Ninja : Non Identity Based, Privacy Preserving Authentication for Ubiquitous Environments

Adrian Leung* and Chris J. Mitchell

Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
{A.Leung,C.Mitchell}@rhul.ac.uk

Abstract. Most of today's authentication schemes involve verifying the identity of a principal in some way. This process is commonly known as entity authentication. In emerging ubiquitous computing paradigms which are highly dynamic and mobile in nature, entity authentication may not be sufficient or even appropriate, especially if a principal's privacy is to be protected. In order to preserve privacy, other attributes (e.g. location or trustworthiness) of the principal may need to be authenticated to a verifier. In this paper we propose Ninja: a non-identity-based authentication scheme for a mobile ubiquitous environment, in which the trustworthiness of a user's device is authenticated anonymously to a remote Service Provider (verifier), during the service discovery process. We show how this can be achieved using Trusted Computing functionality.

Keywords: Security, Privacy, Ubiquitous, Trusted Computing.

1 Introduction

In the Mobile VCE[1] Core 4 research programme on Ubiquitous Services, it is envisaged that in a mobile ubiquitous environment (as shown in figure 1), users (through one of their mobile devices and via some network access technologies) will be able to seamlessly discover, select, and access a rich offering of services and content from a range of service providers. To realise this vision, security and privacy issues must be addressed from the outset, alongside other technological innovations. Only if users are confident that their security and privacy will not be compromised, will we see the widespread adoption of ubiquitous services.

As shown in figure 1, one of the primary aims for a user is to access the various services that are offered. But, before any services can be accessed and consumed, they must first be located via a process known as service discovery. Many service discovery schemes [8, 12, 30] have recently been proposed for ubiquitous environments, but few [32, 33] have addressed security and privacy issues,

* This author is supported by the British Chevening/Royal Holloway Scholarship, and in part by the European Commission under contract IST-2002-507932 (ECRYPT).

[1] http://www.mobilevce.com/

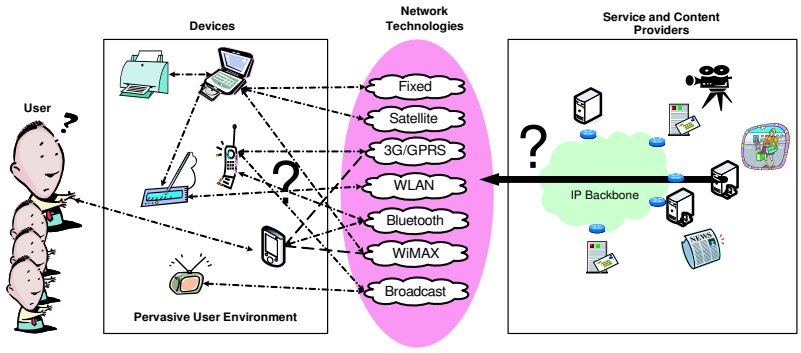

Fig. 1. A Ubiquitous Computing Environment

despite their fundamental importance. It is imperative that the process of service discovery is conducted in a secure and private way, in order to protect the security and privacy of both users and service providers. One fundamental security requirement is mutual authentication between a user and service provider.

Authentication is important for several reasons. Firstly, it is a basic security service upon which a range of other security services (e.g. authorisation) can be built. Secondly, it gives users and service providers assurance that they are indeed interacting with the intended parties, and not some malicious entities. Unfortunately, conventional entity authentication [15] may not be adequate for a ubiquitous environment [10], because an identity may be meaningless in such a setting. Instead, other user attributes [10] may need to be authenticated to a service provider. Furthermore, consumers are becoming increasingly concerned about their privacy [3, 4], and the potential risks (such as identity theft) of leaving any form of digital trail when making electronic transactions. Given a choice, users may prefer to interact with service providers anonymously (or pseudonymously). Under these circumstances, it may in fact be undesirable to authenticate the identity of a user. Preserving user privacy can be particularly challenging in a ubiquitous environment [7, 29], and if privacy is preserved (through user anonymity), how can we then convince a service provider that an anonymous user is trustworthy? This is the challenge addressed in this paper.

We thus propose Ninja: a non-identity based, privacy preserving, mutual authentication scheme designed to address the service discovery security and privacy challenges in a mobile ubiquitous environment. During service discovery, a service user and service provider are mutually authenticated, whilst preserving the privacy of a user. Instead of authenticating the user identity to a service provider, the user's trustworthiness is authenticated. Our scheme employs two key functionalities of Trusted Computing (TC) technology [2, 18], namely, the Integrity Measurement, Storage and Reporting Mechanism, and the Direct Anonymous Attestation Protocol. We therefore implicitly assume that a user device is equipped with TC functionality; current trends suggest that this is a reasonable assumption for the near future. Ninja is an application layer solution, and pos-

sesses many desirable security and privacy properties, such as: user anonymity, service information confidentiality, unlinkability, and rogue blacklisting.

The remainder of the paper is organised as follows. In section 2, we discuss various service discovery security and privacy issues. Section 3 describes the relevant Trusted Computing functionality. In section 4, we present the Ninja authentication scheme, and in section 5 analyse its security. In the penultimate section, we discuss related work, and finally, conclusions are drawn in section 7.

2 Service Discovery Security & Privacy Issues

In this section, we focus on the security and privacy issues arising from the service discovery process in a ubiquitous computing environment.

2.1 Adversary Model

Service discovery typically involves interactions between a user, the user's device, a service provider, and at times, a trusted third party. Unfortunately, these entities may be malicious, and pose a variety of threats to the service discovery process and to the participating entities. Against this backdrop, we identify eight adversary settings, covering both active and passive adversaries. They are:

1. **Innocent User with Malicious Device (IUMD)** Unbeknownst to a user, his/her device is compromised (e.g. with malware, keystroke-logger, etc).
2. **Malicious User with Trustworthy Device (MUTD)** A malicious user who has taken physical control of (e.g. stolen) another entity's device.
3. **Malicious User with Malicious Device (MUMD)** The combination of IUMD and MUTD.
4. **Malicious Service Provider(s) (MSP)** A MSP's main motive is to masquerade to a user as a legitimate service provider.
5. **Curious Service Provider(s) (CSP)** A CSP is not malicious, but seeks only to learn more about the behaviour of its users.
6. **Malicious Man-in-the-Middle (MitM)** A MitM's actions are intended to disrupt the proper operation of the service discovery process.
7. **Curious Trusted Third Party (CTTP)** A CTTP performs its role correctly, but also seeks to learn about the activities and habits of a user.
8. **Passive Eavesdropper (PE)** A PE does not disrupt the communication, but monitors it to learn the content and the entities involved.

2.2 Security and Privacy Threat Model

We now consider possible service discovery threats. We also consider what threats are posed by each of the above adversarial settings, and present them in a Threats versus Adversary Matrix (in Table 1). The service discovery threats are:

1. **Spoofing** A malicious entity may masquerade as a legitimate service provider or service user either by sending false service advertisements/requests, through replay, or by man-in-the-middle attacks.

2. **Information Disclosure**
   (a) **User's Personally Identifiable Information (PII)** During the process of service discovery, a user's PII, such as his/her identity (e.g. in the form of a long lived key) or physical location, may be revealed (either willingly or unwillingly) to a service provider or passive eavesdropper.
   (b) **Service Information (SI)** By observing the service information exchanged by a user and service provider (e.g. the service request types), a passive adversary may build up a profile of the user. This information may later be used to predict future patterns and habits of the user. The privacy of the user is potentially compromised as a result.
3. **Profile Linking** Colluding service providers may buy, sell or exchange information about their users or customers. This could not only provide service providers with monetary benefits, but also enhance their business intelligence and gain competitive advantage, e.g. if they are able to build more comprehensive user profiles (with or without their permission). Finally, the consequences for user privacy could be even more serious if a trusted third party colludes with service providers.
4. **Encouragement of Rogue Behaviour** With the knowledge that privacy enhancing technologies are employed to protect their identities, users may be tempted to "misbehave" or act maliciously, since it may be difficult or even impossible for service providers to determine who is misbehaving.

Table 1. Threats and Adversary Matrix

Threats vs Adversary	IUMD	MUTD	MUMD	MSP	CSP	MitM	CTTP	PE
Spoofing	✓	✓	✓	✓		✓		
User Identity Disclosure	✓	✓	✓	✓	✓	✓	✓	✓
SI Disclosure	✓		✓		✓	✓		✓
User Profile Linking					✓	✓	✓	
Rogue Behaviour Denial	✓		✓					

2.3 Specific Security and Privacy Requirements

From the above threat analysis, we derive the corresponding security and privacy requirements:

- **Mutual Authentication** This is one of the most important requirements, as it can prevent spoofing (by malicious users or service providers). The mutual authentication scheme should also be designed to prevent replay and man-in-the-middle attacks. To protect privacy, a user may want to remain anonymous to a service provider. So, instead of authenticating his identity to a service provider, the user may want to somehow prove or authenticate his "trustworthiness" to the service provider.
- **User Anonymity** Unique user identifying information (e.g. an identifier or a long lived key) should not be divulged to a service provider during service discovery. A user may interact with service providers using a pseudonym.

- **Service Information Confidentiality** To further preserve the privacy of the user, service information originating from the user may be encrypted.
- **Unlinkability** Colluding service providers should not be able to link the activities of a user. Similarly, when a trusted third party colludes with a service provider, they should not be able to correlate the actions of a particular user. In other words, it should be impossible for colluding service providers to tell if two sets of prior service transactions (made with different providers) involved the same or different user(s).
- **Transaction Linkability/History** For billing or other purposes, it may be necessary for a service provider to maintain the transaction histories of its users. A service provider may thus need to be able to determine whether a particular user is a repeat user (and, if so, which one) or a first time user, whilst still being unable to determine the unique identity of the user. This is not always a requirement, and providing it may require user consent.
- **Rogue Blacklisting** Service providers should be able to identify and blacklist malicious and untrustworthy hosts.

2.4 Challenges

We therefore need to devise a mutual authentication scheme that meets all these requirements. This is particularly challenging for several reasons. Conventional mutual authentication schemes normally require the user identity to be authenticated to a verifier. But here, user privacy is a priority, and so user anonymity is required during authentication. How then can we convince a service provider that an anonymous user is trustworthy? Also, if user anonymity is provided, can we detect malicious or illegitimate users? We are, in fact, trying to achieve security and privacy concurrently, whilst protecting the interests of both users and service providers. This is the challenge addressed here.

Our scheme, Ninja, allows a user to authenticate the service provider, whilst simultaneously allowing a service provider to anonymously authenticate a user's trustworthiness. The scheme is so called because the process is to some extent analogous to the process of a ninja assassinating a person in Japanese folklore[2].

3 Trusted Computing Overview

Trusted Computing, as developed by the Trusted Computing Group[3] (TCG), is a technology designed to enhance the security of computing platforms. It involves incorporating trusted hardware functionality, or so called "roots of trust", into platforms. Users can thereby gain greater assurance that a platform is behaving in the expected manner [2, 18, 26]. The trusted hardware, in the form of

[2] A ninja is asked to assassinate someone (Bob) whom he has never met; he is only given Bob's photograph. When they meet, the ninja authenticates Bob physically. Bob, on seeing a ninja with a sword, knows (trusts) that the ninja wishes to kill him, but does not need to know the ninja's real identity, whose anonymity is preserved.

[3] http://www.trustedcomputinggroup.org/

a hardware component called the Trusted Platform Module (TPM), is built into a host platform. The TPM provides the platform with a foundation of trust, as well as the basis on which a suite of Trusted Computing security functionalities is built. The TPM and its host are collectively referred to as a *Trusted Platform*.

We next introduce the keys used by a TPM, as well as the Trusted Computing functionality used in our scheme, i.e. the Integrity Measurement, Storage and Reporting Mechanisms, and the Direct Anonymous Attestation (DAA) Protocol. The descriptions below are based upon v1.2 of the TCG TPM specifications [26].

3.1 TPM Keys and Identities

Each TPM has a unique 2048-bit RSA key pair called the *Endorsement Key* (EK). The EK is likely to be generated by the TPM manufacturer, and the EK private key, together with a certificate for the corresponding public key, can be used to prove that a genuine TPM is contained in a platform. However, since a TPM only has one such key pair, a TPM can be uniquely identified by its EK.

The EK is therefore only used in special circumstances. A TPM can, however, generate an arbitrary number of 2048-bit RSA *Attestation Identity Key* (AIK) key pairs, which are used for interacting with other entities. AIKs function as pseudonyms for a trusted platform, and platform privacy can be achieved by using a different AIK to interact with different entities. In order to prove that a particular AIK originates from a genuine TPM, a platform has to prove that the AIK public key is associated with a genuine trusted platform; this involves using the EK with a trusted third party in such a way that the AIK cannot be linked to a particular EK, even by the trusted third party that sees the EK public key. The DAA protocol (discussed in Section 3.3) is used to support this process.

3.2 Integrity Measurement, Storage and Reporting

Integrity Measurement, Storage and Reporting (IMSR) is a key feature of Trusted Computing that builds on the three Roots of Trust in a trusted platform: a *root of trust for measurement* (RTM), a *root of trust for storage* (RTS), and a *root of trust for reporting* (RTR). Together, they allow a verifier to learn the operational state of a platform, and hence obtain evidence of a platform's behaviour. This functionality is extremely important, as a platform may potentially enter a wide range of operational states, including insecure and undesirable states.

Integrity Measurement Integrity measurement involves the RTM, a computing engine in the TPM, measuring a platform's operational state and characteristics. The measured values, known as integrity metrics, convey information about the platform's current state (and hence trustworthiness).

Integrity Storage Details of exactly what measurements have been performed are stored in a file called the *Stored Measurement Log* (SML). Using the RTS, a

digest (i.e. a cryptographic hash computed using Secure Hash Algorithm 1 (SHA-1) [19]) of the integrity metrics is saved in one of the TPM's internal registers, called *Platform Configuration Registers* (PCRs). The SML contains the sequence of all measured events, and each sequence shares a common measurement digest. Since an SML may become fairly large, it does not reside in the TPM. Integrity protection for the SML is not necessary, since it functions as a means to interpret the integrity measurements in the PCRs, and any modifications to the SML will cause subsequent PCR verifications to fail.

There are only a limited number of PCRs in the TPM. So, in order to ensure that previous and related measured values are not ignored/discarded, and the order of operations is preserved, new measurements are appended to a previous measurement digest, re-hashed, and then put back into the relevant PCR. This technique is known as *extending* the digest, and operates as follows:

$$PCR_i[n] \leftarrow \textit{SHA-1}\,(PCR_{i-1}[n] \,||\, \text{New integrity metric}),$$

where $PCR_i[n]$ denotes the content of the nth PCR after i extension operations, and $||$ denotes the concatenation of two messages.

Integrity Reporting The final phase of the IMSR process is Integrity Reporting. The RTR has two main responsibilities during Integrity Reporting:

1. to retrieve and provide a challenger with the requested integrity metrics (i.e. the relevant part of the SML, and the corresponding PCR values); and
2. to *attest to* (prove) the authenticity of the integrity metrics to a challenger by signing the PCR values using one of the TPM's AIK private keys.

To verify the integrity measurements, the verifier computes the measurement digest (using the relevant portion of the SML), compares it with the corresponding PCR values, and checks the signature on the PCR values. The process of integrity reporting is also often referred to as *Attestation*.

3.3 Direct Anonymous Attestation

Direct Anonymous Attestation (DAA) [5, 18] is a special type of signature scheme that can be used to anonymously authenticate a TCG v1.2 compliant platform to a remote verifier. The key feature that DAA provides is the capability for a TPM (a prover) to convince a remote verifier that:

- it is indeed a genuine TPM (and hence it will behave in a trustworthy manner) without revealing any unique identifiers;
- an AIK public key is held by a TPM, without allowing colluding verifiers to link transactions involving different AIKs from the same platform.

The above-mentioned features help to protect the privacy of a TPM user. Another important feature of DAA is that the powers of the supporting Trusted Third Party (DAA Issuer) are minimised, as it cannot link the actions of users (even when it colludes with a verifier), and hence compromise the user's privacy.

DAA allows a prover to anonymously convince a remote verifier that it has obtained an anonymous attestation credential, or DAA Certificate (a Camenisch-Lysyanskaya (CL) signature [6]), from a specific DAA Issuer (Attester). The DAA Certificate also serves to provide the implicit "link" between an EK and an AIK. The DAA scheme is made up of two sub-protocols: *DAA Join* and *DAA Sign*. We now provide a simplified description of these two sub-protocols [5].

DAA Join Protocol The Join protocol enables the TPM to obtain a DAA Certificate (also known as an anonymous attestation credential) from the DAA Issuer. The Join protocol is based on the CL signature scheme [6].

Let (n, S, Z, R) be the DAA Issuer public key, where n is an RSA modulus, and S, Z, and R are integers modulo n. We assume that the platform (TPM) is already authenticated to the DAA Issuer via its Endorsement Key, EK.

The platform (TPM) first generates a DAA secret value f, and makes a commitment to f by computing $U = R^f S^{v'} \bmod n$, where v' is a value chosen randomly to "blind" f. The platform (TPM) also computes $N_I = \zeta_I^f \bmod \Gamma$, where ζ_I is derived from the DAA Issuer's name, and Γ is a large prime. The platform (TPM) then sends (U, N_I) to the DAA Issuer, and convinces the DAA Issuer that U and N_I are correctly formed (using a zero knowledge proof [13, 14]). If the DAA Issuer accepts the proof, it will sign the hidden message U, by computing $A = (\frac{Z}{US^{v''}})^{1/e} \bmod n$, where v'' is a random integer, and e is a random prime. The DAA Issuer then sends the platform (i.e. the TPM) the triple (A, e, v''), and proves that A was computed correctly. The DAA Certificate is then $(A, e, v = v' + v'')$.

DAA Sign Protocol The Sign protocol allows a platform to prove to a verifier that it is in possession of a DAA Certificate, and at the same time, to sign and authenticate a message. The platform signs a message m using its DAA Secret f, its DAA Certificate, and the public parameters of the system. The message m may be an Attestation Identity Key (AIK) generated by the TPM, or an arbitrary message. The platform also computes $N_V = \zeta^f \bmod \Gamma$ as part of the signature computation (the selection of ζ will be discussed in the next section). The output of the Sign protocol is known as the DAA Signature, σ.

The verifier verifies the DAA Signature σ, and on successful verification of σ, is convinced that:

1. the platform has a DAA Certificate (A, e, v) from a specific DAA Issuer, and hence it is a genuine TPM containing a legitimate EK; this is accomplished by a zero-knowledge proof of knowledge of a set of values f, A, e, and v such that $A^e R^f S^v \equiv Z \pmod n$;
2. a message m was signed by the TPM using its DAA secret f, where f is the same as the value in the DAA Certificate (used in step 1); if m includes an AIK public key, then the AIK originates from a genuine TPM.

In summary, once a platform (TPM) has obtained a DAA Certificate (which only needs to be done once), it is able to subsequently DAA-Sign as many AIKs as its wishes, without involving the DAA Issuer.

Variable Anonymity Anonymity and unlinkability are provided to a user by using two parameters: ζ, also referred to as the *Base*, and the AIK. The choice of the base directly affects the degree of anonymity afforded to a TPM user. If perfect anonymity is desired, then a different, random, base value should be used for every interaction with a verifier. Conversely, if the same base value is used for every interaction with a verifier, then the verifier can identify that this is the same TPM. In addition, if the same base value is used to interact with different verifiers, then they are able to correlate the activities of a particular TPM. (A more detailed discussion of the effects of choices of base values is given in [25]).

As discussed in Section 3.1, a TPM is capable of generating multiple platform identities, simply by generating different AIK key pairs. Different AIKs may therefore be used to interact with different verifiers so as to remain unlinkable (provided the base is different).

4 The Ninja Authentication Scheme

In this section, we present the Ninja authentication scheme, designed to mutually authenticate a user (via his platform) and a service provider, whilst preserving the privacy of the user. The Ninja scheme is intended to be used during the service discovery process, immediately prior to service provisioning. It is designed to meet all the security and privacy requirements set out in Section 2.3.

First, we introduce the entities participating in the protocol. Next, we state the assumptions upon which the scheme is based. Finally, we describe the operation of the scheme.

4.1 The Entities

The entities involved in the protocol are as follows.

- The **Service User**, often a human, is the end consumer of a service.
- The trusted platform, or **Platform** in short, is the device which a service user will use to interact with other entities.
- The **DAA Issuer** issues DAA Certificates to legitimate platforms.
- The **Service Provider** is an entity that provides service(s) or content (e.g. music, videos, podcasts) to service users (via the platform). A service provider also acts as the verifier of a platform's DAA Signatures.

4.2 Assumptions

The correct working of the scheme relies on a number of assumptions.

- The service user is already authenticated to the platform.
- The platform running the Ninja protocol is equipped with TC functionality conforming to v1.2 of the TCG specifications [26].

- Each service provider possesses one or more X.509 v3 [17] public key certificates, issued by trustworthy Certification Authorities (CAs). The platform is equipped with the root certificates of these trusted CAs, and is capable of periodically receiving Certificate Revocation List (CRL) updates.
- Service users and service providers have loosely synchronised clocks (e.g. within an hour of each other). This enables service users and service providers to check that a service advertisement message or a service reply message is fresh (or recent enough).
- Service providers have set up system parameters, p and g, for the Diffie–Hellman (DH) key agreement protocol [11], prior to the protocol run. The (large) prime p is chosen such that $p-1$ has a large prime factor q (e.g. $p = 2q+1$), and g is chosen to have multiplicative order q, so that it generates a multiplicative subgroup of $\mathbb{Z}_p^*$ of prime order q.

Finally note that the scheme is independent of the underlying transport and network layer protocols, as it is purely an application layer solution.

4.3 The Scheme

Before describing the scheme, we first introduce some notation (see Table 2).

Table 2. Notation

Notation	Description
P	The Platform
SP	The Service Provider
I	The DAA Issuer
f	A non-migratable DAA secret value generated by the TPM
v', v'', e	DAA parameters (described in Section 3.2)
p, g	System parameters for DH–key agreement
$SrvAdv$	Service Advertisement Message
$SrvRep$	Service Reply Message
$SrvInfo$	Service Information
$AdvID$	An Advertisement ID number
t_A	A Timestamp generated by a principal, A
N	A Nonce (a random value)
ID_A	The identity of a principal, A
(EK_{pk}, EK_{sk})	The pair of Public and Private Endorsement Keys
(AIK_{pk}, AIK_{sk})	A pair of Public and Private Attestation Identity Keys
(PK_A, SK_A)	The Public and Private Key pair of a principal, A
$Cert_A$	An X.509 v3 Public Key Certificate for a principal, A
H	A cryptographic hash-function
$Enc_K(M)$	The encryption of a message, M, using the key K
$Dec_K(M)$	The decryption of a message, M, using the key K
$MAC_K(M)$	The message authentication code (MAC) of a message, M, computed using the key K
$Sig_K(M)$	A signature on a message, M, signed using the key K

The Ninja scheme involves three distinct phases, the *Join Phase*, the *Mutual Authentication Phase*, and the *Verification Phase*, described below.

Join Phase The *Join Phase* enables a platform to obtain a *DAA Certificate* from a *DAA Issuer*. The platform later uses this DAA Certificate, in the mutual authentication phase, to anonymously authenticate itself to a service provider. The entities involved are the *Platform*, P, and the *DAA Issuer*, I. Note that the Join Phase is identical to the DAA Join Protocol specified in [5]; it may have taken place before a device is shipped to the user. The sequence of events is as follows (see also figure 2).

1. The platform (TPM) generates its DAA Secret value f, and a random value v'. It computes U and N_I (as described in Section 3.3), and then sends U, N_I, and its Endorsement Public Key, EK_{pk} to the DAA Issuer.
2. To verify that U originates from the TPM in the platform that owns EK_{pk}, the DAA Issuer engages in a simple *Challenge-Response* protocol with the platform. It generates a random message m, and encrypts m using EK_{pk}. It sends the challenge, $Chl = Enc_{EK_{pk}}(m)$ to the platform.
3. If the platform owns EK_{pk}, it should have the corresponding EK_{sk}, and hence be able to decrypt $Enc_{EK_{pk}}(m)$, to retrieve m. The platform (TPM) then computes and sends the response $r = H(U||m)$ to the DAA Issuer.
4. The DAA Issuer computes $H(U||m)$ using the value of m it sent in step 2, and compares the result with the value of r received from the platform. If the two values agree, then the DAA Issuer is convinced that U originated from the TPM that owns EK_{pk}.
5. Finally, the DAA Issuer generates v'' and e, and then computes A (as described in Section 3.3). The DAA Issuer then sends (A, e, v'') to the platform.
6. The DAA Certificate is (A, e, v), where $v = v' + v''$. The DAA Issuer does not have full knowledge of the DAA Certificate, since the certificate was jointly created by the platform and the DAA Issuer. This property helps preserve the anonymity of the user/platform.

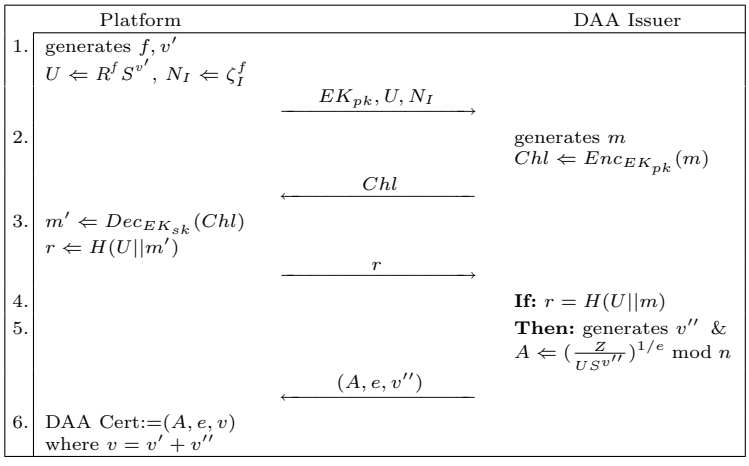

Fig. 2. Join Phase

Mutual Authentication Phase Service discovery typically involves the exchange of service advertisement and service reply messages between the user and service provider. To avoid increasing the communication overheads, we incorporate the authentication mechanisms into these messages. In other words, service discovery and mutual authentication take place concurrently. We now examine how the messages are constructed to achieve our aim of mutual authentication.

The service provider, *SP*, initiates the service discovery and mutual authentication processes by constructing and sending an authenticated service advertisement message, as follows (also shown in figure 3).

1. *SP* generates a random number b and computes $g^b \bmod p$. These values are used later as part of a Diffie-Hellman key agreement protocol to establish a shared key with the user.
2. *SP* constructs the service advertisement message,

$$SrvAdv = (ID_{SP}, SrvInfo, AdvID, N, t_{sp}, g, p, g^b \bmod p).$$

3. *SP* signs *SrvAdv*, using its private key, SK_{SP}, and obtains the signature, $Sig_{SK_{SP}}(SrvAdv)$. *SP* then broadcasts *SrvAdv*, $Sig_{SK_{SP}}(SrvAdv)$, and $Cert_{SP}$ to the platforms of prospective service users:

$$SP \rightarrow \text{platforms} : SrvAdv, Sig_{SK_{SP}}(SrvAdv), Cert_{SP}.$$

Suppose that a prospective user receives the above service advertisement (via his platform), and is interested in the advertised service. The user's platform then authenticates the service provider by retrieving PK_{SP} from $Cert_{SP}$, and then using it to verify $Sig_{SK_{SP}}(SrvAdv)$, and checking to see if the timestamp is valid. If the verification outcome is satisfactory, then, at this point, the service provider is authenticated to the user.

Using the platform, the user now anonymously authenticates itself (i.e. its trustworthiness) to the service provider, as follows (see also Protocol 3).

1. The platform generates an AIK key pair (AIK_{pk}, AIK_{sk}).
2. The platform sends its SML and the corresponding PCR values to the service provider for validation. To further prove that the PCR values originate from the TPM, the TPM signs the PCR values (from *SrvAdv*), using AIK_{sk} (from step 1), to create:

$$Sig_{AIK_{sk}}(PCR||N).$$

   The Nonce N is included to prevent replay attacks.
3. The platform computes $\zeta = H(ID_{SP})$. It then creates a pseudonym, $N_v = \zeta^f$ (where f is the DAA Secret generated during the join phase) for use when interacting with the service provider.
4. To prove that the AIK (from steps 1 and 2) originates from a genuine TPM, the platform DAA-Signs AIK_{pk} using f, *DAA Certificate*, and the other public parameters of the system. The output is the DAA Signature σ (which also includes ζ and N_v).

5. To complete the Diffie-Hellman key agreement protocol, the platform generates a random number a, and computes:

$$g^a \bmod p \quad \text{and} \quad K = (g^b)^a \bmod p.$$

6. The platform constructs the Service Reply message as:

$$SrvRep = (AdvId, SrvInfo, AIK_{pk}, SML, Sig_{AIK_{sk}}(PCR||N), \sigma, t_p).$$

7. The platform encrypts $SrvRep$ using the key K_1, and then computes a MAC of the encrypted $SrvRep$ using the key K_2, where $K = K_1||K_2$, to give:

$$Enc_{K_1}(SrvRep) \quad \text{and} \quad MAC_{K_2}(Enc_{K_1}(SrvRep)).$$

8. The user sends $Enc_{K_1}(SrvRep)$, $MAC_{K_2}(Enc_{K_1}(SrvRep))$, and $g^a \bmod p$ to the service provider.

$$P \rightarrow SP : Enc_{K_1}(SrvRep), MAC_{K_2}(Enc_{K_1}(SrvRep)), g^a \bmod p.$$

	Platform		Service Provider
1.			generates b, $x \Leftarrow g^b \bmod p$
2.			constructs $SrvAdv{:=}(ID_{SP}, SrvInfo, AdvID, N, t_{sp}, g, p, x)$
3.			$S_x \Leftarrow Sig_{SK_{SP}}(SrvAdv)$
		$\xleftarrow{SrvAdv, S_x, Cert_{SP}}$	
(1)	generates AIK_{pk}, AIK_{sk}		
(2)	retrieves SML & PCR, $S_y \Leftarrow Sig_{AIK_{sk}}(PCR||N)$		
(3)	$\zeta \Leftarrow \mathrm{H}(ID_{SP})$, $N_v \Leftarrow \zeta^f$		
(4)	$\sigma \Leftarrow$ DAA-Signs (AIK_{pk})		
(5)	generates a, $y \Leftarrow g^a \bmod p$ $K \Leftarrow (g^b)^a \bmod p$		
(6)	constructs $SrvRep{:=}(AdvId, SrvInfo, S_y, SML, AIK_{pk}, \sigma, t_p)$		
(7)	$E_x \Leftarrow Enc_{K_1}(SrvRep)$, $MAC_{K_2}(E_x)$		
		$\xrightarrow{E_x, MAC_{K_2}(E_x), y}$	
(8)			

Fig. 3. Mutual Authentication Phase

Verification Phase On receiving a service reply message from a platform, the service provider SP performs the following steps to verify its trustworthiness.

1. SP computes $K = (g^a)^b \bmod p$ and hence obtains K_1 and K_2 (where $K = K_1||K_2$). SP then checks the integrity of the received value of $Enc_{K_1}(SrvRep)$ by recomputing the MAC using K_2 and comparing it with the received value.
2. SP extracts $SrvRep$ by decrypting $Enc_{K_1}(SrvRep)$ using K_1. SP also checks that the timestamp t_p extracted from $SrvRep$, is valid.

3. *SP* verifies the DAA Signature σ, and is thus convinced that the platform is in possession of a legitimate DAA Certificate from a specific DAA Issuer, which implies that a genuine TPM is contained in the platform.
4. *SP* is also convinced that AIK_{pk} was signed using the platform's DAA Secret f. Even though the value of f is never revealed to SP, SP knows that the value is related to a genuine DAA Certificate.
5. *SP* checks that the nonce N is the same as that sent in *SrvAdv*.
6. *SP* verifies the trustworthiness of the platform by examining the platform integrity measurements. This involves recursively hashing the values contained in the SML, and comparing them with the corresponding PCR values.
7. If *SP* is satisfied with the integrity measurements, then the platform (and hence the user) is authenticated to *SP*.

To authenticate to another service provider, the user platform should generate a new AIK key pair, but only needs to repeat the mutual authentication phase, i.e. it does not need to perform the join phase again. The user platform should also use a different N_v value.

5 Security Analysis and Discussion

We now assess the scheme against our security and privacy requirements.

Mutual Authentication Mutual authentication is achieved in the following way. A service provider is first authenticated to a prospective service user through a service advertisement message, protected using conventional cryptographic mechanisms (e.g. as enabled by a PKI). If a prospective user is interested in the service, then the trustworthiness of the user platform is anonymously authenticated to the service provider via a service reply message, using DAA.

The scheme is also resistant to the following attacks.

- **Replay:** The use of the timestamps t_{sp} and t_p in the *SrvAdv* and *SrvRep* messages allows the recipients to check that they are fresh or recent (enough). An adversary which knows an old session key K may be able to decrypt an old *SrvRep* message, and could try to use the corresponding old signature, $Sig_{AIK_{sk}}(PCR||N)$, to reply to a new *SrvAdv* message. This will fail because the signature is computed as a function of the nonce N from *SrvAdv*, and a replayed signature will have been computed using a different value of N.
- **Man-in-the-Middle (MitM):** Since *SrvAdv* is authenticated, a MitM cannot masquerade as an SP to a user. A MitM can make a response on its own behalf (as can anyone receiving *SrvAdv*). However, a MitM cannot masquerade as a legitimate user by manipulating the *SrvRep* message. If it tries to generate a *SrvRep* with a different Diffie-Hellman parameter y, then it can only generate a completely new response, since it cannot decrypt a *SrvRep* generated by another user. If it leaves y unchanged, then any modifications to *SrvRep* will be detected by the service provider, since it is integrity protected using a MAC computed as a function of the Diffie-Hellman key.

User Anonymity The public part of the Endorsement Key, EK_{pk}, is never disclosed to a service provider, since it would function as a permanent identifier for the platform. Users instead interact with service providers using AIKs, which act as pseudonyms. Since it is computationally infeasible for service providers, or even the DAA Issuer, to link two AIK public keys from the same platform (see Section 3.3), users will remain anonymous to service providers (e.g. CSPs), as well as curious DAA Issuers (i.e. CTTPs) and passive eavesdroppers.

Service Information (SI) Confidentiality A *SrvRep* message contains service information which, if disclosed, could reveal a user's service preferences and habits, thereby compromising user privacy. To prevent such a disclosure (e.g. to eavesdroppers or a MitM), *SrvRep* is encrypted using a secret key known only to the service user and the service provider. Whilst there is nothing to prevent a MSP from divulging the SI of an anonymous user, the user's SI confidentiality is still preserved, as the MSP is unable to determine which SI corresponds to which user.

Unlinkability/Collusion Resistance User platforms should interact with different SPs using different AIK public keys and N_v values. It is computationally infeasible for colluding service providers to link these keys (see Section 3.3), i.e. a user's service activities with different service providers are unlinkable. This remains true even in the case of a colluding DAA Issuer (i.e. a CTTP), again as discussed in Section 3.3. Our scheme is therefore resistant to two or more colluding SPs (the CSP case), as well as a DAA Issuer colluding with one or more SPs (the CTTP case).

Transaction History For business reasons (e.g. to support customer loyalty rewards or discounts), it may be necessary for service providers to link a repeat service user. This can be achieved without compromising a user's privacy or anonymity if a service user always uses the same value of N_v to interact with a particular service provider. A service user will not need to store N_v, as it will be recovered during re-computation (since ζ and f should remain unchanged).

Blacklisting malicious parties A detected rogue service provider can be added to the appropriate CRL, enabling users to avoid known fraudulent SPs. Similarly, an SP may want to blacklist a misbehaving or malicious user, to bar this user from future service interactions. This requires a means for the SP to recognise a malicious platform, whilst it remains anonymous. This can be achieved by blacklisting platform pseudonyms, i.e. the N_v values of such platforms. Blacklisting the AIK will not work, as a rogue user can simply generate a new AIK, DAA-Sign it, and then interact with the service provider again.

A rogue user could only avoid detection by obtaining a new pseudonym, N_v. This would involve using a new value for f (the DAA secret). Although a TPM could generate a new f value, it is unlikely that it will be able to obtain a DAA

Certificate for it. DAA certificate issue is expected to be subject to careful checks, and a platform is not expected to possess more than one DAA Certificate from a DAA Issuer. Also, if a DAA Certificate (i.e. a triple of values A,e,v) and the value f are found in the public domain (e.g. on the Internet), then they should be sent to all potential service providers for blacklisting. The service providers can then add them to privately maintained lists of rogue keys.

6 Related Work

Apart from being unsuitable for ubiquitous computing environments [30], existing service discovery approaches (such as Java Jini [23], UPnP [28], SLP [16], DEAPspace [20] and Salutation [22]) do not address the privacy issues raised here. Zhu et al. describe a privacy preserving service discovery protocol [31, 32], where users and service providers progressively reveal Personally Identifiable Information (PII) to each other. A user's PII is eventually divulged to a service provider, and so service providers could still collude and link user activities. Abadi and Fournet proposed a private authentication scheme [1], which protects two communicating principals' privacy (identity and location) from third parties. This only protects a user's PII against eavesdropping third parties, and not from the service providers. Ren et al.'s privacy preserving authentication scheme [21] uses blind signatures and hash chains to protect the privacy of service users. This scheme requires a mobile user and service to authenticate each other via some out of band mechanisms, prior to a privacy-preserving service interaction. This may not be a realistic approach for a mobile ubiquitous environment.

In the k-Times Anonymous Authentication scheme [24], a user can anonymously access a service a predetermined number of times (as decided by the service provider). This approach is extremely inflexible for a ubiquitous environment. For instance, a service provider cannot prevent a malicious user from having future service interactions. In the Chowdhury et al. Anonymous Authentication scheme [9], users interact with different service providers using different surrogates (one-time values) every time, to preserve user anonymity. However, the trusted 'Issuing Authority', can still link user activities. Similarly, in v1.1 of the TCG specifications [2, 27], a user's activities are unlinkable by different service providers, but if the trusted 'Privacy CA' colludes with the service providers, then the activities of a user are linkable, and his/her privacy will hence be compromised. In the Ninja scheme, the trusted third party, i.e. the DAA Issuer, is unable to collude with service providers and link the activities of a user.

7 Conclusions

We identified security and privacy threats that may arise during service discovery in a ubiquitous computing environment; we also derived corresponding security and privacy requirements. We presented the Ninja mutual authentication scheme, using Trusted Computing functionality, which preserves user privacy. Apart from being communications-efficient (only two messages are required), the

scheme also satisfies all the identified security requirements. To a service user and service provider, security and privacy are both desirable. However, they are potentially conflicting requirements, and it is challenging to achieve them both. However, this is achieved by the Ninja mutual authentication scheme presented here, enabling services to be discovered securely and privately.

In future work we plan to integrate anonymous payment mechanisms into the scheme, and to explore ways to secure the process of service provisioning between a user and a service provider, whilst (again) protecting user privacy. A formal security analysis of the scheme is also being performed.

Acknowledgements. We would like to thank Liqun Chen, Marc Langheinrich, Rene Mayrhofer, Kenny Paterson, and the anonymous reviewers for their valuable comments.

References

1. M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
2. B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. PH PTR, Upper Saddle River, NJ, 2003.
3. F. Bao and R. H. Deng. Privacy protection for transactions of digital goods. In *3rd Int'l Conf. on Information & Communications Security (ICICS'01)*, LNCS 2229, pages 202–213. Springer-Verlag, 2001.
4. B. Berendt, O. Gnther, and S. Spiekermann. Privacy in e-commerce: Stated preferences vs. actual behavior. *Communications of the ACM*, 48(4):101–106, 2005.
5. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *11th ACM Conf. on Computer & Communications Security*, pages 132–145. ACM Press, 2004.
6. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *3rd Conf. on Security in Communication Networks (SCN 2002)*, LNCS 2576, pages 268–289. Springer-Verlag, 2003.
7. R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas. Towards security and privacy for pervasive computing. In *Int'l Symposium on Software Security*, pages 1–15, 2002.
8. D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112, 2006.
9. P. D. Chowdhury, B. Christianson, and J. Malcolm. Anonymous authentication. In *12th Int'l Workshop on Security Protocols*, LNCS 3957, pages 299–305. Springer-Verlag, 2006.
10. S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. Authentication for pervasive computing. In *Int'l Conf. on Security in Pervasive Computing*, LNCS 2802, pages 116–129. Springer-Verlag, 2004.
11. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
12. A. Friday, N. Davies, N. Wallbank, E. Catterall, and S. Pink. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wireless Networks*, 10(6):631–641, 2004.

13. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.
14. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
15. D. Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society, 1996.
16. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608, The Internet Engineering Task Force (IETF), June 1999.
17. R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure. RFC 3280, The Internet Engineering Task Force (IETF), April 2002.
18. C. J. Mitchell, editor. *Trusted Computing*. IEE Press, London, 2005.
19. National Institute of Standards and Technology (NIST). Secure Hash Standard. Federal information processing standards publication (FIPS) 180-2, 2002.
20. M. Nidd. Service discovery in DEAPspace. *IEEE Personal Communications*, 8(4):39–45, 2001.
21. K. Ren, W. Luo, K. Kim, and R. Deng. A novel privacy preserving authentication and access control scheme for pervasive computing environments. *IEEE Transactions on Vehicular Technology*, 55(4):1373–1384, 2006.
22. Salutation Consortium. Salutation Architecture Specification, June 1999. http://www.salutation.org/.
23. Sun Microsystems. Jini Architecture Specification. Version 1.2, Sun Microsystems, Palo Alto, CA, USA, December 2001. http://www.sun.com/software/jini/specs/.
24. I. Teranishi, J. Furukawa, and K. Sako. k-times anonymous authentication. In *ASIACRYPT 2004*, LNCS 3329, pages 308–322. Springer-Verlag, 2004.
25. Trusted Computing Group (TCG). TPM v1.2 Specification Changes. A summary of changes, Trusted Computing Group, Portland, Oregon, USA, October 2003.
26. Trusted Computing Group (TCG). TCG Specification Architecture Overview. Version 1.2, The Trusted Computing Group, Portland, Oregon, USA, April 2004.
27. Trusted Computing Platform Alliance (TCPA). TCPA Main Specification. Version 1.1b, Trusted Computing Group, Portland, Oregon, USA, February 2002.
28. Universal Plug and Play (UPnP) Forum. UPnP Device Architecture. version 1.0, December 2003. http://www.upnp.org/.
29. M. Wu and A. Friday. Integrating privacy enhancing services in ubiquitous computing environments. In *Ubicomp 2002: Security in Ubiquitous Computing*, 2002.
30. F. Zhu, M. Mutka, and L. Li. Service discovery in pervasive computing environements. *IEEE Pervasive Computing*, 4(4):81–90, 2005.
31. F. Zhu, M. Mutka, and L. Ni. Prudent Exposure: A private and user-centric service discovery protocol. In *2nd IEEE Conf. on Pervasive Computing & Communications*, pages 329–328, 2004.
32. F. Zhu, M. Mutka, and L. Ni. A private, secure and user-centric information exposure model for service discovery protocols. *IEEE Transactions on Mobile Computing*, 5(4):418–429, 2006.
33. F. Zhu, W. Zhu, M. W. Mutka, and L. Ni. Expose or not? A progressive exposure approach for service discovery in pervasive computing environments. In *3rd IEEE Conf. on Pervasive Computing & Communications*, pages 225–234, 2005.