Building general purpose security services on trusted computing

Chunhua Chen[1*] Chris J. Mitchell[2] Shaohua Tang[3†]
[1,3] School of Computer Science and Engineering
South China University of Technology
Guangzhou 510641, China
[1] chen.chunhua@mail.scut.edu.cn, [3] csshtang@scut.edu.cn
[2] Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
c.mitchell@rhul.ac.uk

October 31, 2011

Abstract

The Generic Authentication Architecture (GAA) is a standardised extension to the mobile telephony security infrastructures (including the Universal Mobile Telecommunications System (UMTS) authentication infrastructure) that supports the provision of generic security services to network applications. In this paper we propose one possible means for extending the widespread Trusted Computing security infrastructure using a GAA-like framework. This enables an existing security infrastructure to be used as the basis of a general-purpose authenticated key establishment service in a simple and uniform way, and also provides an opportunity for trusted computing aware third parties to provide novel security services. We also discuss trust issues and possible applications of GAA services.

Keywords: GAA, Trusted Computing, security service

1 Introduction

Almost any large scale network security system requires the establishment of some kind of a security infrastructure. For example, if network authentica-

[*]The author is a PhD student at the South China University of Technology. This work was performed during a visit to the Information Security Group at Royal Holloway, University of London, sponsored by the Chinese Scholarship Council and the Natural Science Foundation of Guangdong Province, China (No. 9351064101000003).

[†]This author is the corresponding author.

tion or authenticated key establishment is required, then the communicating parties typically need access to a shared secret key or certificates for each other's public keys.

Setting up a new security infrastructure for a significant number of clients is by no means a trivial task. For example, establishing a public key infrastructure (PKI) for a large number of users involves setting up a secure certification authority (CA), getting every user to securely generate a key pair, securely registering every user and corresponding public key, and securely generating and distributing public key certificates. In addition, the ongoing management overhead is non-trivial, covering issues such as revocation and key update.

At the same time, there are a number of existing security infrastructures, in some cases with almost ubiquitous coverage. When deploying a new network security protocol it is therefore tempting to try to exploit one of these existing security infrastructures to avoid the need for the potentially costly roll-out of a new infrastructure.

This is by no means a new idea (see, for example, [9]). However, previous proposals have been application-specific. We instead propose the use of a general framework which enables almost any pre-existing infrastructure to be used as the basis for the provision of generic security services.

Of particular (and motivating) importance to our work is the Generic Authentication Architecture (GAA) [1]. This architecture has been designed to enable the Universal Mobile Telecommunications System (UMTS) authentication infrastructure to be exploited for the provision of security services. Building on previous work [4], we propose the adoption of the architecture used by UMTS GAA to enable a wide range of other pre-existing infrastructures to be similarly exploited. One security infrastructure of particular interest is the emerging Trusted Computing (TC) infrastructure, including the Trusted Platform Modules (TPMs) present in a significant proportion of all new Personal Computers (PCs).

We first generalise the concepts and procedures of GAA. We then consider how this generalised notion can be supported by the trusted computing security infrastructure. We refer to this combination as TC GAA. We also discuss related trust issues and consider possible applications of GAA services.

The remainder of this paper is organised as follows. In section 2 we introduce our generalised version of GAA, and also briefly describe the standardised version building on the UMTS authentication infrastructure. In section 3 we give details of TC GAA, building on a general Trusted Computing security infrastructure. This is followed by a description of an instantiation of TC GAA using TPMs in section 4. In section 5 we provide an informal security analysis. We discuss related trust issues and possible applications of GAA services in section 6. In section 7 we draw conclusions.

2 Generic Authentication Architecture

We start by describing our generalised version of the GAA architecture, introducing the main roles in the framework, the goals and rationales, and the two main procedures. This generalised GAA architecture was first described in [4]. We follow this by briefly describing the standardised implementations of GAA as supported by the UMTS authentication infrastructure.

2.1 Overview of GAA

As shown in Figure 1, the following entities play a role in the GAA architecture.

- The *Bootstrapping Server Function* (BSF) *server* B acts as a Trusted Third Party (TTP), and is assumed to have the means to access credentials belonging to a pre-existing security infrastructure. B uses the pre-established credentials to provide authenticated key establishment services to *GAA-enabled user platforms* and *GAA-aware application servers*.

- A *GAA-aware application server* S is assumed to have the means to establish a mutually authenticated and confidential secure channel with B, and an arrangement to access the security services provided by B. The means by which the secure channel between B and S is established is outside the scope of the GAA framework. In the GAA context, the functionality of a *GAA-aware application server* is also referred to as the *Network Application Function* (NAF) *server*. We use the terms application server and NAF server interchangeably throughout.

- A *GAA-enabled user platform* P is assumed to be equipped with credentials belonging to the pre-existing security infrastructure. P possesses a BSF client C_B, which uses the platform credentials to interact with B to provide authenticated key establishment services. P also possesses a NAF client C_S that accesses services provided by S. C_S interacts with C_B to obtain the cryptographic keys necessary to provide client-server security services.

The user platform and the BSF server need to interact with the pre-existing security infrastructure, whereas the application server does not (it only needs to interact with the BSF server and the user platform). Also, the user platform and the application server do not need to have a pre-existing security relationship.

GAA provides a general purpose key establishment service for user platforms and application servers. As described below, GAA uses a two-level

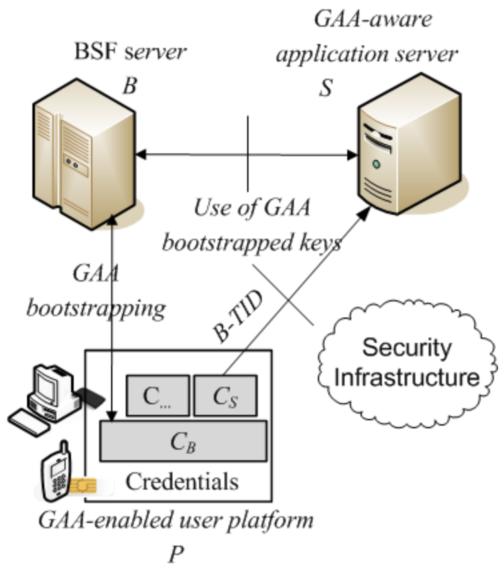

Figure 1: GAA framework

key hierarchy, consisting of a master session key and server- and application-specific session keys. The master session key is established using the pre-existing security infrastructure, and is not used directly to secure GAA-based applications. Instead it is used to generate the server/application-specific session keys using a *key diversification* function. By choosing a function with appropriate properties, it can be arranged that knowledge of a server/application specific session key will not reveal any information about the master session key or any other server/application-specific keys.

2.2 GAA procedures

As we now describe, GAA incorporates two main procedures: *GAA bootstrapping* and *Use of bootstrapped keys*.

GAA bootstrapping uses the pre-existing security infrastructure to set up a shared master key *MK* between *P* and *B*. Also established is a Bootstrapping Transaction Identifier *B-TID* for *MK* and the lifetime of this key. *B-TID* must consist of a (statistically) unique value which can identify an instance of *GAA bootstrapping* as well as *B*'s network domain name.

The *Use of bootstrapped keys* procedure establishes a server/application-specific session key *SK* between *P* and *S*, using the master key *MK* shared by *P* and *B*. The procedure operates in the following way. *P* first derives a session key *SK* as:

$$SK = \text{KDF}(MK, NAF\text{-}Id, \text{other values})$$

where KDF is a one-way *key diversification* function, and *NAF-Id* is an application-specific value consisting of the Fully Qualified Domain Name (FQDN) of S and the identifier of the underlying application protocol. Other values may be included in the key derivation computation depending on the nature of the underlying security infrastructure. P (strictly, C_S) then starts the application protocol by sending a request containing *B-TID* to S. S submits the received *B-TID* and its own identifier *NAF-Id* to B to request the session key *SK*. Note that *B-TID* contains B's network domain name, so S knows where to send the request. As stated above, we require that S and B have the means to establish a mutually authenticated and confidential secure channel, and hence B can verify S against its FQDN. If S is authorised, B derives *SK* from the *MK* identified by *B-TID*, and sends *SK*, its lifetime, and other relevant information to S via the secure channel. P and S now share *SK*, which they can use to secure application-specific messages.

Note that *key separation* is enforced by including *NAF-Id* as an input to the *key diversification* function. Other values used in the computation of *SK* could include identifiers for the GAA bootstrapping instance and the user platform.

2.3 UMTS GAA

The standardised versions of GAA [1] build on the mobile authentication infrastructures (including those for UMTS and GSM). In the UMTS version of GAA, a UMTS network operator provides the BSF with the key *MK*, and the user platform is a UMTS mobile. The UMTS authentication and key agreement protocol is used to establish the key *MK*, which is independently generated by the user platform and the network operator as part of *GAA bootstrapping*. The identifier *B-TID* is a combination of the *RAND* used in UMTS authentication and the BSF's identifier.

3 TC GAA

In this section we propose a possible means of using the Trusted Computing security infrastructure to support a GAA-like framework, which we refer to as TC GAA. We start by giving a high-level description of the Trusted Computing security infrastructure, without referring to any specific trusted computing technology. We then specify the operation of TC GAA as built on this general infrastructure. A specific instantiation of TC GAA using the features of a TCG-compliant TPM is described in the next section.

Note that a very brief sketch of a possible TC GAA implementation has previously been described [4]. By contrast, in this paper we give detailed descriptions of instantiations of TC GAA, and provide an analysis of its security properties.

3.1 Trusted Computing Security Infrastructure

A fundamental notion in Trusted Computing (TC) is the Trusted Platform (TP). According to Balacheff et al. [2]: “A trusted platform (TP) is defined as a computing platform that has a trusted component, which is used to create a foundation of trust for software processes”. We refer to such a trusted component as a Trusted Module (TM). A TM encompasses all the platform functionalities and data areas within a TP that must be trusted, if the platform is to be trusted. Gallery [5] identifies a minimum set of trusted TM features. In practice, a trusted computing technology might make use of a range of mechanisms to meet these requirements.

Listed below (following Gallery [5]) are the features that a TM must possess in order to support our general instantiation of TC GAA.

- The TM is a self-contained processing module containing specialist capabilities, including random number generation, asymmetric key generation, digital signing, encryption/decryption and hashing.
- The TM contains shielded locations, data stored in which (e.g. TM-generated keys) is protected against interference or snooping and is only accessible to the specified capabilities.
- The TM is equipped with a unique asymmetric encryption key pair at the time of (or soon after) manufacture. The private decryption key is stored securely in the TM-shielded location and is never exported from the TM. A certificate for the associated public key, containing a general description of the TM and its security properties, is generated by a CA.
- The TM is capable of generating asymmetric signature key pairs. The TM can, by some means, obtain certificates for the public keys of such key pairs from a CA. The private signature keys are securely held by the TM.
- The TM is capable of generating asymmetric encryption key pairs. The TM can generate certificates for the public keys of these key pairs using the signature keys described above. The private decryption keys are securely held by the TM.

Note that information said to be held securely by the TM may actually be stored externally to the TM, encrypted using a key known only to the TM.

A TM will typically possess a range of other security-related features, not directly used by TC GAA. Some of these features could be used to enhance the trustworthiness of the TC GAA application software running on the TP. In particular, platform integrity measurement, storage and reporting services could be used to provide assurance regarding the software state of the platform.

Trusted Computing makes use of public key cryptography, and realising its full potential requires a supporting PKI. We use the term Trusted Computing security infrastructure to refer to the set of deployed TMs, the associated keys, and the supporting PKIs. Trusted computing technology can be implemented in a variety of computing platforms, including PCs (e.g. laptops) and mobile devices (e.g. mobile phones). In this paper we focus on PC-based TPs.

3.2 The TC GAA Architecture

As shown in Figure 2, the following Trusted Computing specific entities play a role in TC GAA.

- The supporting PKIs. We assume that all relevant certificates are obtainable by the entities involved.
- The *GAA-enabled user platform P* is a Trusted Platform containing a Trusted Module M, as defined in section 3.1. We assume that M has already generated a signature key pair, and has obtained a certificate Cert_M for the public key of this key pair from a CA, where Cert_M binds an identity of M (Id_M) to the public key (where M may have many such identities). The private signing key is available only to M (we assume it is stored externally to M, encrypted using a key known only to M). The BSF client, C_B, implements the authentication and key establishment protocol which forms part of the *TC GAA bootstrapping* procedure specified below.
- The BSF server B has a signature key pair and a certificate Cert_B for the public key of this key pair. This key pair is used for entity authentication.

In practice, M might be equipped with multiple certified signature key pairs. We assume that the certified signature key pair specified above is used for *TC GAA bootstrapping*, and is used for multiple instances of the protocol. Typically this involves M, in conjunction with C_B, obtaining such a key pair via a separate configuration procedure prior to the *TC GAA bootstrapping* procedure. Thus C_B knows which signature key pair is to be used in *TC GAA bootstrapping.*

3.3 The TC GAA procedures

In this section we specify the *TC GAA bootstrapping* and the *TC GAA Use of bootstrapped keys* procedures, which use the general Trusted Computing security infrastructure defined in section 3.1. The authentication and

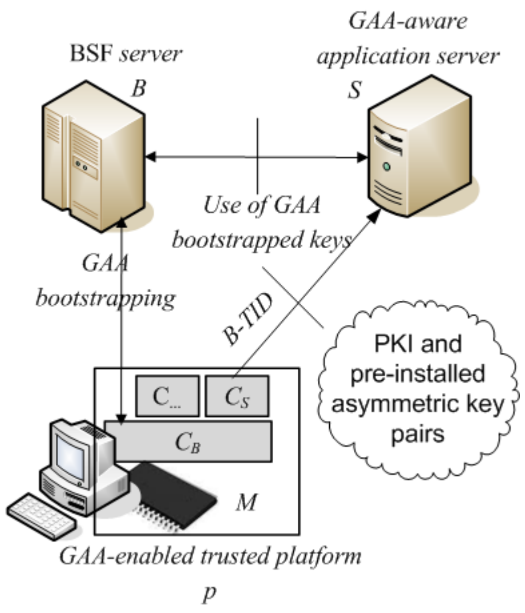

Figure 2: TC GAA framework

key establishment protocol which forms part of *TC GAA bootstrapping* is motivated by the protocol defined in Gallery and Tomlinson [6]. Table 1 summarises the notation used in the remainder of this paper.

P	a GAA-enabled trusted platform
M	a trusted module embedded in P
I	integrity metrics that reflect a certain state of P
B	a BSF server
C_B	a BSF client residing in P
S	a GAA-aware application server
CA	a Certification Authority trusted by all entities
Cert_X	a certificate for entity X's signature public key
Id_X	an identity of entity X
R_X	a random number issued by entity X
M_{pub}	a TM-generated temporary public encryption key
M_{pri}	a TM-generated temporary private decryption key corresponding to M_{pub}
$E_{M_{pub}}(Z)$	the result of the asymmetric encryption of data Z using the public key M_{pub}
H	a one-way hash function
$S_X(\mathrm{Z})$	the digital signature of data Z computed using entity X's private signature transformation
$X\|\|Y$	the concatenation of data items X and Y in that order

Table 1: Notation

The *TC GAA bootstrapping* protocol involves the following sequence of steps, where $X \to Y : Z$ is used to indicate that message Z is sent by entity X to entity Y.

1. $C_B \to B$: request to bootstrap a master session key MK.

2. B: generates and caches a random value R_B.

3. $B \to C_B$: R_B.

4. $C_B \to M$: request to generate a random number.

5. $M \to C_B$: R_M.

6. $C_B \to M$: request to load M's private signature key.
   (Note that M's private signing key must be loaded into M before use because it is stored externally to M.)

7. M: loads M's private signing key.

8. $M \to C_B$: the handle of the loaded private signing key.

9. $C_B \rightarrow M$: request generation of an asymmetric encryption key pair (M_{pub}, M_{pri}), and association of M_{pri} with a specified protected environment state of P.

10. M: generates (M_{pub}, M_{pri}), where M_{pri} is bound to the specified protected environment state.

11. $M \rightarrow C_B$: (M_{pub}, M_{pri}), where M_{pri} is encrypted using a key available only to M.

12. $C_B \rightarrow M$: request to generate a certificate for M_{pub} in association with R_M, R_B, Id_B and I.
    (Id$_B$ is B's network domain name. The integrity metrics I reflect both the state of the protected environment when the key pair (M_{pub}, M_{pri}) was generated and the state required for use of the newly generated M_{pri}.)

13. M: signs a data string including M_{pub}, R_M, R_B, Id_B and I using its private signing key to obtain: $S_M(R_M||R_B||\text{Id}_B||M_{pub}||I)$.

14. $M \rightarrow C_B$: $M_{pub}||I||S_M(R_M||R_B||\text{Id}_B||M_{pub}||I)$.

15. $C_B \rightarrow B$: $\text{Cert}_M||\text{Id}_M||R_M||R_B||\text{Id}_B||M_{pub}||I||S_M(R_M||R_B||\text{Id}_B||M_{pub}||I)$.

16. B: retrieves Cert_M and verifies it.
    B: verifies $S_M(R_M||R_B||\text{Id}_B||M_{pub}||I)$.
    B: verifies R_B to ensure that the message is fresh.
    B: verifies Id_B to ensure that the message is intended for it.
    B: verifies that I indicates that C_B is executing as expected, i.e. that it has not been tampered with.

17. Assuming that the signature from M verifies correctly, the value of R_B is fresh, the value of Id_B is as expected, and the integrity metrics I are acceptable, then
    B: generates a master session key *MK*, sets the lifetime of *MK* according to B's local policies, and generates an identifier *B-TID* for *MK* consisting of R_M, R_B and B's network domain name.

18. B: caches *B-TID*, *MK*, lifetime of *MK*, R_M, R_B, and Id_M.

19. $B \rightarrow C_B$: $\text{Cert}_B||B\text{-}TID||\text{lifetime of } MK||R_B||R_M||\text{Id}_M||E_{M_{pub}}(MK)||$ $S_B(R_B||R_M||\text{Id}_M||E_{M_{pub}}(MK))$.

20. C_B: retrieves Cert_B and verifies it.
    C_B: verifies $S_B(R_B||R_M||\text{Id}_M||E_{M_{pub}}(MK))$
    C_B: verifies R_M to ensure that the message is fresh.
    C_B: verifies Id_M to ensure that the message is intended for it.

21. Assuming that the signature from B verifies correctly, the value of R_M is fresh, and the value of Id_M is as expected, then:
$C_B \to M$: request to load the encrypted key M_{pri}.

22. M: loads the encrypted key M_{pri}.

23. $M \to C_B$: the handle of the loaded M_{pri}.

24. $C_B \to M$: request to decrypt $E_{M_{pub}}(MK)$ using M_{pri}.

25. M: decrypts $E_{M_{pub}}(MK)$ and deletes M_{pri}.

26. $M \to C_B$: *MK*.

27. C_B: caches *B-TID*, *MK*, lifetime of *MK*, R_M, R_B, and Id_M.

28. C_B: deletes the part-encrypted key pair (M_{pub}, M_{pri}).

After successful execution of the above protocol, B and C_B share a new set of bootstrapped credentials, including random challenges R_M and R_B, M's identity Id_M, and a master session key *MK* together with its identifier *B-TID* and lifetime. We assume that these bootstrapped credentials are held securely by C_B by some means (e.g. encrypted and integrity protected by M).

Verifying the trustworthiness of P's software environment is not necessary in order to complete authenticated key establishment, which is, of course, the main goal of the *TC GAA bootstrapping* protocol. If B does not need to verify the trustworthiness of P's software environment at the time of protocol execution, a fresh encryption key pair (M_{pub}, M_{pri}) does not need to be generated for every instance of the bootstrap procedure. Instead M could generate a encryption key pair (without associating it with a specified protected environment state) in advance of the protocol, and use it multiple times. When bootstrapping, M would load the public key of this encryption key pair and use its private signing key to generate a certificate for this public key that includes the nonces for the current session (i.e. R_M and R_B).

In the *TC GAA use of bootstrapped keys* procedure, C_S and S follow the procedure defined in section 2.2 to establish a server/application-specific session key *SK*. The session key *SK* is derived as follows:

$$SK = \mathrm{KDF}(MK, R_M, R_B, Id_M, NAF\text{-}Id).$$

4 Building TC GAA using the TCG Specifications

The generic version of TC GAA described above could be implemented using a range of technologies, including a platform constructed in accordance with the specifications of the Trusted Computing Group (TCG). In this section we specify an instantiation using TPMs as defined in the version 2.1 of the Trusted Computing Group (TCG) specifications [11, 12, 13].

4.1 The TCG Specifications

A TCG-compliant TPM meets the requirements for the TM identified in section 3.1. Gallery [5] describes the TPM features. In this section we map the necessary features for a TM identified section 3.1 onto a TPM.

- The TPM is a secure module which contains protected capabilities and shielded locations. The protected capabilities include all the functionalities required for TC GAA, as well as other capabilities such as a SHA-1 engine, a HMAC engine, and a monotonic counter. When implemented as a hardware chip, the TPM must be inextricably bound to its host platform.

- The TPM is equipped with a unique Endorsement Key (EK) pair, an RSA encryption key pair, at the time of (or soon after) manufacture. The private decryption key is stored in a TPM-shielded location and is never exported from the TPM. An endorsement credential (a certificate for the public key of this EK key pair) is signed by a CA (as provided by a Trusted Platform Module Entity (TPME)). The endorsement credential, in conjunction with its associated conformance credential and platform credential, describes the security properties of the TPM and its host platform.

- The TPM is capable of generating Attestation Identity Keys (AIKs), which are RSA signature key pairs. A certificate Cert_{TPM} for the public key of an AIK key pair can be obtained in two ways: using a privacy CA, and using Direct Anonymous Attestation [3]. The associated private key is securely held by the TPM. The TPM can use an AIK to certify other TPM-generated keys.

- The TPM is capable of generating asymmetric encryption key pairs on demand, which can be migratable or non-migratable. For the purposes of TC GAA, we assume that non-migratable keys are used. The private key of a TPM-generated encryption key pair is securely held by the TPM. A certificate for the encryption public key can be generated by the TPM using an AIK.

- Integrity measurement, storage and reporting are supported. Measuring events on a platform is a two-stage process that begins with appending a hash of the event (e.g. the launch of an application) being measured to the content of one of a number of internal registers (known as Platform Configuration Registers (PCRs)). The hash of the resulting string is written back to the PCR concerned. The other part of the process involves recording details of the event in the Stored Measurement Log (SML) file. The values of the PCRs identify the current platform state. When a challenger wishes to verify a trusted platform's integrity, it requests (a portion) of the platform's SML, together with a TPM-generated signature (generated using an AIK) on a subset of PCR values that describe the desired portion of the platform's operating state.

4.2 TC GAA Procedures Using a TPM

A trusted platform which contains a TCG-compliant TPM M can play the role of the *GAA-enabled user platform* P. M must possess an AIK pair, which plays the role of the signature key pair used in the *TC GAA bootstrapping* protocol. A certificate Cert_M is required to bind an identity of M (Id_M) to the public key of the AIK. We suppose that the private signing key is stored externally to M, encrypted using a key available only to M.

We now describe a means of using the version 1.2 TCG TPM data structures [12] and command set [13] to implement the *TC GAA bootstrapping* protocol defined in section 3.3. The data structures involved include TPM_NONCE, TPM_KEY_HANDLE, TPM_KEY and TPM_CERTIFY_INFO. The TPM commands involved include TPM-CreateWrapKey, TPM_GetRandom, TPM_LoadKey, TPM-CertifyKey and TPM-UnBind.

During protocol execution, C_B calls the TPM_GetRandom command to request M to generate a random value R_M (step 4). M returns R_M in a TPM_NONCE data structure (step 5). C_B then calls the TPM_LoadKey command, requesting M to load a private signing key (step 6). M returns the handle of the loaded key in a TPM_KEY_HANDLE data structure (step 8). C_B next invokes the TPM-CreateWrapKey command, requesting M to generate an encryption key pair (M_{pub}, M_{pri}) (step 9). The TPM-CreateWrapKey command arguments include an unwrapped TPM_KEY data structure and a parent wrapping key. The unwrapped TPM_KEY specifies information about the key pair to be created, such as the key size (e.g. 1024 bits), the key usage (i.e. TPM_KEY_BIND), and the key flag (i.e. non-migratable); it also specifies the platform state at the time the key pair is created (referred to as digestAtCreation) and the platform state required for use of the generated private key (referred to as digestAtRelease).

M returns a wrapped TPM_KEY data structure (step 11). The wrapped TPM_KEY contains M_{pub}, the encrypted M_{pri} (encrypted using the parent wrapping key), a value indicating that the key pair is non-migratable, and a value indicating that the key pair can only be used for TPM-Bind and TPM-UnBind operations. The wrapped TPM_KEY also identifies the PCRs whose values are bound to M_{pri}, the PCR digests at the time of key pair creation, and the PCR digests required for M_{pri} use. The PCR data included in the wrapped TPM_KEY maps to the integrity metrics I in the generic protocol.

M is then requested to sign M_{pub} and I in conjunction with external data R_M, R_B and Id_B (step 12). This involves a call to the TPM-CertifyKey command, which takes arguments that include the public key of the TPM-generated key pair to be certified (i.e. a wrapped TPM_KEY) and a private signature key (i.e. M's private signing key). A hash of R_M, R_B and Id_B is also input as 160 bits of externally supplied data. In response, M returns a TPM_CERTIFY_INFO data structure and a signature on TPM_CERTIFY_INFO (step 14). The string TPM_CERTIFY_INFO contains (a description of) the public key that has been certified, the 160 bits of externally supplied data, a hash of the certified public key, and the PCR data in use.

B needs to encrypt MK so that it can be decrypted by M (step 19). B calls the Tspi_Data_Bind command ([10], p. 363), which takes a data block to be encrypted (i.e. MK) and a public encryption key (i.e. M_{pub}) as arguments and returns an encrypted MK (i.e. $E_{M_{pub}}(MK)$).

Assuming that the response from B is correct, C_B requests M to load the encrypted key M_{pri} (step 21), and then calls the TPM-UnBind command to decrypt $E_{M_{pub}}(MK)$ (step 24). M outputs the master key MK to C_B (step 26).

We now summarise the *TC GAA bootstrapping* protocol using the version 1.2 TCG TPM commands and data structures.

1. $C_B \rightarrow B$: request to bootstrap a master session key MK.
2. B: generates and caches a random value R_B.
3. $B \rightarrow C_B$: R_B.
4. $C_B \rightarrow M$: TPM_GetRandom.
5. $M \rightarrow C_B$: TPM_NONCE (containing R_M).
6. $C_B \rightarrow M$: TPM_LoadKey (M's private signing key).
7. M: loads M's private signing key.
8. $M \rightarrow C_B$: TPM_HANDLE (containing the handle of M's private signing key).
9. $C_B \rightarrow M$: TPM-CreateWrapKey (an unwrapped TPM_KEY, the handle of the loaded parent wrapping key).
10. M: generates (M_{pub}, M_{pri}), where M_{pri} is bound to a specified protected environment state.
11. $M \rightarrow C_B$: a wrapped TPM_KEY.

12. $C_B \rightarrow M$: TPM-CertifyKey (the wrapped TPM_KEY, $H(R_M||R_B||\text{Id}_B)$). (Note that the PCR data included in TPM_KEY maps to the integrity metrics I).

13. M: generates TPM_CERTIFY_INFO data structure, and signs it.

14. $M \rightarrow C_B$: TPM_CERTIFY_INFO$||S_M(H(R_M||R_B||\text{Id}_B)||H(M_{pub})||I)$. (We represent the signature on TPM_CERTIFY_INFO generated by M in simplified form as $S_M(H(R_M||R_B||\text{Id}_B)||H(M_{pub})||I)$.)

15. $C_B \rightarrow B$: Cert$_M$||Id$_M$||R_M||R_B||Id$_B$||TPM_Key||SMLData TPM_Certify_Info|| $S_M(H(R_M||R_B||\text{Id}_B)||H(M_{pub})||I)$.

16. B: verifies Cert$_M$, the received signature, R_B, Id$_B$ and I, as described in section 3.3.
    (B uses the SML data received in step 15 to recompute I for verification. If B does not want to verify the trustworthiness of P's software environment, the SML data does not need to be sent.)

17. Assuming that the signature from M verifies correctly, the values of R_B and Id$_B$ are as expected, and the integrity metrics I are acceptable, then:
    B: Generates a symmetric session key *MK*, sets the lifetime of *MK* according to B's local policies, and generates an identifier *B-TID* for *MK* consisting of R_{TPM}, R_{BSF} and B's network domain name.

18. B: caches *B-TID*, *MK*, lifetime of *MK*, R_M, R_B and Id$_M$.

19. $B \rightarrow C_B$: Cert$_B$||*B-TID*||lifetime of *MK*$||R_B||R_M||$Id$_M||E_{M_{pub}}(MK)||$ $S_B(R_B||R_M||\text{Id}_M||E_{M_{pub}}(MK))$.

20. C_B: verifies Cert$_B$, the received signature, R_M and Id$_M$, as described in section 3.3.

21. Assuming that the signature from B verifies correctly, the value of R_M is fresh, and the value of Id$_M$ is as expected, then:
    $C_B \rightarrow M$: TPM_LoadKey (the encrypted key M_{pri}).

22. M: loads the encrypted key M_{pri}.

23. $M \rightarrow C_B$: KEY_HANDLE (containing the handle of M_{pri}).

24. $C_B \rightarrow M$: TPM-UnBind ($E_{M_{pub}}(MK)$, the handle of the loaded key M_{pri}).

25. M: decrypts $E_{M_{pub}}(MK)$ and deletes M_{pri}.

26. $M \rightarrow C_B$: *MK*.

27. C_B: caches *B-TID*, *MK*, lifetime of *MK*, R_M, R_B and Id_m.

28. C_B: deletes the part-encrypted key pair (M_{pub}, M_{pri}).

5 Informal Security Analysis

We now provide an informal security analysis of the authentication and key establishment protocol used by the *TC GAA bootstrapping* protocol in section 3.3 (including steps 3, 15 and 19). We consider a threat model in which an attacker $\mathcal{A}$ is able to observe and make arbitrary modifications to messages exchanged between B and P, including replaying and blocking messages as well as inserting completely spurious messages. This allows a trivial denial of service attack which cannot be prevented. Note that $\mathcal{A}$ is not allowed to compromise the implementations of B and P; such attacks on system integrity cannot be prevent by the key establishment process, and are thus not addressed by the schemes we propose.

1. *Entity authentication.* The protocol provides mutual authentication between B and M using digital signature techniques. B can verify the identity of the M (Id_M); that is, the signature of M on R_B and Id_B allows B to authenticate M (step 15). Similarly, M can authenticate B by verifying the signature of B on R_M and Id_M (step 19). Step 3, 15 and 19 of the protocol conform to the three pass unilateral authentication protocol mechanism described in clause 5.2.2 of ISO/IEC 9798-3:1998 [8], in which the values R_B and R_M, generated by B and M respectively, serve as the nonces.

2. *Confidentiality of the master session key MK.* The signature of M on M_{pub} allows B to verify that M generated the key pair (M_{pub}, M_{pri}) (step 15). *MK* is generated by B, and is encrypted using the TM-generated temporary public key M_{pub} before being sent to M (step 19). The corresponding private key M_{pri} is securely held by M, and is only useable when the protected platform is in a particular trusted state. Hence, $\mathcal{A}$ cannot access to *MK* under the assumed threat model.

3. *Origin authentication.* R_B, R_M, Id_B, Id_M, M_{pub} and $E_{M_{pub}}(MK)$ are signed by B and M (steps 15 and 19), and thus both parties can verify the origin of the received message. The signatures also provide integrity protection.

4. *Freshness.* R_B, generated by B, is included in the signed bundle sent to B in step 15; similarly R_M, generated by M, is included in the signed bundle sent to C_B in step 19. Hence, $\mathcal{A}$ cannot later replay the messages to either entity.

5. *Key confirmation.* Upon receipt of the message in step 19, C_B can be sure that B has generated the *MK* within the current session by verifying the signature of B on R_M, Id_M and $E_{M_{pub}}(MK)$. However, C_B does not confirm the receipt of *MK* to B. Note that $\mathcal{A}$ can block all the messages exchanged, and network errors might occur, and hence only C_B can be sure that it shares a fresh *MK* with B (until successful use of the key by P).

6. *Key control.* The protocol is an authentication and key transport protocol. B generates the master session key *MK*, and hence B has key control.

6 Using the GAA Framework

We now discuss trust issues and possible applications of the GAA services.

6.1 Trust issues

The nature of the GAA architecture means that the end users implicitly trust the provider of the BSF service. This means that the entity providing this service needs to be selected with care, and it may also mean that the service may not be appropriate for every application. Nevertheless, in the non-electronic world, trusted third parties are relied on for a huge range of services, some very sensitive, and hence this does not appear to be a fundamental obstacle. In addition, if security sensitivity justifies the additional cost, multiple BSF services could be accessed simultaneously, thereby distributing the necessary trust.

6.2 Applications

A wide range of applications for UMTS GAA have been explored — see, for example, Holtmanns et al. [7]. Any other scheme providing a GAA service such as the system described here can support very similar applications.

In ongoing work we are examining ways in which a range of variants of the GAA service can be used to support one time passwords [4]. The schemes enable an *GAA enabled user platform* (e.g. a mobile phone or a trusted commodity computer) to act as a one-time password generator. If a user registers with an application server (establishing a *username* and *password*, a human-memorable weak secret), one-time passwords can be generated as a function of on-demand GAA bootstrapped application-specific keys and the shared password. A prototype of one of the schemes (*Ubipass*[1], which makes use of UMTS GAA services) has been developed in collaboration with the Nokia Research Center in Helsinki. We are currently studying its usability

[1]http://ubipass.research.isg.rhul.ac.uk/

and performance. The same OTP generation protocol (the *OTP agreement* protocol in *Ubipass*) could also be built using the TC GAA service. *Ubipass* provides an Internet one-time password solution which could be deployed to enable the provision of ubiquitous one-time password services for a large class of users.

7 Conclusions

GAA is a framework that enables pre-existing security infrastructures to be used to provide general purpose security services, such as key establishment. We have shown how GAA services can be built on the Trusted Computing security infrastructure, complementing the previously standardised GAA schemes built on the mobile phone infrastructures. The solution described in section 3.3 has been designed to apply to a range of trusted computing technologies. We have also provided an instantiation of this solution as supported by the TCG specifications.

TC GAA provides a way of exploiting the now very widespread trusted computing infrastructure (as supported by PC-based trusted platforms) for the provision of fundamentally important generic security services. Of course, application-specific security protocols building on the infrastructure can be devised independently of any generic service and, indeed, there is a large and growing literature on such schemes. However, the definition of a standard GAA-based security service enables the trusted computing infrastructure to be exploited in a simple and uniform way, and it also provides an opportunity for trusted computing aware third parties to provide novel security services. This may help with providing the business case necessary for the emergence of the wide range of third party security services necessary to fully realise the goals of trusted computing.

References

[1] 3rd Generation Partnership Project (3GPP). *Technical Specification Group Services and Systems Aspects, Generic Authentication Architecture (GAA), Generic Bootstrapping Architecture, Technical Specification TS 33.220, Version 9.2.0*, 2009.

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

[3] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *The 11th ACM Conference on Com-*

puter and Communications Security, CCS 2004, Washingtion, DC, USA, October 25-29, 2004, Proceedings, pages 132–145. ACM, 2004.

[4] Chen Chunhua, Chris Mitchell, and Tang Shaohua. Ubiquitous One-Time Password Service Using the Generic Authentication Architecture. *Mobile Networks and Applications*, to appear.

[5] Eimear Gallery. An overview of trusted computing technology. In Chris J. Mitchell, editor, *Trusted Computing*, pages 29–114. IEE, 2005.

[6] Eimear Gallery and Allan Tomlinson. Secure Delivery of Conditional Access Applications to Mobile Receivers. In Chris J. Mitchell, editor, *Trusted Computing*, pages 195–237. IEE, 2005.

[7] Silke Holtmanns, Valtteri Niemi, Philip Ginzboorg, Pekka Laitinen, and N. Asokan. *Cellular Authentication for Mobile and Internet Services*. John Wiley and Sons, 2008.

[8] International Organization for Standardization, Genève, Switzerland. *ISO/IEC 9798-3:1998, Information technology—Security techniques—Entity authentication—Part 3: Mechanisms using Digital Signature Techniques*, 1998.

[9] Andreas Pashalidis and Chris J. Mitchell. Single sign-on using trusted platforms. In Colin Boyd and Wenbo Mao, editors, *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*, volume 2851 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2003.

[10] Trusted Computing Group. *TCG Software Stack (TSS) Specification Part 1: Commands and Structures, Version 1.2*, 2007.

[11] Trusted Computing Group. *TPM Main, Part 1 Design Principles, TCG Specification, Version 1.2, Revision 103*, 2007.

[12] Trusted Computing Group. *TPM Main, Part 2 TPM Data Structures, TCG Specification, Version 1.2, Revision 103*, 2007.

[13] Trusted Computing Group. *TPM Main, Part 3 Commands, TCG Specification, Version 1.2, Revision 103*, 2007.