Application of Trusted Computing to Secure Video Broadcasts to Mobile Receivers

Eimear Gallery, Allan Tomlinson, and Rob Delicata

Technical Report
RHUL–MA–2005–11
14 June 2005

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

This paper addresses the problem of configuring mobile devices to receive broadcast services protected by legacy conditional access systems. The protocols apply the concepts of trusted computing to allow a mobile host to demonstrate that it is secure, before any application or associated keys are securely downloaded. Thus the protocols are applicable anywhere a secure download is required. A general analysis of the security of the protocols is presented, followed by the results of formal verification.

1 Introduction

Recent developments in communications systems have given mobile devices the potential to receive complex multimedia content. It is expected that the next generation of mobile devices will be able to collaborate with broadcast networks to enable mobile access to broadcast content [1]. For a service like this to achieve its full commercial potential, the owners of the content will require assurance that their material is not illegally accessed. Current broadcast systems accomplish this by using conditional access systems to ensure that only bona fide subscribers have access to the content.

Each broadcast network is free to choose from a large number of commercially available conditional access systems. However, the security mechanisms and algorithms used by these systems are proprietary, and closely guarded by the system provider. Therefore, receivers may not switch freely between broadcast networks. This inflexibility presents an impediment to the reception of broadcast content by mobile receivers.

Standards exist that define an interface to proprietary systems at both the transmission site and at the receiver [2–5]. These standards were designed to give consumers and broadcasters some flexibility in their choice of equipment, and, while receivers remain static and consumers subscribe to one or two service providers, they provide a practical solution to this problem.

However, if a mobile subscriber requires access to services protected by a range of conditional access systems, then the current solutions become impractical. This paper proposes a flexible approach that allows consumer products to support a wide range of proprietary content protection systems.

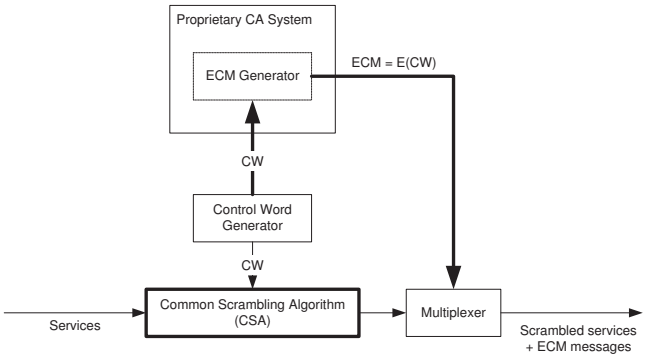

Figure 1: Video scrambling

2 Background

The Digital Video Broadcasting (DVB) organisation[1] has defined standards specifying two mechanisms to provide a degree of flexibility in the application of proprietary conditional access (CA) systems to broadcast services [2]. At the transmission site, the simulcrypt standard [3] allows a service to be controlled by two or more conditional access systems. At the receiver, the common interface standard [4] allows conditional access modules to be plugged into pc-card slots in the receiver to configure the device for the required conditional access system. Both systems rely on the service being scrambled using a standard common scrambling algorithm [5].

In the DVB system a service is scrambled using the common scrambling algorithm (CSA) under a key known as the control word (CW). Other systems use proprietary scrambling algorithms. However, in all cases the cryptographic scheme is a symmetric algorithm, and consequently the control word must be delivered to the receiver in a secure manner. This is the function of the conditional access system. At the transmission site, the control word is encrypted by the conditional access system and the encrypted control word $E(CW)$, or Entitlement Management Message (ECM), is broadcast to the receiver in advance of the scrambled service. This is illustrated in Fig. 1.

2.1 Simulcrypt

If a second conditional access system is available to the broadcaster, then the same control word may also be encrypted by this system. The simulcrypt standard [3], illustrated in Fig. 2, describes the interface to the conditional access systems. Both encrypted control words are then broadcast in advance

[1] www.dvb.org

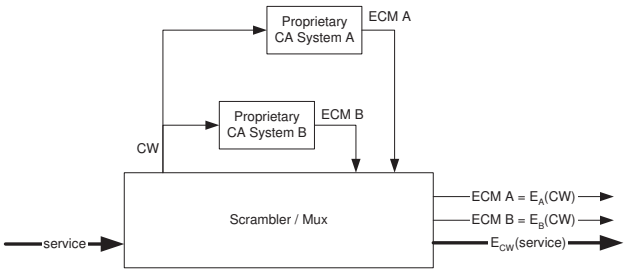

Figure 2: Simulcrypt

of the scrambled service. Thus, receivers running either conditional access system are able to decipher the control word and access the scrambled service. As far as the consumer is concerned, the operation of this system is completely transparent. The service provider, however, must operate multiple conditional access systems.

While it may be commercially feasible for large networks to operate two or three conditional access systems, the cost and logistics of running many such systems simultaneously could prove to be prohibitive for smaller networks.

2.2 Common interface

A parallel means of supporting multiple CA systems is to provide a solution at the receiver. This is accomplished by specifying a standard interface at the receiver that provides access to the scrambled service and the encrypted control words [4]. Fig. 3 shows how the scrambled service and the encrypted control word are passed to a separate pc-card module containing the hardware and software for a specific conditional access system. The encrypted control words are deciphered on the module to provide access to the service.

By swapping modules, the receiver can thus be configured to match the conditional access system used by the broadcaster. This system is therefore transparent to the broadcaster but not the subscriber. As is the case with simulcrypt, this mechanism may provide a solution for two or three conditional access systems, but, as the numbers increase, the cost to the subscriber could become prohibitive.

2.3 Limitations of existing mechanisms

The current solutions are well suited to current technology, where services are generally controlled by one or two conditional access systems and subscribers generally only require authorisation from one or two broadcasters. With

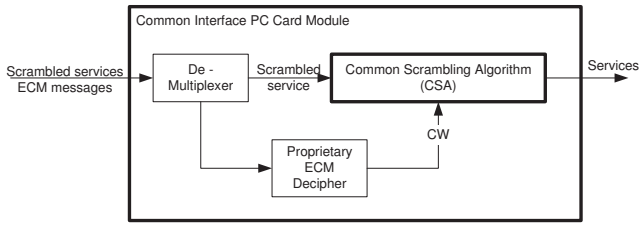

Figure 3: Common Interface module

a mobile receiver, however, subscribers will require authorisation from an increasing number of service providers as they move around.

The common interface solution would require mobile devices to have a pc-card interface and require the user to carry a number of modules. The cost of adding such an interface as well as the practical design issues could make this infeasible. The cost of the modules may also deter some subscribers. The alternative solution using current technology would be to broadcast each service under the control of as many conditional access systems as possible, which would prove expensive for many broadcasters, especially those operating in small niche markets. Both current solutions therefore have potential difficulties which could significantly restrict the content available to mobile receivers. These existing solutions do not transfer well to mobile systems and the problem of providing flexible conditional access would benefit therefore from some reconsideration in the light of new requirements.

2.4 Modifications required for mobile receivers

The goal is to provide access controlled broadcast content to mobile devices. This should be achieved with minimum impact on existing networks while at the same time minimising the hardware overhead on the mobile device and the cost to the user. An attractive solution is to allow a mobile receiver to be re-configured to be compatible with the appropriate proprietary conditional access system. The implication of this is that the proprietary application is implemented in software and delivered to the mobile device on demand.

The difficulty lies in convincing the software provider that the application, including any embedded secret keys, will be protected to at least the level offered by current solutions. This contrasts with the more familiar problem of securing the platform against malicious applications.

To address this problem, the delivery mechanism must protect both the integrity and confidentiality of the application. Moreover, the mobile device must be able to demonstrate to the software provider that the platform on which the application will execute is trustworthy. The former can be accom-

plished by well known security mechanisms such as encryption and the use of message authentication codes [6], while the latter may be achieved by the deployment of emerging trusted computing technologies. This is what we are proposing in this paper and the remaining sections describe the technologies used, the protocols to implement secure download, and an analysis of the security of these protocols.

3 Enabling standards and technologies

To provide a platform with the ability to prove its configuration to a challenger, and prove that it can be trusted to maintain that configuration, is a major research challenge. This challenge is being addressed by industrial and academic research groups and the principal results are described in this section. The suitability of these results for use in resolving the specific problems described in the previous section is also assessed.

3.1 The Trusted Computing Group

The Trusted Computing Group (TCG) [7], is an industry forum which is developing standards for a trusted computing platform. The platform specified by the TCG incorporates three fundamental components.

The first of these components is a **Core Root of Trust for Measurement (CRTM)** [8–10]. This component has the ability to measure one or more integrity metrics which reflect a portion of the software environment of the platform. These integrity metrics are recorded in one of sixteen Platform Configuration Registers (PCRs), located in the Trusted Platform Module (TPM).

The second component, the **Trusted Platform Module (TPM)** [8–10], is a tamper-resistant component responsible for recording and reporting the integrity metrics measured by the CRTM. The TPM also provides security functionality to the platform, such as platform attestation and protected and sealed storage.

The third fundamental component of a TCG compliant trusted platform is the **TCG Software Stack (TSS)** [11]. This is software on a trusted platform which supports the TPM. The TSS supports a stored measurement log, in which a record of all platform software that has been measured is stored. In conjunction with the stored measurement log, the TSS provides the functionality to support measurement agents and a trusted platform agent [12]. The measurement agents have similar functionality to that of the CRTM. They perform integrity measurements, but they do not constitute a root of

trust. The measurement agents must, therefore, be measured by the CRTM before they can be trusted and run. The trusted platform agent co-ordinates the supply of integrity metrics to challengers, i.e. third parties who wish to be given assurance regarding the current state of the trusted platform.

An important security service offered by a TCG compliant trusted platform is authenticated boot. During this boot process, the software state of the platform is measured and recorded in the Platform Configuration Registers in the TPM. For the problem addressed by this paper, we also make use of the platform attestation mechanism. This permits the state of the trusted platform's software configuration to be demonstrated to a challenger. To accomplish this, the TPM signs PCR values representative of the current state of the platform and, optionally, any external data provided as an input parameter to the attestation instruction.

TCG compliant platforms also provide confidentiality and integrity protection for stored keys, data, and application code. This is accomplished using the protected storage mechanism. In this instance, any binary object may be encrypted under a key which is itself protected in a key hierarchy. The root of this key hierarchy, the storage root key, is embedded in the tamper-resistant TPM.

Further protection is provided by a mechanism called sealing. Sealing associates an arbitrary binary object with specific PCR values (cumulative digests of software running on the platform) before encrypting the object. The encrypted object cannot subsequently be retrieved unless the current PCR values, representing the current software environment of the platform, match the required PCR 'digest at release' associated with the object. Input of correct authorisation data may also be required before access to the binary object is permitted.

3.2 Next Generation Secure Computing Base

In parallel with the Trusted Computing Group, Microsoft is developing the Next Generation Secure Computing Base, or NGSCB [13, 14]. Although details of the NGSCB specifications remain unpublished, Microsoft have produced technical reports [15–17] that provide an insight into the proposed architecture. NGSCB mandates the presence of a Security Support Component (SSC) which may be implemented by the TCG's version 1.2 compliant TPM. NGSCB also requires modifications to be made to the CPU, memory controller, graphics adaptor, and keyboard.

The NGSCB architecture incorporates three fundamental components, the first being a **Root Of Trust For Measurement**, similarly to the TCG's CRTM. Again, similar to the TCG, the second component is a **Se-**

curity Support Component or 'SSC'. This is a tamper-resistant chip required to provide cryptographic processing, a small amount of memory, and a monotonic counter. A module compliant with version 1.2 of the TCG's TPM specifications [8–10] is expected to serve as a potential SSC in the NGSCB architecture.

The **Isolation Kernel** is the third component. This enables several operating systems to execute in parallel on the same machine, and controls access to system resources. Through the deployment of the isolation kernel, mass market operating systems may be run in parallel to, but isolated from, one or more high assurance components (HACs). These high assurance components were previously known as 'nexuses', and are where trusted applications, previously called nexus computing agents (NCAs), are hosted.

The security services offered by the NGSCB architecture include confidentiality and integrity protection of stored data and applications. This is provided using the sealed storage mechanism offered by the NGSCB commands Seal/Unseal and PKSeal/PKUnseal [18]. The sealing mechanism ensures that a particular binary object is only accessible to the trusted application that created it, running on a specific instance of a particular HAC, or to a trusted application specifically permitted by the creator to have access to it.

NGSCB also provides attestation through the Quote command. This command instructs the SCC to sign a set of integrity metrics and, optionally, any external data provided as an input parameter to the command [18].

Secure input/output facilities are also provided by NGSCB, in which only trusted applications have access to input and output operations. These operations, therefore, cannot be observed or modified by any other processes. In a fundamental sense, in order to implement the above service, hardware changes to input and output devices are not strictly necessary. In principle, the isolation kernel could give control of the keyboard and the graphics card to a piece of trusted code, thus guaranteeing that input and output will not be observed or corrupted. There are two reasons, however, why new hardware for input and graphics is desirable. The first reason is to minimise the size of the Trusted Computing Base (TCB), which should ideally be kept as small as possible to preserve security. The second reason is is to maintain performance and ease of running off-the-shelf legacy operating systems which expect to have direct access to the graphics card. Although Virtual Machine Monitors (VMMs) routinely solve this problem by exposing a virtual graphics card to their guest operating systems, in practice this solution results in significant performance degradation.

Another important aspect of the NGSCB architecture is its memory protection mechanisms. The protected operating environment isolates secure areas of memory that are used to process data with high security require-

ments. These memory protection mechanisms are facilitated by the isolation kernel, using a page table edit control (PTEC) algorithm [18]. The memory controller or chipset must also be specially designed to provide protection against Direct Memory Access (DMA) attacks.

CPU modifications allow the newly designed isolation kernel to execute in a new ring, ring -1. This permits the execution of legacy operating systems with few modifications. It also means that problems usually associated with virtualising operating system instruction sets may be avoided.

To meet these needs, Intel's LaGrande Technology (LT) project [19] has developed a set of hardware components, including a processor and chipset extensions, keyboard, mouse and graphic subsystem enhancements and a TCG version 1.2 compliant TPM. These components will support an Intel architecture 32-bit platform that is able to provide the security services described above, such as protected execution (including prevention of DMA attacks), sealed storage, and platform attestation.

3.3 Open source architectures

In addition to the major trusted computing initiatives outlined above, parallel developments are taking place in the open source community. For example, the PERSEUS project [20] proposes a new architecture for a trusted platform which uses TCG/NGSCB hardware features in conjunction with an open source security kernel [21]. Marchessini et al. [22] also describe experiments where TCG hardware has been used to transform a desktop Linux machine into a virtual secure processor.

Two alternative architectures that may meet our objectives are the AEGIS processor architecture [23], and the executable only memory (XOM) architecture [24]. Both these projects propose architectures developed around security hardened processors.

3.4 Application to secure downloads

Balacheff et al. [12] and Sadeghi and Stüble [21] have both highlighted the potential of the TCG architecture for the secure delivery to and storage of content on, a trusted platform. Their schemes use a challenge-response mechanism so that the challenger (in our application the software provider) can be certain that the mobile host can indeed be trusted. The schemes then require encryption of the application by the software provider so that it is only retrievable by the platform that has been proven trustworthy.

There are, however, potential issues which arise with a trusted platform that deploys only the mechanisms described by the TCG. The main issue is

that there are no mechanisms described by the TCG to partition the system into trusted and untrusted environments, nor is there any mention of trusted operating systems or applications which may exist within these environments. On the face of it, one could take this to imply that the 'protected execution environment' we speak of encompasses *the entire platform*. Consequently, platform use could become very restricted. If the platform state is to be considered trustworthy, the challenger may insist that only a specific operating system and limited set of applications are running.

If platform use is to remain unrestricted, the challenger could be faced with the task of verifying a large set of complex integrity metrics. It is clear from the way PCR values are calculated that verification of an integrity challenge could become a difficult and overly complex task. In particular, this may become a problem if a challenged platform has been running for a long time, resulting in a large number of entries in the trusted platform measurement store. Verification may also be particularly complex if many different validation entities are being used by the challenged platform to certify the entries in the software measurement log.

Further problems arise if the host platform has deployed extensible software. PCR values should reflect every piece of software running on the trusted platform, and a PCR value containing the integrity metric for a piece of extensible software would therefore be very difficult to verify as 'correct' or trustworthy. The integrity mechanism deployed by the TCG specifications may ultimately mean that a very large number of different software states may have to be verified, which may be impractical.

A question also arises in relation to the protection of binary objects after they have been unsealed. When an application is unsealed we must consider the threats it may be exposed to during execution, for example tampering or replication.

Finally, there exists the possibility that malicious software may penetrate the system and bypass the operating system kernel via DMA. In this way, applications that have been measured by the CRTM may be overwritten by malicious code. Therefore, during platform attestation, a false impression of the software environment may be given to a challenger.

To overcome these limitations, the protocols proposed in sections 4.5 and 4.6 are based on a combination of the security mechanisms offered by NGSCB, LT hardware extensions, and the CRTM and TPM as defined by the TCG. This combination of security mechanisms is being proposed here for a number of reasons.

The CRTM and the TPM described in the TCG specifications are well defined, in the public domain, and have undergone a review process leading to the recent release of version 1.2 of the main TPM specification [8–10].

They facilitate authenticated boot, secure storage and platform attestation.

Memory protection mechanisms provided by the NGSCB isolation kernel and LT hardware extensions guarantee that the downloaded application can execute without interference or without external monitoring. By implementing an isolation kernel, as described by Microsoft, trusted partitions can be separated from the untrusted system partitions. Thus any legacy operating system and any untrusted applications may still run in parallel to trusted applications. The extra ring (ring -1) allows these memory protection mechanisms to be implemented in an efficient manner. Furthermore, the process of platform attestation or software integrity verification also becomes much simpler. The LT chipset extensions can prohibit system vulnerabilities caused by physical attacks (DMA) bypassing the integrity measuring mechanisms.

Therefore, in defining the protocols which follow, we assume that the trusted computing technology described above is available on the mobile host. This platform ideally consists of a TPM and CRTM as described in [8–10]; additional hardware extensions as described in [15–17] and [19]; and an isolation kernel as described in [18]. This will then enable the problem of securely downloading the conditional access application to be tackled, as described below.

4 Trusted download protocols

Having identified technology with the potential to solve the problem of proving that a platform is trustworthy, we now address the challenge of securing a downloaded conditional access application. Our two objectives are:

1. To protect the confidentiality and integrity of the application as it is transported from the software provider to the host platform.

2. To protect the application when it executes and is stored on the host platform.

4.1 Model

The model under consideration is illustrated in Fig. 4 and involves three parties: the user, who has a mobile receiver, M; the broadcaster, B; and the software provider, S.

4.1.1 Overview

The mobile user does not have a long term relationship with the broadcaster, but is aware of the services that are available. Some of these services may be

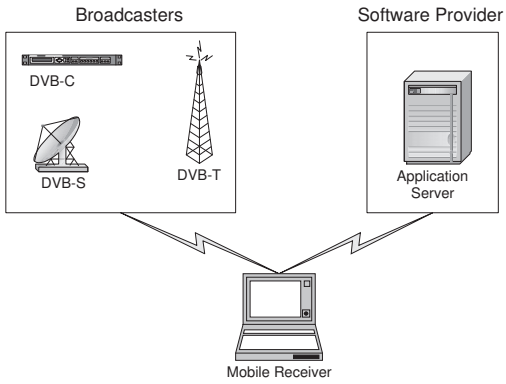

Figure 4: Model.

scrambled, in which case access is controlled by a conditional access system. For each scrambled service, the associated conditional access application, A_C, must be acquired by the mobile receiver. The protocol requires that the mobile receiver is able to demonstrate to the software provider that it is a bona fide receiver and is not in a malicious state that may attempt to illegally modify or replicate the downloaded application or keys. Once the receiver has proved itself to be legitimate, the application is made available only to that receiver. The protocol must then protect the confidentiality and integrity of the application as it is transported to the mobile device. Finally, the application must be protected on the platform itself.

The software provider in this model is required to supply the appropriate conditional access software to the mobile receiver. This software provider may, in practice, be the same entity as the broadcaster; alternatively it may be a third party broker who provides the application, A_C, to the receiver. The mobile user needs to be aware of which software provider can deliver the application, A_C. He may be informed of this by either the broadcaster or the software provider. The mobile receiver is therefore in a position to download A_C from the software provider and descramble the broadcast service, subject to the appropriate commercial agreements.

4.1.2 Assumptions about M

We assume the presence of a TPM as described in version 1.2 of the TCG specifications. We also assume the presence of chipset extensions which allow memory protection to prevent physical attacks via DMA, and we assume the presence of an isolation kernel. Thus the mobile receiver M can support multiple execution environments or system partitions. Within a protected

partition, applications may run in isolation, free from being observed or compromised by software running in any other partition. Trusted applications running in the protected environment have access to the TPM.

Within this protected partition, a trusted broadcast application, A_B, and a trusted download application, A_D execute upon a trusted operating system. It is this protected environment into which the conditional access application, A_C, will be downloaded and executed.

The download application, A_D, will perform two fundamental tasks. It will complete one of the protocols described below. It will also prevent the potential interference of any other application with A_C while it is executing. It may, for example, incorporate a monitoring function which adheres to a specified policy, such that once the conditional access application is running on the device, any attempt by another application to start up will fail; alternatively the start-up of any additional applications will result in A_D stopping A_C, and erasing it from memory.

We assume that all secret keys required by the mobile receiver are protected by and are only accessible via the TPM.

A unique asymmetric encryption key pair, called the endorsement key pair, is associated with the TPM. The private endorsement key is securely stored in the TPM. The associated public key is certified, and the certificate contains a general description of TPM and its security properties.

Credentials are generated indicating whether the particular design of TPM in a particular class of mobile platform meets specified security requirements, and whether a particular mobile host is an instance of this class of mobile platform and incorporates a specific TPM which meets this design.

A private signing key is securely stored by the TPM. This key is used only for entity authentication. The public signature verification key corresponding to this private key is certified by a certification authority, CA. The certificate issued, Cert_{TPM}, binds the identity of the TPM (the trusted platform containing the module) to a public key used for the verification of digital signatures. This certificate must be obtainable by the software provider S.

The TPM is capable of generating an asymmetric encryption key pair, where the public encryption key is signed/certified using the signature key described above. This thwarts the privacy and security threats surrounding routine use of the public asymmetric encryption key. The private decryption key from this pair is bound to a particular environment configuration state.

4.1.3 Assumptions about S

We assume that the software provider, S, has a private signing key that is securely stored within its environment and that this key is used only for

entity authentication. We also assume that S, has a certificate, Cert_S, issued by the certification authority, CA. This certificate associates the identity of the S with a public key value for the verification of digital signatures. This certificate must be available to, and verified by, the mobile platform, M. Similarly, we assume that S is in possession of a valid Cert_{TPM}. Finally, we assume that S is able to verify the claims made by the platform containing a particular TPM. In other words, S is able to look up, or obtain from a validation authority, an expected set of trustworthy integrity metrics.

4.2 Notation

The following notation is used in the specification of the protocols:

M	denotes the mobile receiver
B	denotes the broadcaster
S	denotes the software provider
CA	denotes a certification authority trusted by S and M
TPM	denotes a TPM embedded in the mobile receiver M
A_B	denotes a trusted broadcast application
A_C	denotes the conditional access application which is downloaded
A_D	denotes a trusted application/agent responsible for the secure download of conditional access software
Cert_X	is a public key certificate for entity X
$K_{X,Y}$	denotes a secret key possessed only by X and Y
R_X	is a random number issued by entity X
$E_K(Z)$	is the result of the symmetric encryption of data Z using the key K

$\text{Seal}_I(Z)$	is the result of the encryption of data Z concatenated with integrity metrics, I, such that Z can only be deciphered and accessed if the software state of the platform is consistent with I
$H(Z)$	is a one-way hash function of data Z
$\text{MAC}_K(Z)$	is a Message Authentication Code, generated on data Z using key K
$X(public)$	is the public asymmetric key of X
$X(private)$	is the private asymmetric key of X
$S_X(Z)$	is the digital signature of data Z computed using entity X's private signature transformation
p	is a prime number
g	is a generator for Diffie-Hellman key exchange modulo p, i.e. an element of multiplicative order q (a large prime dividing $p-1$) modulo p
a_X	is entity X's Diffie-Hellman private key agreement component
b_X	is entity X's Diffie-Hellman public key key agreement component $= g^{a_X} \bmod p$
$X\|\|Y$	is the result of the concatenation of data items X and Y in that order
Id_X	is the identity of X
$X \rightarrow Y : Z$	indicates that message Z is sent from X to Y

4.3 TPM commands

We imply the use of the TPM command set and data structures as specified by the TCG. TPM commands and data structures used in the protocols include $TPM_{CreateWrapKey}$, $TPM_{CMKCreateKey}$, $TPM_{CertifyKey}$, TPM_{Quote}, TPM_{Seal}. We will also utilise the exclusive transport session mechanism, as described in version 1.2 of the TCG specifications [8–10].

Commands for the public key protocol: The $TPM_{CreateWrapKey}$ command is used in step 3 of the public key protocol to instruct the TPM to generate the asymmetric key pair $A_D(public)$ and $A_D(private)$. The input parameters associated with the $TPM_{CreateWrapKey}$ command include information about the key to be created, e.g. key length, and authorisation data necessary to access the parent wrapping key. Encrypted authorisation data for the newly generated key pair may also be input if required.

For this particular use case we require that the key to be created is non-migratable. This implies that the key cannot be migrated from the TPM in which it is created.

Alternatively, using a new command described in the TPM version 1.2 specifications, a certifiable migratable key may be created using the $TPM_{CMKCreateKey}$ command. This creates a migratable key which may be certified by the TPM and migrated only under strict controls. This prohibits the key protecting the conditional access application from being migrated to any random platform authorised by the TPM owner, but facilitates its migration to select devices, e.g. other TPMs owned by the same entity. Before key migration, the key owner must authorise the migration transformation. The migration destination must be authorised not only by the TPM owner but also by a migration selection authority. This authority may, for example, be the trusted download agent, A_D. We will focus, however, on the case where the key to be created is non-migratable.

In response to the $TPM_{CreateWrapKey}$ command, the TPM returns a *TPM_Key* data structure. This data structure contains $A_D(public)$, and the encrypted private key, $A_D(private)$. The data structure also identifies the operations permitted with the key, and contains flags to indicate whether or not the key is migratable. The data structure may also identify the PCRs to which $A_D(private)$ is bound, and may include the PCR digests at key creation and the PCR digests required for key release. The PCR data provides the integrity metrics, I, used in the protocols.

In our application, S will require that the returned PCR digest at creation reflects a trusted execution environment which consists of correctly functioning broadcast and download applications running on a trusted operating system. The required PCR digest at release could be inserted by A_D as an an input parameter to the $TPM_{CreateWrapKey}$ command. Verification of the returned PCR digest at creation (reflecting correctly functioning A_B and A_D running on the trusted operating system) would ensure that all protocol steps, and the value input by A_D as the digest at release, can be trusted to be correct. The value input into the digest at release by A_D should be checked by S before A_C is dispatched.

The PCR digest at release could also reflect a state in which a particular operating system, a particular video player, and a particular download application, are running, but nothing more. The PCR values describe the state of the execution environment at key creation and that required for key release. However, if this data is to be communicated to the challenger, S, proof must exist that the data originated from a genuine TPM and that it has not been replayed. This is discussed below.

The final part of the *TPM_Key* structure to consider is the *TPM_Auth_Data_Usage* field. This field may take on one of the following values: *TPM_Auth_Never*; *TPM_Auth_Always*; or *TPM_Auth_Priv_Use_Only*. In our scenario, it is A_D that must access the private key to decipher A_C.

The first option is to permit A_D to load the private key without the submission of any authorisation data. In this case the *TPM_Auth_Data_Usage* field is set to *TPM_Auth_Never*. Alternatively, the *TPM_Auth_Data_Usage* field could be set to *TPM_Auth_Always* or *TPM_Auth_Priv_Use_Only*, where, on key pair generation, 20 bytes of authorisation data are associated with the public/private key pair, or with just the private decryption key, respectively.

To facilitate this, before a request for key pair generation, A_D could request that the user provides a password, from which the authorisation data for private key use would be derived. Thus, when access to the private decryption key is required, the correct password would have to be re-entered by the user. This option is acceptable provided that user interaction with A_C is feasible.

Once a key pair has been created using the $TPM_{CreateWrapKey}$ command, the handle associated with this key can be given to the TPM in a $TPM_{CertifyKey}$ command. 160 bits of externally supplied data which, in this protocol, is used to submit a one way hash of R_S, and Id_S may also be given as an input parameter to this command. In response, the TPM returns a *TPM_Certify_Info* data structure. This structure describes the key that was created, including authorisation data requirements, a digest of the public key, and a description of how the PCR data is used. In addition to this structure, the TPM also signs and returns a hash of the public key digest, the 160 bits of external data, and the PCR data contained in *TPM_Certify_Info*, respectively.

Commands for Secret key protocol: For the secret key protocol defined in section 4.6, the TPM *exclusive transport session* feature is required. Any application can request the establishment of an exclusive transport session with the TPM. Once an exclusive transport session has been established, any TPM command made from outside the application will result in that exclusive session being terminated. Any changes to the platform configuration automatically raise a TPM_{Extend} command to update the PCRs. Therefore, once an exclusive transport session has started, any alteration of the platform state will result in a premature termination of the session, which is detected by the application that established it. This feature is used in step 3 of the following protocol in order to ensure that the platform state cannot be changed after the agreed keys have been sealed.

The TPM_{Seal} command is used in step 4 to store the keys $K1_{S,A_D}$ and $K2_{S,A_D}$ in protected memory. The parameters taken by this command include the data to be sealed, and the authorisation data necessary to unseal the data. The TPM_{Seal} command is also given information identifying the PCRs whose value is to be bound to the protected data. In response, the TPM returns a TPM_Stored_Data structure. Specifically, this data struc-

ture contains a reference to the PCRs that the data is bound to, and a TPM_Sealed_Data structure. The latter contains the encrypted data and the authorisation requirements to access the data.

For this particular use case we require that the key used in the seal operation is non-migratable. This implies that the private key cannot be migrated from the TPM in which it is created.

Alternatively, using a new command described in the TPM version 1.2 specifications, a certifiable migratable key, created with the $TPM_{CMKCreateKey}$ command, may be used in the seal operation. This migratable key may be certified by the TPM and migrated, but only under strict controls. This prohibits the sealing key, which essentially protects the conditional access application, from being migrated to any random platform authorised by the TPM owner, but facilitates its migration to select devices, e.g. other TPMs owned by the same entity. We will focus, however, on the case where the sealing key is non-migratable.

Finally, the TPM_{Quote} command instructs the TPM to attest to the platform configuration. The parameters given to this command include the indices of the PCRs defining the platform integrity metrics, I. The TPM_{Quote} command may also be given 160 bits of externally supplied data which, in this protocol, is used to submit a one way hash of b_M, b_S and Id_S for attestation.

4.4 Protocol initiation

Both protocols begin when the user makes a request to the broadcast application, A_B, to view a specific video broadcast. If reception of this broadcast is controlled by a particular conditional access application, A_C, then A_B checks to see if the mobile device already has dedicated hardware or software installed to support A_C. If no dedicated hardware exists on the mobile device, then A_B determines whether A_C has previously been downloaded, and is still available in secure storage. If so, the download application, A_D, is called to retrieve A_C from secure storage and execute the application. If A_C is not available on the mobile device, then A_D is called to download the application which can be accomplished by deploying one of the following two protocols.

One of the two protocols below are completed every time a conditional access application is to be downloaded. Thus, either the asymmetric encryption key pair generated in the public key protocol or the Diffie-Hellman key agreed in the secret key protocol are unique to each protocol run.

4.5 Public key protocol

The protocol is illustrated in Fig. 5 and described in the following.

1. $A_D \rightarrow S:$ Request for A_C
2. $S \rightarrow A_D:\ R_S$
3. $A_D \rightarrow TPM:\ TPM_{CreateWrapKey}$
4. $TPM \rightarrow A_D:\ TPM_{Key}$
5. $A_D \rightarrow TPM:\ TPM_{CertifyKey}$
6. $TPM \rightarrow A_D:$
   $TPM\_Certify\_Info \parallel S_{TPM}(H(A_D(public)) \mid\mid H(R_S \mid\mid Id_S) \mid\mid I\ )$
7. $A_D \rightarrow S:$
   $R_S \mid\mid Id_S \mid\mid TPM_{Key} \mid\mid TPM\_Certify\_Info$
   $\parallel S_{TPM}(H(A_D(public)) \mid\mid H(R_S \mid\mid Id_S) \mid\mid I\ )$

S now verifies the signature on the data received, $S_{TPM}(H(A_D(public)) \mid\mid H(R_S \mid\mid Id_S) \mid\mid I\ )$, checks R_S to ensure the message has not been replayed and Id_S to ensure that the message was destined for S. Assuming that the signature, nonce, and Id_S are correct, S then verifies the integrity metrics, I. If I describes a trustworthy platform, then S generates $K1_{S,A_D}$ used for data encryption, and $K2_{S,A_D}$ used for data integrity protection.

8. $S \rightarrow A_D:$
   $E_{A_D(public)}(K1_{S,A_D} \| K2_{S,A_D}) \mid\mid S_S(E_{A_D(public)}(K1_{S,A_D} \| K2_{S,A_D}))$
   $\mid\mid E_{K1_{S,A_D}}(\mathrm{MAC}_{K2_{S,A_D}}(A_C))$

On receipt of the above message, A_D verifies $S_S(E_{A_D(public)}(K1_{S,A_D} \| K2_{S,A_D}))$ and if the signature is valid, instructs the TPM to decipher $E_{A_D(public)}(K1_{S,A_D} \| K2_{S,A_D})$. A_D then deciphers $E_{K1_{S,A_D}}(\mathrm{MAC}_{K2_{S,A_D}}(A_C))$ and verifies $\mathrm{MAC}_{K2_{S,A_D}}(A_C)$. Once the MAC is verified, the application can be executed. During execution, A_D precludes the potential interference of any other application with A_C, and after execution A_D deletes A_C, and all other keys, when they are no longer required. The encrypted copy of A_C may remain stored for future use, space permitting.

4.6 Secret key protocol

The second protocol outlined below uses symmetric encryption as opposed to asymmetric techniques. In this protocol, the software provider's certificate, Cert_S, contains public Diffie-Hellman parameters g and p, and each protocol runs begins with A_D choosing a Diffie-Hellman private value a_M and calculating b_M based on g and p which are retrieved from Cert_S.

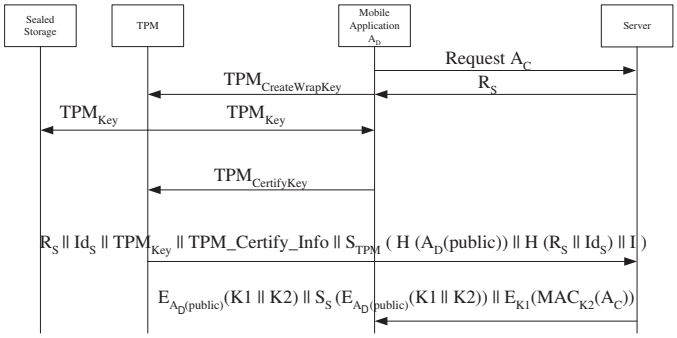

Figure 5: Public key protocol

1. $A_D \rightarrow S:$ Request for $A_C \parallel b_M$

On receipt of this message S chooses a new Diffie-Hellman private value a_S, and calculates b_S using g and p.

2. $S \rightarrow A_D:\ b_S \parallel S_S(b_S \parallel b_M \parallel Id_M)$

A_D then verifies the signature on the message received and calculates the shared key K_{S,A_D}. This key is then used to derive, by an agreed method, $K1_{S,A_D}$ which will be used for data encryption and $K2_{S,A_D}$ which will be used for data integrity protection.

3. $A_D \rightarrow TPM:$
   Request for exclusive transport session.

4. $A_D \rightarrow TPM:\ TPM_{Seal_I}(K1_{S,A_D} \| K2_{S,A_D})$

5. $A_D \rightarrow TPM:\ TPM_{Quote}(H(b_M \parallel b_S \parallel Id_S))$

6. $TPM \rightarrow A_D:\ S_{TPM}(H(b_M \parallel b_S \parallel Id_S) \parallel I)$

7. $A_D \rightarrow S:\ S_{TPM}(H(b_M \parallel b_S \parallel Id_S) \parallel I)$

S then verifies the signature on the data received,
$S_{TPM}(H(b_M \parallel b_S \parallel Id_S) \parallel I)$, and verifies $H(b_M \parallel b_S \parallel Id_S)$ to ensure the message is fresh and was destined for S. S then decides if I represents a sufficiently trustworthy state. Given I, S can verify that the A_D is executing as expected, that it has not been tampered with, and that there is also a legitimate broadcast application executing on the mobile host. Thus S can be sure that an exclusive transport session has been set up between A_D and the TPM, and be confident that $K1_{S,A_D}$ and $K2_{S,A_D}$ have been sealed to the PCR data defined by I. S then calculates K_{S,A_D} and derives $K1_{S,A_D}$ and $K2_{S,A_D}$ by the agreed method.

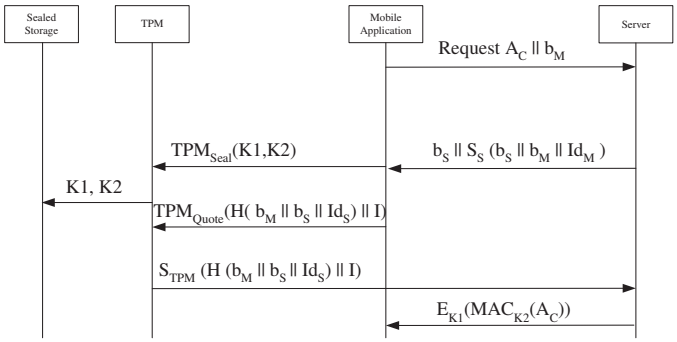

Figure 6: Secret key protocol

8. $S \rightarrow A_D : \ E_{K1_{S,A_D}}(\text{MAC}_{K2_{S,A_D}}(A_C))$

On receipt of the above message, A_D, instructs the TPM to unseal $K1_{S,A_D}$ and $K2_{S,A_D}$. The TPM can only unseal $K1_{S,A_D}$ and $K2_{S,A_D}$ if the platform is in the agreed state. Furthermore, authorisation data may also be required depending on the parameters set when sealing the symmetric keys in step 4. Once the keys are unsealed, A_D may then decipher $E_{K1_{S,A_D}}(\text{MAC}_{K2_{S,A_D}}(A_C))$ and verify the MAC. If the MAC verifies correctly, the application is then executed. During execution, A_D precludes the potential interference of any other application with A_C. After execution, A_D deletes A_C, and all keys. The encrypted copy of A_C may remain stored for future use, space permitting.

5 General security analysis: Secure transmission

We initially examine both protocols with respect to our first objective: protection of the confidentiality and integrity of the application as it is transported from the software provider to the host platform.

5.1 Public key protocol

Confidentiality: Symmetric encryption is deployed to protect the confidentiality of the A_C. The symmetric key is securely transported to the mobile host under a public encryption key of an asymmetric key pair. This key pair is inextricably linked to the requesting trusted mobile host where the private decryption key is securely stored. Assuming that the keys are securely managed by the software provider, an attacker cannot intercept and gain read access to the conditional access application unless there is a hardware-based

attack on the mobile host. This would require the extraction of the private decryption key from the TPM.

Integrity: A message authentication code is deployed to protect the integrity of A_C in transit. The MAC key is protected by the same technique used to protect the symmetric encryption key, described above. Thus, assuming secure MAC algorithms are used, integrity protection depends on the security of the tamper proof TPM.

Entity authentication: The software provider can verify the identity of the TPM and the state of the protected execution environment which utilises TPM security services, via the platform attestation mechanism. That is, the signature of the TPM on R_S, Id_S and on the integrity metrics representing the protected execution environment guarantees the platform's state and allows the software provider to authenticate the trusted platform. Step 2 and step 7 of the above protocol conform to the two pass unilateral authentication protocol described in clause 5.1.2 of ISO/IEC 9798-3:1998 [25], where $A_D(public)$ serves as the nonce in the response message sent by M by virtue of the fact that the asymmetric key pair is generated for each protocol run.

It may be argued that the protocol outlined above also provides entity authentication with respect to the software provider. Since a unique $A_D(public)$ is generated for each protocol run, $A_D(public)$ acts as not only a random nonce but also represents the identity of the destination platform. The signature of the software provider on the unique public key, $A_D(public)$, provides entity authentication.

Alternatively, one of the following additions may be made to the protocol. A random nonce may be included in the signed bundle sent to the software provider in step 7 and returned in conjunction with R_S and Id_M in the bundle signed by the software provider in step 8. If this modification is made to the protocol, steps 2, 7, and 8 conform to the three pass mutual authentication protocol described in clause 5.2.2 of ISO/IEC 9798-3:1998 [25].

Instead of this, a timestamp, in conjunction with the Id_M, could be included in the signed bundle from step 8. If this modification was made step 8 would conform to the one pass unilateral authentication protocol as described in clause 5.1.1 of ISO/IEC 9798-3:1998 [25].

Origin authentication: Since S signs $K1_{S,A_D}$ and $K2_{S,A_D}$, M is able to verify that these keys have been sent from S. Also, since $K2_{S,A_D}$ is used to compute the MAC, M can thus verify that the A_C has been sent from the same source. An attacker attempting to deliver a malicious application would require the collaboration of S.

Replay: Platform attestation using the integrity metrics proves that the platform is in a specific configuration, and that S is communicating with a particular trusted module. This is only useful if proof exists that

communications are fresh. This is accomplished by including R_S in the signed bundle sent to S in step 7.

It may be possible for an attacker to replace the message in step 8 with an older message destined for the same mobile host, or with a corresponding message destined for a different mobile host. However, since a unique $A_D(public)$ is generated for each protocol run, the verification of the MAC at the end of the protocol would detect this.

To detect replay earlier, a random nonce may be included in the signed bundle sent to the software provider in step 7 and returned in the bundle signed by the software provider in step 8. Alternatively, a timestamp could be included in the signed bundle in step 8.

5.2 Secret key protocol

Confidentiality: As was the case in the first protocol, symmetric cryptography is deployed to protect the confidentiality of the conditional access application. However, rather than using the public encryption key of the mobile host to securely transport the symmetric keys, authenticated Diffie-Hellman is used to agree upon a symmetric key. Since key agreement is authenticated, this protocol is safe from man in the middle attacks during the key agreement process.

Integrity: As was the case in the first protocol a message authentication code is deployed in order to protect the integrity of the conditional application in transit, where the MAC key is agreed using an authenticated Diffie-Hellman protocol [26].

Entity authentication: As this protocol is based on the station-to-station protocol [26](a type of authenticated Diffie-Hellman protocol) it offers mutual authentication. The software provider can verify the identity of the TPM and the state of the protected execution environment which utilises TPM security services. This may be accomplished via the platform attestation mechanism, i.e. the verification of the $TPM's$ signature upon b_M, b_S, Id_S and the integrity metrics representative of the protected execution environment. The mobile device can authenticate the software provider identity via the verification of the digital signature which has been generated over b_S, b_M and Id_M.

Origin authentication: Since S signs b_M and b_S in step 2, and A_C is protected by using keys derived from b_M and b_S, then M is able to verify the origin of A_C. As before, an attacker attempting to deliver a malicious application would require collaboration from S.

Replay: The inclusion of a unique, freshly generated b_S in step 2 prevents an attacker from blocking the message in step 7 and sending an older version

captured from a previous protocol run.

The attacker could replace the message in step 8 with an older message destined for the same mobile host, or with a corresponding message destined for a different mobile host. However, since a unique K_{S,A_D} is generated for each protocol run, verification of the MAC at the end of the protocol would detect this. As additional measure, if deemed necessary, a timestamp may be integrated into step 8 to prevent replay.

6 General security analysis: Secure execution

We now examine both protocols with respect to our second objective, namely protection of the application when it reaches the host platform. Both protocols are the same in this respect, and rely on the security mechanisms deployed by the trusted platform. Our analysis considers the confidentiality and integrity protection of the application and on the host and the prevention of unauthorised access to the application while in storage, and during execution.

Security in storage: The confidentiality and the integrity of the application while in storage on the mobile platform are protected via the same mechanisms used to protect the application in transit: MACing and symmetric encryption.

In the public key protocol, these MACing and encryption keys are encrypted under a public key, where the corresponding private decryption key is securely stored by the TPM and is non-migratable.

If either the encrypted and MACed application or the encrypted MACing and encryption keys are captured, they will remain privacy protected due to encryption. If, on the other hand, either the encrypted and MACed application or the encrypted MACing and encryption keys are modified, the MAC upon the application will fail.

In order to prevent unauthorised access to the private decryption key, two further measures may be taken. The private decryption key may be bound to a specific execution environment state such that this key may not be loaded until the current environment configuration matches that to which the private key is bound.

In conjunction with this, twenty bytes of authorisation data may be stored with the private decryption key. However, a problem arises regarding where this authorisation data may be stored. It may be securely stored by the TPM, i.e. encrypted and bound to a specific execution environment but this offers

no additional protection than if the first access control mechanism described above is used in isolation. This is an important issue, but one that is not dealt with in the TCG specifications.

Nevertheless, as an alternative option, it may be relatively straightforward for a *user* to provide the necessary password, during key pair generation, from which authorisation data is derived. This option may be acceptable so long as user interaction with A_D is permitted and there is a secure link between the user entering the password and the TPM.

In the secret key protocol, the MACing and encryption keys are directly encrypted by the TPM and bound to a specific execution environment state (sealed). Twenty bytes of authorisation data may also be associated with the sealed MACing and encryption keys for more stringent control against unauthorised access.

No integrity mechanisms are offered by the TPM-protected storage functionality. However, if the encrypted MACing and encryption keys are modified, the MAC on the application will not be verifiable.

Security during execution: The NGSCB architecture facilitates system partitioning through the implementation of an isolation kernel which then exposes the original hardware to one guest OS. This system partitioning also enables simpler PCR verification, as the number of applications running on a trusted operating system in a particular protected execution environment may be strictly controlled. With respect to memory partitioning, the NGSCB isolation kernel uses an algorithm called PTEC to partition physical memory among guests. This is described in further detail in [18] but from a security perspective, protection is analogous to that of traditional virtual memory protection methods such as those that use translation lookaside buffers or page tables.

To facilitate efficient implementation of parallel environments, a new CPU mode has been introduced such that the isolation kernel may run in a new ring -1, and guest operating systems can still execute in ring 0. This avoids problems in relation to the virtualisability of particular OS instruction sets which may arise if a virtual machine monitor were to be deployed in ring 0.

The memory protection mechanisms described above, while protecting the application during execution, offer no protection against an attacker attempting to subvert or bypass the operating system kernel via DMA. Such an attack could lead to the execution of malicious code that has not been recorded in the PCRs, or alternatively bypass virtual memory protections, described above. To prevent this type of attack, Microsoft have encouraged chipset manufacturers to make certain changes to the hardware. An access control policy map is then defined by software (for example the isolation kernel) and stored in main memory. This policy map then indicates whether

a particular subject (DMA device) should be able to access (read or write) to a particular resource (physical address). This policy map is enforced by hardware.

Within the protected environment, where A_B, A_D and A_C execute, A_D, as defined, will also prevent the potential interference of any other application with A_C while it is executing. A_D may, for example, incorporate a monitoring function which adheres to a specified policy, such that once the conditional access application is running on the device, any attempt by another application to start up will fail. Alternatively the start-up of any additional applications will result in A_D stopping A_C, and erasing it from memory. Once the application has run, it is either deleted or the encrypted copy is stored for future use. Recovery of A_C from encrypted storage would require extraction of the keys from the TPM.

7 Formal analysis

Our protocols are conceptually simple: they consist of a challenge-response message exchange followed by the transmission of the (encrypted and MACed) application. Nonetheless, experience has shown that even simple protocols can harbour subtle errors, and we must be careful in using informal arguments to justify their correctness. In this section we outline a formal analysis that has been carried out to establish that the protocols satisfy their primary security goal. Full details of this analysis can be found in a University of Surrey technical report [27].

7.1 Analysis approach

Our analysis uses an established formal verification technique, first described by Schneider [28], based on a process algebraic model of the protocol. There are three components to the approach:

- The protocol is captured in the process algebra Communicating Sequential Processes (CSP) [29] in terms of the behaviour of each participating agent.
- The environment is also described as a CSP process. This is generally an unreliable medium which may lose, reorder or duplicate messages.
- Requirements on the protocol are expressed as a specification of the observable behaviour of the system.

When these three components are brought together, well established proof-techniques can be brought to bear on the task of proving that the model satisfies the requirements of the protocol. In the next section we briefly state some modelling assumptions and justify some simplifications we have made in modelling the protocols. We then give a high-level description of our protocol model, introduce our formalisation of the *attestation* property and explain how the concept of a *rank function* can be used to prove that the protocols meet their attestation goal.

7.2 Simplifying assumptions

Devising the CSP model of the protocols has motivated us to make several simplifying assumptions to the protocols. For example, we assume that each mobile device has access to a single TPM and is running a single instance of the download application. We further assume that the attacker has access to a single TPM. Although this is a limitation on our intruder model it is not immediately clear whether allowing the intruder access to further trusted platforms affords him any greater powers. Further work is needed to confirm this.

Our model of the functionality of the trusted platform is both abstract and limited. For example, we model just the TPM operations that are required by the protocols. This results in a rather strong assumption that the intruder cannot not use any further TPM operations to gain an advantage. This assumption is strong because recent work has shown that security conditions can violated via legal, but unexpected, calls to a security API [30]. The assumption is reasonable, however, since our goal here is to reason about the correctness of the protocols in the presence of a 'perfect' trusted platform. This approach can be viewed as a first step to a deeper analysis which would look at the protocols in the context of a wider variety of platform behaviour. Arguably, it would be imprudent to attempt such an analysis by hand.

We assume that the integrity metric reported by a TPM is represented by an atomic message component, I, and that, on receipt of the attestation message, the software provider can make use of a publicly known function $validate(I)$ to determine whether or not I is indicative of a trusted platform in a safe state. Both protocols demand that the platform state remains unchanged between key generation and association of these keys with I, and platform attestation. In the public key protocol, key generation and association with I, and attestation is an atomic operation, whereas the secret key protocol makes use of the exclusive transport session to ensure that the state remains unchanged. It is possible to model this behaviour in CSP with an *interrupt* that fires whenever a change of state is detected, effectively abort-

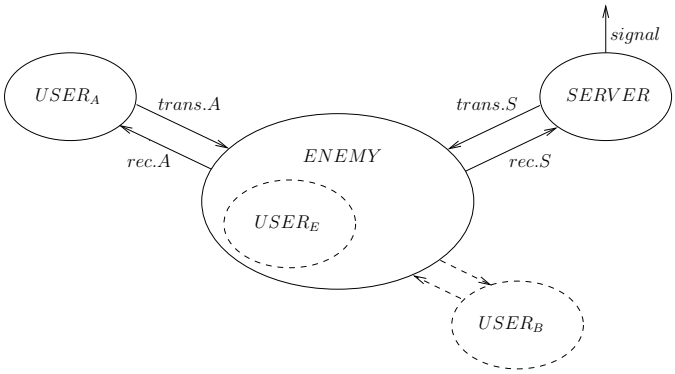

Figure 7: CSP network model

ing the protocol. However, this seems unnecessary. Instead, we model the platform by a recursive process that may engage in a run, and on completion of that run, engage in a further run. Each platform may therefore be engaged in at most one protocol run at any given time. The software provider may, of course, run arbitrarily many protocol instances concurrently.

7.3 Modeling the protocols

7.3.1 The attacker

The capabilities of the attacker are based on the Dolev-Yao model [31] in which the attacker is in complete control of (and, to all intents and purposes, replaces) the network. All messages sent by honest agents are intercepted by the attacker who may choose to send them on to their intended recipient, but may also decide to block the message, redirect it, or save it for later use. The network is composed of a number of honest agents (representing mobile devices) and a single software provider. Agents transmit messages on the communication channel *trans* and receive messages on the channel *rec*. The event $trans.i.j.m$ represents agent i's transmission of a message m intended for j. Similarly, $rec.j.i.m$ represents the reception of message m by j, apparently (but not necessarily) from i. This network arrangement is given in Figure 7.

The attacker's behaviour is bounded by the assumption of *perfect encryption*: he may only encrypt and decrypt messages if he possesses the appropriate key. He cannot, for example, decrypt arbitrary data. The attacker's store of knowledge is represented by a set F such that he knows a fact f precisely when $f \in F$. The attacker's deduction of new facts from facts he already knows is represented by the generates relation $\vdash$, such that $F \vdash m$ means

that the attacker can deduce the fact m if he knows all the facts in the set F. We extend the capabilities of the intruder based on the assumption that he has access to an uncompromised trusted platform. Crucially, this trusted platform will never be in a state that can convince a software provider to send A_C. We allow the attacker to run precisely the TPM commands of which the protocols makes use. Consider, for example, our model of the TPM_{Quote} operation:

$$\text{TPM\_Quote}\quad \frac{F \vdash H(b_M \parallel b_S \parallel id_S)}{F \vdash S_{TPM}(H(b_M \parallel b_S \parallel id_S) \parallel I)}(validate(I) = False)$$

This states that an attacker in possession of a message $H(b_M \parallel b_S \parallel id_S)$ can deduce the message $S_{TPM}(H(b_M \parallel b_S \parallel id_S) \parallel I)$ — the attestation message — where the side condition ensures that I is an integrity metric representing an unsafe platform .

7.4 Formalising the notion of attestation

The main security goal of the protocols is to ensure that an application A_C is only be retrievable from a software provider S by the agent A_D if A_D has previously proven its trustworthiness to S (in the case of the public key protocol, by using the nonce R_S). More formally we have:

> If the server S completes a run of the protocol, apparently with A_D, using data items R_S and A_C, then the TPM has previously attested its trustworthiness in a run of the protocol, taking the role of initiator, using the same data items R_S and A_C, and further each such run of S corresponds to a unique run of A_D.

Viewed in this way, attestation is a form of injective agreement [32] and we can model such claims using correspondence assertions. Briefly, if, on receipt of a valid attestation from A_D containing nonce R_S, S signals that he believes A_D to be trustworthy (by performing the signal event $signal.Trustworthy.A_D.R_S.A_C$) then we can check that the presence of this signal implies that a message was previously sent, by A_D, that included both R_S and a valid attestation.

For a download application A_D, this attestation consists of a message:

$$trans.A_D.S.S_{TPM}(H(A_D(public)) \parallel H(R_S \parallel Id_S) \parallel I\ )$$

with I such that $validate(I) = True$. We formalise the attestation goal as a predicate over the observable behaviour of the system:

$$\{trans.A_D.S.S_{TPM}(H(A_D(public))||H(R_S||Id_S)||I)\}$$
$$\textbf{precedes}$$
$$\{signal.Trustworthy.A_D.R_S.A_C\}$$

A CSP protocol model will satisfy this predicate if, whenever the event

$$signal.Trustworthy.A_D.R_S.A_C$$

is observed, the event

$$trans.A_D.S.S_{TPM}(H(A_D(public)) \;||\; H(R_S \;||\; Id_S) \;||\; I \;)$$

(where $validate(I) = True$) has previously been observed.

Moreover, if we block the process representing A_D from ever asserting its trustworthiness, then the event $signal.Trustworthy.A_D.R_S.A_C$ should never appear in the observable behaviour of the protocol model. If this condition is met, we will be able to conclude that the attestation goal holds.

7.5 Rank function

In order to prove that the **precedes** predicate holds, we use the concept of a *rank function*. The rank function is used to partition the message-space of a protocol such that all messages which should remain secret are assigned a rank of sec and all messages that might be public are assigned a rank of pub. The central rank theorem [28] gives a series of healthiness conditions that the function must satisfy in order for us to conclude that the protocol does actually maintain the secrecy of a given set of messages. Stated another way, the rank theorem gives the conditions under which it is reasonable to conclude that the sets of public and secret messages are disjoint; that there are no messages which the protocol makes public that we desire to keep secret. Given the two sets $\{trans.A_D.S.S_{TPM}(H(A_D(public)) \;||\; H(R_S \;||\; Id_S) \;||\; I \;)\}$ and $\{signal.Trustworthy.A_D.R_S.A_C\}$ the rank theorem tells us the following. If we can find a rank function ρ with the following properties:

1. The attacker process, $ENEMY$, starts by only knowing messages of rank pub.

2. If the attacker can perform the deduction $F \vdash m$ to learn the message m from the set of messages F then m has a rank of pub whenever the messages in F have a rank of pub.

3. $signal.Trustworthy.A_D.R_S.A_C$ has a rank of sec.

4. If each of the protocol participants is blocked from ever sending the message
   $S_{TPM}(H(A_D(public)) \parallel H(R_S \parallel Id_S) \parallel I\ )$ then they never give out a message of rank sec unless they have previously received a message of rank sec.

then we can conclude that the **precedes** predicate holds of the protocol model under consideration.

The formalisation and proof of this theorem can be found in [28]. Informally, conditions 1 and 2 tell us that an attacker cannot introduce a secret message onto the network unless someone else has already told him a secret. Condition 4 tells us that no users of the system can introduce a secret message either. Thus, 1, 2 and 4 together allow us to deduce that no message of rank sec can circulate in the system. Since, by condition 3, we know that $signal.Trustworthy.A_D.R_S.A_C$ has a rank of sec we can conclude that this message cannot therefore appear on the network, and that the attestation goal holds.

The rank theorem poses the question: *can we find such a rank function?* In fact, the answer to this question turns out to be *yes*. The full report on this verification [27] details these rank functions and demonstrates that they meet each of the four conditions of the central theorem.

7.6 Results

Each of the protocols reaches its attestation goal. However, it is worth considering whether this goal is achieved in a wholly transparent manner. Abadi and Needham have given several principles for protocol design [33] and we find that principle 10 is not satisfied by either protocol:

> If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

Consider the final protocol message:

$$S_S(E_{A_D(public)}(K1_{S,A_D}, K2_{S,A_D})) \parallel E_{K1_{S,A_D}}(MAC_{K2_{S,A_D}}(A_C))$$

This message is only implicitly bound to a particular protocol run. Once the software provider's signature is verified, the agent is confronted with the message component $E_{A_D(public)}(K1_{S,A_D}, K2_{S,A_D})$. It is feasible that this message may have been replayed by the attacker from an earlier protocol run, and it is only by using the decrypted MAC key to verify the MAC on A_C that the agent can discover whether this is the case. Such an assumption is reasonable in our model since we assume that the user can determine whether the decryption of a A_C and verification of the MAC on A_C succeeds. Additional mechanisms may be deployed such that replay of the final message can be detected earlier, as described in section 5.

Our analysis has shown that both of the protocol models meet the above security goal. However, it should be borne in mind exactly what has been achieved. Our protocol models are subject to the assumptions of perfect encryption (an attacker can only decrypt messages to which he knows the key), and other assumptions that place a bound on the behaviour of the protocol agents and the attacker. These assumptions are detailed in [27]. Nonetheless, the results show that the protocol models satisfy their goal when running on an *unbounded network* in which an arbitrary number of protocol instances may be running concurrently. Although a formal proof of correctness should not be mistaken as an unconditional proof, it is nonetheless sufficient to inspire confidence that the design of the protocols is sound.

8 Conclusion

This paper identifies a challenge presented by the desire to use mobile devices to receive broadcast content. Although existing standards may be applied to meet this challenge, these standards were not designed with the requirements of mobile devices in mind, and we suggest that a new approach is needed. With this aim, we identify emerging technology that enables new solutions to be developed and describe two potential solutions to the problem. The two resulting protocols defined in this paper allow a mobile receiver to describe its configuration to a software provider in a secure manner, and allow the software provider to securely deliver a conditional access application to the mobile receiver with confidence that the application will be protected on that receiver.

An general analysis of the security of the proposed protocols is presented followed by the results of a formal analysis that was carried out using CSP. The results of the analysis suggest that the protocols will provide a secure solution to the problem.

In developing the protocols we were motivated by a particular application,

the secure delivery of broadcast content to mobile devices. However, it is clear that the protocols described may be generalised and apply to a wider problem space. Using these protocols, any application may be securely downloaded to a mobile device with some degree of confidence that it will be protected form malicious attacks during transmission, storage, and execution.

Acknowledgment

The work reported in this paper has formed part of the Core 3 Research programme of the Virtual Centre of Excellence in Mobile and Personal Communications, Mobile VCE, www.mobilevce.com, whose funding support, including that of EPSRC, is gratefully acknowledged. Fully detailed technical reports on this research are available to Industrial Members of Mobile VCE.

References

[1] W. Tuttlebee, D. Babb, J. Irvine, G. Martinez, and K. Worrall, "Broadcasting and Mobile Telecommunications: Interworking — Not Convergence," *EBU Technical Review*, vol. 293, pp. 1–11, Jan. 2003.

[2] D. J. Cutts, "DVB Conditional Access," *IEE Electronics and Communications Engineering Journal*, vol. 9, no. 1, pp. 21–27, Feb. 1997.

[3] ETSI, "Digital Video Broadcasting (DVB); Head-End Implementation of DVB Simulcrypt," European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, ETSI Standard TS 103 197 V1.3.1, Jan. 2003.

[4] CENELEC, "Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications," European Committee for Electrotechnical Standardization (CENELEC), Brussels, Belgium, CENELEC Standard 50221, Feb. 1997.

[5] ETSI, "Digital Video Broadcasting (DVB); Support for use of Scrambling and Conditional Access (CA) within Digital Broadcasting Systems," European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, ETSI Technical Report ETR 289, Oct. 1996.

[6] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, ser. Discrete Mathematics and its Applications. Boca Raton, Fl: CRC Press, 1997, vol. 6. [Online]. Available: http://www.cacr.math.uwaterloo.ca/hac/

[7] TCG, "TCG Specification Architecture Overview," The Trusted Computing Group, Portland, OR, USA, TCG Specification Revision 1.2, Apr. 2003. [Online]. Available: https://www.trustedcomputinggroup.org/downloads/specifications/TCG_1_0_Architecture_Overview.pdf

[8] ——, "TPM Main, Part 1 Design Principles," The Trusted Computing Group, Portland, OR, USA, TCG Specification Version 1.2 Revision 62, Oct. 2003. [Online]. Available: https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part1_Design_Principles.pdf

[9] ——, "TPM Main, Part 2 TPM Data Structures," The Trusted Computing Group, Portland, OR, USA, TCG Specification Version 1.2 Revision 62, Oct. 2003. [Online]. Available: https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part2_TPM_Structures.pdf

[10] ——, "TPM Main, Part 3 Commands," The Trusted Computing Group, Portland, OR, USA, TCG Specification Version 1.2 Revision 62, Oct. 2003. [Online]. Available: https://www.trustedcomputinggroup.org/downloads/tpmwg-mainrev62_Part3_Commands.pdf

[11] ——, "TCG Software Stack (TSS) Specification," The Trusted Computing Group, Portland, OR, USA, TCG Specification Version 1.1, Aug. 2003. [Online]. Available: https://www.trustedcomputinggroup.org/downloads/TSS_Version__1.1.pdf

[12] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler, *Trusted Computing Platforms: TCPA Technology in Context*, S. Pearson, Ed. Prentice Hall, 2003.

[13] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A Trusted Open Platform," *IEEE Computer*, vol. 36, no. 7, pp. 55–62, July 2003.

[14] M. Peinado, Y. Chen, P. England, and J. Manferdelli, "NGSCB: A Trusted Open System," in *Proceedings of 9th Astralasian Conference on Information Security and Privacy, ACISP 2004*, ser. Lecture Notes in Computer Science, H. Wang, J. Pieprzyk, and V. Varadharajan, Eds., vol. 3108. Berlin, Germany: Springer-Verlag, July 2004, pp. 86–97.

[15] Microsoft, "Security Model for the Next-Generation Secure Computing Base," Microsoft Corporation," Windows Platform Design Notes, 2003. [Online]. Available: http://www.microsoft.com/resources/ngscb/documents/ngscb_security_model.doc

[16] ——, "Hardware Platform for the Next-Generation Secure Computing Base," Microsoft Corporation," Windows Platform Design Notes, 2003. [Online]. Available: http://www.microsoft.com/resources/ngscb/documents/NGSCBhardware.doc

[17] ——, "NGSCB: Trusted Computing Base and Software Authentication," Microsoft Corporation," Windows Platform Design Notes, 2003. [Online]. Available: http://www.microsoft.com/resources/ngscb/documents/ngscb_tcb.doc

[18] C. Mitchell, Ed., *Trusted Computing*, 1st ed. United Kingdom: IEE Press, 2005, to appear.

[19] Intel, "LaGrande Technology Architectural Overview," Intel Corporation, Tech. Rep. 252491-001, Sept. 2003. [Online]. Available: http://www.intel.com/technology/security/downloads/LT_Arch_Overview.pdf

[20] B. Pfitzmann, J. Riordan, C. Stuble, M. Waidner, and A. Weber, "The PERSEUS System Architecture," IBM, IBM Research Division, Zurich Laboratory, Tech. Rep. RZ 3335 (#93381), Apr. 2001.

[21] A.-R. Sadeghi and C. Stuble, "Taming "Trusted Platforms" by Operating System Design," in *Proceedings of Information Security Applications, 4th International Workshop, (WISA 2003)*, ser. Lecture Notes in Computer Science, K. Chae and M. Yung, Eds., vol. 2908. Berlin, Germany: Springer-Verlag, Aug. 2003.

[22] J. Marchesini, S. Smith, O. Wild, and R. MacDonald, "Experimenting with TCPA/TCG Hardware," Dartmouth PKI Lab, Dartmouth College, Hanover, New Hampshire, USA, Tech. Rep. 2003-476, Dec. 2003.

[23] G. E. Suh, D. Clarke, B. Gassend, M. van Dyke, and S. Devadas, "The AEGIS Processor Architecture for Tamper–Evident and Tamper-Resistant Processing," in *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS'03)*. San Francisco: ACM Press, June 2003, pp. 160 – 171. [Online]. Available: http://www.csail.mit.edu/research/abstracts/abstracts03/architecture/51suh.pdf

[24] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*. ACM Press, Nov. 2000, pp. 169–177.

[25] ISO/IEC, "Information Technology - Security Techniques - Entity Authentication - Part 3: Mechanisms using Digital Signature Techniques," International Organisation for Standardisation, Geneva, Switzerland, ISO/IEC Standard 9798-3, 1998.

[26] A. W. Dent and C. J. Mitchell, *User's Guide to Cryptography and Standards*. Artech House, 2005.

[27] R. Delicata, "An Analysis of Two Protocols for Conditional Access in Mobile Systems," Department of Computing, University of Surrey, UK, Tech. Rep. CS-04-13, 2004.

[28] S. A. Schneider, "Verifying Authentication Protocols with CSP," in *Proceedings of the 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997, pp. 3–17. [Online]. Available: http://www.citeseer.nj.nec.com/196196.html

[29] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985.

[30] M. Bond, "Understanding security APIs," Ph.D. dissertation, University of Cambridge Computer Laboratory, June 2004.

[31] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[32] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997. [Online]. Available: citeseer.nj.nec.com/lowe96hierarchy.html

[33] M. Abadi and R. M. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 6–15, 1996. [Online]. Available: http://www.citeseer.nj.nec.com/abadi96prudent.html