phase* and one process successively performed in a *repeated phase*.

a) AIK-Certification: After the customer chooses a media to be streamed, an AIK is being created inside the Set-Top-Box and requested to be certified by the Privacy CA (PCA). The TSTB therefore sends the public portion of the AIK signed with the EK together with several credentials constituting the TSTB's TCG-conformance to the PCA. There a random nonce is being generated, exclusively encrypted for the TPM (using the public part of the EK) and sent back to the client. The nonce is decrypted on the client side and sent back to the PCA. Only after successful verification of the nonce, the certification process is being started on the server side. This handshake modification [GR06] of the protocol specified by the TCG [TCG06] ensures that the PCA is communicating with the TPM that initially signed the AIK. After that the certification is encrypted using the public key of the EK and sent back to the client where it is decrypted and stored for further usage.

b) Remote-Attestation: In the next step the TSTB is being attested to be unmodified. Therefore PCR-10 is signed by the TPM's EK and sent to the RAserver together with the SML. There it is matched against a database of hashes for programs known to be trusted. If the SML contains any hash unknown to the RAserver and thus untrusted, the client receives no attestation hence ensuring the integrity of each TSTB participating in the P2P media sharing. In addition the PCR-10 is re-computed using the chronological history of measured programs in the SML and a software implementation of the `TPM_EXTEND` command. If both values appear to be equal the SML proved to be untampered and the TSTB is regarded *trusted*. An attestation certificate is being sent to the client, where it is stored for further usage. This certificate is only valid for a short period of time, in order to narrow the time an attacker is given to unnoticeable modify the TSTB. This is one major issue with Remote-Attestation, as the means of measurement provided by Trusted Computing only allow to attest a system's state at a certain time. It is impossible to assert a system to stay in the attested state, so limiting an attestation certificate's validity time-frame is the only way to lessen the impact of attacks.

c) Peer-List-Retrieval: Now that the TSTB is successfully attested, it is granted to receive a list of peers sharing the requested media. It therefore has to provide a valid attestation certificate to the Vserver that, after successful validation of the certificate, creates and sends a random nonce to the client. This nonce is incorporated in the process of creating a certified key using the `TPM_CertifyKey` command implemented by our demonstration framework. As AIKs are limited to be only used for signing, additional encryption keys have to be generated. These encryption keys can furthermore be certified, i.e. signed by the AIK. The result of the CertifyKey process is sent to the Vserver to be checked for a valid signature and the nonce provided in the previous step. After successful validation, an encrypted data blob containing several parameters for the

subsequent sharing process is sent to the client. It is bound to the TPM by encrypting it with the certified key and thus can only be decrypted inside the TPM. The data blob contains an *AES-Seed*, a *Watermark-Seed*, a *list of peers* and the *information* to be embedded into the shared media.
Using both seeds, keys for a) decrypting the shared media and b) basing the watermark process on can be generated. As all data is bound to the TPM and the TSTB is in a trustworthy state, there is no way for an attacker to gain access to it.

d) Video-Chunk-Retrieval: In the last repetitive step the actual media is shared among all peers. After successful retrieval of a video chunk, it is decrypted using the AES-key (that was created using the submitted AES-Seed) and marked with the corresponding watermark information based on the watermarking algorithm illustrated in the following section.

B. Digital Watermarking Algorithm

In this section we will briefly depict the Digital Watermarking Algorithm that was implemented to demonstrate the concept of this paper. The algorithm developed for this paper is more of a conceptual manner since it only serves for demonstration purposed. It thus lacks of efficiency and is very susceptible to attacks rendering it useless for being adopted outside a conceptual scope.

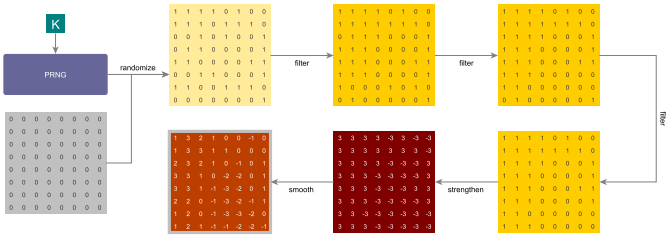

Fig. 2. Generation of a Fridrich Pattern

For simplicity reasons we decided to implement a watermarking algorithm for video content that is based on image watermarking. We therefore chose the MJPEG video format for our purposes, as it consists of full JPEG images for each movie frame thus representing a modern flip-book. To parse, manipulate and save MJPEG video files as well as JPEG images, we incorporated Sun's *Java Media Framework* [Sun99]. We implemented an image watermarking algorithm introduced by J. Fridrich [Fri97] and further developed by S. Thiemert [Thi02]. This algorithm applies 8x8 low-frequency patterns to the luminance values of an image. Two of those patterns (one for embedding a `0` and another one for embedding a `1`) P_0 and P_1 are generated using a process of different filtering and smoothing algorithms. This results in low-frequency patterns with low perceptibility (see figure 2).

In the following these patterns are converted to their frequency-domain representation by using a 2D discrete cosine transform (see figure 3) and applied to every single DCT-Block of an image/video-frame. To embed a bit-string into one frame, it is applied bit for bit in a sequential order to

$$F_{x,y} = \frac{2 \cdot C(x) \cdot C(y)}{N} \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f_{i,j} \cdot \cos\left(\frac{(2i+1) \cdot x \cdot \pi}{2 \cdot N}\right) \cdot \cos\left(\frac{(2j+1) \cdot y \cdot \pi}{2 \cdot N}\right) \quad (2)$$

$$C(n) = \begin{cases} \frac{1}{\sqrt{2}} & , n = 0 \\ 1 & , n \neq 0 \end{cases} \quad (3)$$

Fig. 3. Discrete Cosine Transform (DCT)

each DCT-Block of the frame. In order to improve imperceptibility of the embedded watermark, each block is analyzed using a so-called *Smooth Edge Detection* (SED) algorithm. This algorithm returns a good estimate of how *smooth* or *edgy* a block is. Smooth blocks consisting of mainly low-frequent parts, are ignored for the embedding process, as they would severely decrease imperceptibility of the watermark. Furthermore we customized the embedding algorithm to be only applied starting from a specific DCT coefficient. These are ordered in *ZigZagOrdering* as shown in figure 4. By only applying coefficients of the upper regions one can improve preserved image quality with degrading watermark robustness at the same time.

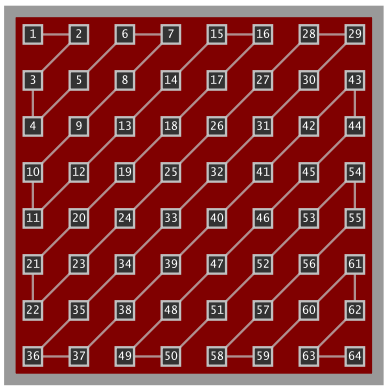

Fig. 4. ZigZag ordering

To furthermore embed an information into a MJPEG video, the information is split up into parts of 3 bits. These 3 bits are applied to one frame providing increased redundancy inside each of them. The 3-bit slices are randomly distributed over all video frames. In order to be able to recompose those slices in the readout-process, they are furthermore numbered with a 5-bit sequence number. The resulting 8-bit bit-string is then padded with a CRC-8 checksum to allow for verification of a successful read-out. This 16-bit bit-string is finally applied to one frame.

To read-out an embedded information from a watermarked video, each frame has to be inspected. To detect embedded bits, P_0 and P_1 are re-created using the process in figure 2. Both patterns are then cross correlated (see figure 5) against the blocks in the marked frame. The pattern resulting in a

$$Cor(b,p) = \frac{\sum_{i=f}^{g} b_i \cdot p_i}{\sqrt{\sum_{i=f}^{g} b_i^2 \ \cdot \ \sum_{i=f}^{g} p_i^2}} \tag{4}$$

$$f = \text{index to start from (ZigZag order)} \tag{5}$$

$$g = \text{dimension}^2 - 1 \tag{6}$$

Fig. 5. Cross Correlation

higher correlation factor is most likely to be embedded into this block. Again we customized the cross correlation equation to be started from a given index in ZigZag ordering. After reading out each bit/block of a frame, all read bits are averaged for their position in the 16-bit bit-string (due to redundancy of each bit inside the frame). The result is then verified against the CRC-8 checksum contained in the last 8 bits. By successively reading out every frame of the video, it is possible to re-assemble the information that was initially embedded into the video.

VI. Conclusion

We have shown a solution to improve media distribution by rerouting server load towards clients participating in the sharing of media. This reduces costs for powerful but expensive server infrastructures. We furthermore enhanced copyright protection of media by providing the customer with nonrestrictive copyrighted material that is also usable on other devices. In addition we could also implement means to trustworthy verify the integrity of a platform for the purpose of banning manipulated devices from sharing media. This prevents malicious customers from modifying their devices in order to gain access to unprotected media or even encryption keys or watermark parameters.

Risks that emerge from outsourcing the process of digital watermarking onto the customer's devices are limited or even eliminated by involving Trusted Computing technology. Moreover the so-called analog-hole - a typical exploit to bypass copyright protection - could be closed to some degree. This exploit uses DAD (digital-analog-digital) conversion of copyrighted media in order to remove copyright protection. As several watermarking algorithms provide good robustness against such attacks, the copyright protection presented in this paper is less prone to this exploit.

Future research will have to examine appropriate watermarking algorithms to fulfill requirements towards robustness, perceptibility and capacity. Shifting some work load from the clients to network load, novel digital watermarking techniques like Watermark Containers [WHS08] may be of interest. This concept presented by M. Steinebach, E. Hauer and P. Wolf introduces server prepared containers for media that is to be watermarked. These containers are encrypted in a way, that decrypting them using a specific key results in custom watermarked media. Providers thus are able to share this global container via P2P and deploy customer specific decryption keys. By decrypting the container with its specific key, a customer thus obtains media that is specifically watermarked.

Acknowledgment

This work was partially supported by the NanoDataCenters EU FP7 project.

References

[Bel08] Fabrice Bellard. *QEMU*. 2008.
[Ble07] T. Bleher. *Mailinglist [qemu-devel]: [PATCH] Add TPM support*. 2007.
[Fri97] J. Fridrich. *Methods for data hiding*. 1997.
[GR06] S. Gürgens and C. Rudolph. *AIK certification*. 2006.
[HSH08] K. Hall, R. Sailer, and S. Hallyn. *Linux-IMA: Integrity Measurement Architecture*. 2008.
[IAI08] IAIK. *Trusted Computing for the Java(tm) Platform*. 2008.
[Nan08] NanoDataCenters. *Results - Security Experimentation Environment*. 2008.
[Str04] M. Strasser. *A Software-based TPM Emulator for Linux*. 2004.
[Sun99] Sun. *Java Media Framework API Guide*. 1999.
[TCG06] TCG. *TCG Software Stack (TSS) Specification Version 1.2 Level*. 2006.
[Thi02] S. Thiemert. *Werkzeuge zur Qualitätsevaluierung und Vorschläge zur Optimierung von MPEG-Video-Wasserzeichen*. Diploma Thesis, Hochschule Anhalt (FH), 2002.
[Tro06] TrouSerS. *TrouSerS - The open-source TCG Software Stack*. 2006.
[WHS08] P. Wolf, E. Hauer, and M. Steinebach. *The Video Watermarking Container - efficient real-time transaction watermarking*. 2008.