Remote Attestation Mechanism for User Centric Smart Cards using Pseudorandom Number Generators

Raja Naeem Akram[1], Konstantinos Markantonakis[2], and Keith Mayes[2]

[1] Cyber Security Lab, Department of Computer Science, University of Waikato, Hamilton, New Zealand.
[2] ISG Smart card Centre, Royal Holloway, University of London. Egham, Surrey, United Kingdom
rnakram@waikato.ac.nz, {k.markantonakis, keith.mayes}@rhul.ac.uk

Abstract. User Centric Smart Card Ownership Model (UCOM) gives the "freedom of choice" of respective applications to the smart card users. The user-centric architecture requires a trusted entity to be present on the smart card to provide security assurance and validation to the requesting application providers. In this paper, we propose the inclusion of a trusted computing platform for smart cards that we refer as the Trusted Environment & Execution Manager (TEM). This is followed by the rationale behind the changes to the traditional smart card architecture to accommodate the remote security assurance and validation mechanism. We propose an attestation protocol that provides an on-demand security validation of a smart card by its respective manufacturer. Finally, the attestation protocol is informally analysed, and its test implementation and performance measurements are presented.

1 Introduction

The ecosystem of the User Centric Smart Card Ownership Model (UCOM) [1] in centred around smart cards that have to implement adequate security and operational functionality to support a) enforcement of security policies stipulated by the card platform and individual Service Providers (SPs) for their respective applications, and b) operational functionality that enables an SP to manage its application(s), and a cardholder to manage her ownership privileges. The smart card architecture has to represent this change in ownership architecture. For this purpose, we require a trusted module as part of the smart card architecture. The module would validate the current state of the platform to requesting entities in order to establish the trustworthiness of a smart card in the UCOM ecosystem.

In the UCOM, the card manufacturers make sure that smart cards have adequate security and operational functionality to support user ownership. In addition, the cardholder manages her relationship with individual SPs. These relationships enable her to request installation of their applications. Before leasing an application, SPs will require an assurance of the smart card's security and

reliability. This assurance will be achieved through a third party security evaluation of the smart cards before they are issued to individual users. Furthermore, to provide a dynamic security validation [2], the evaluated smart cards implement an attestation mechanism. The attestation mechanism should accommodate remote validation, as in the UCOM an SP will not always have physical access to the smart card. In addition, the attestation mechanism will certify that the current state of the smart card is as evaluated by the independent third party. Therefore, the trust architecture in the UCOM is based on the adequacy of the third party evaluation, and the security and reliability of the remote attestation mechanism.

1.1 Contributions

In this paper, we propose a smart card remote attestation mechanism based on Pseudorandom Number Generators. The paper also proposes an attestation protocol, both the protocol and attestation mechanism is implemented, and evaluated, along with presenting the underlying performance measurements.

1.2 Organisation

Section 2, discusses the major component that provides security and reliability assurance to (remote) requesting entities: attestation handler and self-test manager. Subsequently, we extend the discussion to the remote attestation mechanism in section 3 and propose two attestation algorithms based on pseudorandom number generators. In section 4 we propose an attestation protocol; in section 5 we detail an informal analysis and test implementation results of the attestation protocol. Finally, concluding the paper in section 6.

2 Proposed Components to Support Attestation Mechanism

The crucial components that support the attestation mechanism are discussed below. Both of these are part of the TEM, and for an indepth discussion on TEM and Security Assurance & Validation Mechanism for UCOM please consult [2,3]. The difference between these two modules (i.e. the attestation handler and the self-test manager) of the TEM is that one focuses on the software and the other on the hardware. However, in the proposed attestation mechanism (section 3) they complement each other to provide proof that a smart card is secure, reliable and trustworthy.

2.1 Attestation Handler

During the application installation process, the attestation handler will verify the current state of the platform runtime environment (e.g. security and operationally sensitive parts of the Smart Card Operating System) and affirm to the

appropriate SP that the platform is as secure and reliable as it is claimed to be by the (third party) evaluation certificate [2]. Once the application is installed the relevant SP can ask the TEM to generate its state validation (e.g. signed hash of the downloaded application), ensuring that the application is downloaded without any errors onto the platform. This function of the TEM is similar to the GlobalPlatform's DAP mechanism [4,5].

2.2 Self-test Manager

The self-test mechanism checks whether the smart card is tamper-resistant as certified by a trusted third party evaluation. The aim of the self-test mechanism is to provide a remote hardware validation framework in a way that enables a requesting entity (e.g. an SP) to independently verify it. As our focus is not the hardware end of the smart card, we do not propose any (pure) hardware-based mechanism in this paper, which is one of the possible directions for future research.

A self-test mechanism in the UCSC should provide the properties that are listed below:

1. Robustness: On input of certain data, it should always produce associated output.
2. Independence: When the same data is input to a self-test mechanism implemented on two different devices, they should output different (random) values.
3. Pseudo-randomness: The generated output should be computationally difficult to distinguish from a pseudo-random function.
4. Tamper-evidence: Any attack aiming to access the function should cause irreversible changes which render the device dead.
5. Unforgeable: It should be computationally difficult to simulate the self-test mechanism and mimic the actual deployed function on a device.
6. Assurance: the function should provide assurance (either implicitly or explicitly) based on independent evaluation (e.g. Common Criteria) to requesting entities. The mechanism should not (always) require an active connection with the device manufacturer to provide the assurance.

There are several possibilities for a self-test mechanism in a UCSC including using active (intelligent) shield/mesh [6], the Known Answer Test (KAT) [7], and the Physical Unclonable Function (PUF) [8].

To provide protection against invasive attacks, smart card manufacturers implement an active shield/mesh around the chip. If a malicious user removes the active shield then the chip will be disabled. The self-test mechanism can be associated with this shield to provide a limited assurance that the protective measures of the chip are still in place and active.

Furthermore, Hash-based Message Authentication Code (HMAC) can be deployed with a hard-wired key that would be used to generate a checksum of randomly selected memory addresses that have non-mutable code related to the

Smart Card Operating System (SCOS). This mechanism requires the involvement of the device manufacturer, as the knowledge of the correct HMAC key would be a secret known only to the card manufacturer and its smart cards.

Another potential protection strategy is to utilise Physical Unclonable Functions (PUFs) [8] to provide hardware validation [9]. It is difficult to find a single and consistent definition of PUF in the literature [10]. However, a property description definition of the PUF is provided by Gassend et al. in [8].

Based on the above listed features, table 1 shows the comparison between different possible functions that can act as the self-test mechanism. Although the debate regarding the viability, security, and reliability of the PUFs is still open in both academic circles and industry [11]; for completeness, we consider them as a possible self-test mechanism in table 1. Similar to the PUF, Psuedorandom Number Generators (PRNG) [12] might also be used to implement the self-test mechanism.

Table 1. Comparison of different proposals for self-test mechanism.

Features	**Active-Shield**	**Keyed-HMAC**	**PRNG**	**PUF**
Robustness	Yes	Yes	Yes	Yes
Independence	No	No	Yes	Yes
Pseudo-randomness	No	Yes	Yes	Yes
Tamper-evidence	Yes	–	Yes*	Yes
Unforgeable	No	Yes	Yes*	Yes
Assurance	Yes	No	Yes	Yes*

Note. "Yes" means that the mechanism supports the feature. "No" indicates that the mechanism does not support the required feature. The entry "Yes*" means that it can supports this feature if adequately catered for during the design.

If a manufacturer maintains separate keys for individual smart cards that support the HMAC then it can provide the independence feature. However the HMAC key is hard-wired and this makes it difficult for it to be different on individual smart cards of the same batch. Furthermore, it requires other features to provide tamper evidence, like active-shield. On the other hand, PUFs and adequately designed PRNGs can provide assurance that the platform state and the tamper-resistant protections of a UCSC are still active. In this paper, we propose the PRNG based design for the self-test mechanism that is detailed in section 3.1.

Before we discuss how a self-test manager and an attestation handler can be implemented based on PRNG, we first discuss the overall framework that is responsible for providing security assurance and validation of a smart card.

3 Attestation Mechanisms

In this section, we discuss the attestation mechanism based on PRNGs that combine the functionality of attestation handler and self-test manager discussed in section 2.

3.1 Pseudorandom Number Generator

In this section, we propose the use of a Pseudorandom Number Generator (PRNG) to provide the device authentication, validation, and implicit anti-counterfeit functionality. Unlike (non-simulatable) PUFs, PRNGs are emulatable and their security relies on the protection of their internal state (e.g. input seed values, and/or secret keys, etc.).

The PRNGs implemented in one device will be the same as they are in other devices of the same batch and given the same input, they will produce the same output. Therefore, the manufacturer will populate the PRNG seed file with unique values in each smart card (no two smart cards from the same batch should have the same seed file).

Algorithm 1: Self-test algo for offline attestation based on a PRNG

Input : l; list of selected memory addresses.
Output: S; signature key of the smart card.
Data:
seed; temporary seed value for the PRNG set to zero.
n; number of memory addresses in the list l.
i; counter set to zero.
a; memory address.
k; secret key used to encrypt the signature key of the smart card.
S_e; encrypted signature key using a symmetric algorithm with key k.

```
1  SelfTestOffline (l) begin
2      while i < n do
3          a ⟵ ReadAddressList (l,i)
4          seed ⟵ Hash (ReadMemoryContents (a), seed)
5          i ⟵ i+1
6      if seed ≠ ∅ then
7          k ⟵ GenPRNG (seed)
8      else
9          return testfailed
10     S ⟵ DecryptionFunction (k, S_e)
11     return S
```

The seed file is a collection of inputs that is fed to the PRNG to produce a random number, and it is updated constantly by the PRNG [12]. This will enable a card manufacturer to emulate the PRNG and generate valid Chanllenge-Response Pairs (CRPs: discussed in section 3.2) for a particular device. The PRNG mechanism is not tamper-evident and it relies on the tamper-resistant mechanisms of the smart card to provide physical security. Based on the PRNG, algorithms 1 and 2 show the offline and online attestation mechanism, respectively.

The `SelfTestOffline` takes a list of selected memory addresses l that is illustrated in algorithm 1. The function iterates through the l reading one memory

address at a time, and then generating a hash of the contents stored at the given memory address. In the next step at line six, the function `SelfTestOffline` checks the value of *seed* and if it is not zero it will proceed; otherwise, it will throw a test fail exception. If the *seed* value is not zero then the *seed* is input to the PRNG and a sequence k is generated. The k is used to encrypt the smart card signature key, and if the input to the PRNG at line seven is as expected the signature key will be correctly decrypted.

Algorithm 2: Self-test algo for online attestation based on a PRNG

Input : c; randomly generated challenge sent by the card manufacturer.
Output: r; hash value generated on selected memory addresses.
Data:
$seedfile$; seed file that has a list of non-zero values.
seed; temporary seed value for the PRNG set to zero.
ns; number of entries in a seed file.
s; unique reference to an entry in the $seedfile$.
nc; number of bytes in the c.
i; counter set to zero.
l; upper limit of memory address defined by the card manufacturer.
m; memory address.
mK; HMAC key shared between a smart card and respective card manufacturer

```
1 SelfTestOnline (c) begin
2     while i < nc do
3         s ⟵ ReadChallenge(c, i) % ns
4         seed ⟵ ReadSeedFile(seedfile, s)
5         m ⟵ GenPRNG(seed) % l
6         r ⟵ r ⊕ Hash(ReadMemoryContents(m), mK)
7         i ⟵ i + 1
8     return r
```

The algorithm returns the signature key, which is used by the attestation handler to sign a message. The requesting entity will verify the signed message and if the state of the platform is in conformance with the evaluated state then the signature will be verified; otherwise, it will fail. The signature verification will fail because the decrypted signature key will be different as the input to the PRNG at line seven of the algorithm was different. Therefore, we can assume that if the state is changed, the signature key will change, and the generated signature will not verify.

The PRNG-based online attestation mechanism is illustrated in algorithm 2. The function `SelfTestOnline` takes the challenge c from the card manufacturer as input. The received challenge is treated as an array of bytes and individual bytes of the challenge c are used to generate indexes to the $seedfile$; values stored on these indexes are used to generate memory addresses (within the range specified by the card manufacturer). The contents of the generated memory addresses

are then HMACed and the result is securely sent to the card manufacturer. The SP can use the same process described in algorithm 2 to generate the HMAC result and if the result matches with the one sent by the smart card, then the card manufacturer can ascertain that the current state of the card is trustworthy. At line six of the algorithm 2, we update the *seedfile* with the value stored in 'm'. This update is necessary to avoid generation of the same 'r' if the card manufacturer sends the same challenge 'c'.

In the implementation of the attestation protocol (section 4), we implement the online online attestation based on a PRNG illustrated in the algorithm 2.

3.2 Challenge-Response Pair (CRP) Generation

In the case of the mechanism based exclusively on the PRNG as depicted in algorithm 2, the card manufacturer will provide a set of seed values that is referred to as the seed file. This file is internally updated by the PRNG; however, as the card manufacturer knows the initial seed file it does not need to communicate CRPs with the smart card as it can generate the correct response independently (using the seed file and the PRNG associate with the respective smart card).

3.3 Keys Generation

Individual smart cards have a unique set of cryptographic keys that the card uses for different protocols/mechanisms during its lifetime. Therefore, after the hardware fabrication and masking of the SCOS is completed [13] the card manufacturer initiates the key generation process.

Each smart card will generate a signature key pair that does not change for the lifetime of the smart card. The smart card signature key pair is certified by the card manufacturer, and it is used to provide offline attestation (section 3). Furthermore, in the certificate hierarchy shown in figure 1, the smart card signature key pair is linked with the Platform Assurance Certificate (PAC) [2] via the card manufacturer's certificate. The reason for this is that a malicious user might copy a PAC that belongs to a genuine device and put it on his tampered device and when an SP requests security assurance from the tampered device, it provides the (copied) PAC of a (trusted) genuine device. By ensuring the PAC is tied to genuine devices by the certificate hierarchy shown in figure 1 we can avert such scenarios.

The evaluation authority (e.g. Common Criteria evaluation laboratory) issues a certificate (e.g. a PAC) [2], which certifies that the signature key of the card manufacturer is valid only for the evaluated product. If an adversary can get hold of the manufacturer's signature key pairs then he can successfully masquerade as the smart card; either as a dumb device or by simulating the smart card on a powerful device like a computer.

The smart card will also generate a public encryption key pair that is certified by the smart card signature key. The smart card user signature key pair is used to identify the owner of the device and to provide "proof of ownership" that is beyond the scope of this paper. This signature key is unique to the individual

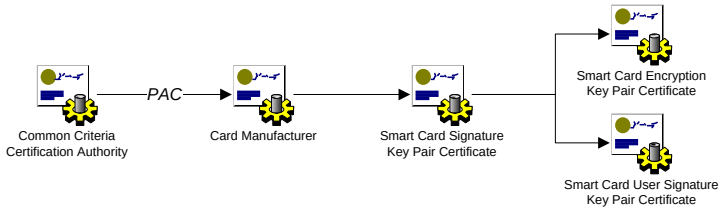

Fig. 1. Certificate hierarchy in the UCOM.

user and it is generated on the successful completion of ownership acquisition process.

Finally, the smart card and card manufacturer share an encryption key for symmetric algorithms (e.g. TDES, AES) and a MAC key. These keys will be used to encrypt and MAC communication-messages between the smart card and the card manufacturer.

4 Attestation Protocol

The attestation protocol, referred as Attestation Protocol (ATP), involves the card manufacturer in the security assurance and validation framework by using the online attestation mechanisms. The aim of the protocol is to provide an assurance to a remote SP that the current state of the smart card is not only secure but also (dynamically and on-demand) attested by the card manufacturer. The card manufacturer generates a security validation message that testifies to the requesting SP that its product is safe and still in compliance with the security evaluation indicated by the associated PAC.

4.1 Protocol Goals

The goals for the attestation protocol are listed as below:

PG-1 Secrecy: During the attestation protocol, the communication messages are adequately protected.
PG-2 Privacy: In the attestation protocol, the identity smart card owner (user) should not be revealed to any eavesdropper or the card manufacturer.

4.2 Intruder's Capabilities

The aim of an adversary $\mathcal{A}$ could be to retrieve enough information to enable him to successfully masquerade as a card manufacturer or as a smart card. Therefore, we assume an adversary $\mathcal{A}$ is able to intercept all messages communicated between a smart card and its manufacturer. In addition, $\mathcal{A}$ can modify, change, replay, and delay the intercepted messages.

If $\mathcal{A}$ is able to masquerade as a card manufacturer then $\mathcal{A}$ can issue fake attestation certificates to individual smart cards, which might compromise the security and privacy of the user and related SPs. On the other hand, if $\mathcal{A}$ is able to compromise the smart card then he can effectively simulate the smart card environment. This will enable him to reverse engineer the downloaded applications and retrieve sensitive data related to the user and application (e.g. intellectual property of the SP).

4.3 Protocol Notation and Terminology

Table 2 summarises the notation used in the proposed attestation protocol.

Table 2. Protocol notation and terminology

Notation	Description
$\mathcal{SC}$	Denotes a smart card.
$\mathcal{SP}$	Denotes a Service Provider.
$\mathcal{CM}$	Denotes the respective card manufacturer of the SC.
$\mathcal{CC}$	Denotes the respective Common Criteria evaluation laboratory that evaluates the $\mathcal{SC}$.
X_i	Indicates the identity of an entity X.
N_X	Random number generated by entity X.
$h(Z)$	The result of applying a hash algorithm (e.g. SHA-256) on data Z.
K_{X-Y}	Long term encryption key shared between entities X and Y.
mK_{X-Y}	Long term MAC key shared between entities X and Y.
B_X	Private decryption key associated with an entity X.
V_X	Public encryption key associated with an entity X.
$e_K(Z)$	Result of encipherment of data Z with symmetric key K.
$f_K(Z)$	Result of applying MAC algorithm on data Z with key K.
VM	The Validation Message (VM) issued by the respective CM to a SC representing that the current state of the SC is as secure as at the time of third party evaluation, which is evidenced by the PAC.
$Sign_X(Z)$	Is the signature on data Z with the signature key belonging to an entity X using a signature algorithm like DSA or based on the RSA function.
$CertS_{X\leftarrow Y}$	Certificate of entity X's signature key, issued by an entity Y.
$CertE_{X\leftarrow Y}$	Certificate of entity X's public encryption key, issued by an entity Y.
$X \rightarrow Y : C$	Entity X sends a message to entity Y with contents C.
$X\|\|Y$	Represents the concatenation of data items X and Y.
SID	Session identifier that is used as an authentication credential and to avoid Denial of Service (DoS) attacks. The SID generated during the protocol run 'n' is used in the subsequent protocol run (i.e. n+1).

4.4 Protocol Description

In this section, we describe the attestation protocol, and each message is represented by ATP-n, where n represents the message number.

$$\textbf{ATP-1.} \quad \begin{aligned} \mathcal{SC} &: mE = e_{k_{SC-CM}}(SC_i||N'_{SC}||CM_i||ReqVal) \\ \mathcal{SC} \rightarrow \mathcal{CM} &: SC_{i'}||mE||f_{mk_{SC-CM}}(mE)||SID \end{aligned}$$

Before issuing the smart card to the user, the $\mathcal{SC}$ and $\mathcal{CM}$ will establish two secret keys; encryption key K_{SC-CM} and MAC key mK_{SC-CM}. The $\mathcal{SC}$ and $\mathcal{CM}$ can use these long-term shared keys to generate the session encryption key k_{SC-CM} and the MAC key mk_{SC-CM}. The method deployed to generate session keys is left to the sole discretion of the card manufacturer. Each $\mathcal{SC}$ has a unique identifier $\mathcal{SC}_i$ that is the identity of the smart card. To provide privacy to each smart card (and its user) the identity of the $\mathcal{SC}$ is not communicated in plaintext. Therefore, the psuedo-identifier $SC_{i'}$ is used in the ATP-1, which is generated by the $\mathcal{SC}$ and corresponding $\mathcal{CM}$ on the successful completion of the previous run of the attestation protocol. We will discuss the generation of $SC_{i'}$ and SID in subsequent messages, as the generated $SC_{i'}$ and SID during this message will be used in the next execution of the attestation protocol. A point to note is that for the very first execution of the attestation protocol, the smart card uses the pseudo-identifier ($SC_{i'}$) that was generated by the card manufacturer and stored on the smart card before the card was issued to the user. The SID is used for two purposes: firstly to authenticate the $\mathcal{SC}$ and secondly, to prevent a Denial of Service (DoS) attack on the attestation server. The $ReqVal$ is the request for attestation process.

On receipt of the first message, the $\mathcal{CM}$ will check whether it has the correct values of $SC_{i'}$ and SID. If these values are correct, it will then proceed with verifying the MAC. If satisfied, it will then decrypt the encrypted part of the message.

$$\textbf{ATP-2.} \quad \begin{aligned} \mathcal{CM} &: mE = e_{k_{SC-CM}}(CM_i||N'_{SC}||N_{CM}||Challenge) \\ \mathcal{CM} \rightarrow \mathcal{SC} &: mE||f_{mk_{SC-CM}}(mE)||SID \end{aligned}$$

The $\mathcal{CM}$ generates a random number N_{CM} and a $Challenge$. In case of the PRNG-based attestation mechanism, the $Challenge$ would also be a random number.

$$\textbf{ATP-3.} \quad \begin{aligned} \mathcal{SC} &: mE = e_{k_{SC-CM}}(N'_{SC}||N_{CM}||N_{SP}||N_{SC}||Resp||Opt) \\ \mathcal{SC} \rightarrow \mathcal{CM} &: mE||f_{mk_{SC-CM}}(mE)||SID \end{aligned}$$

After generating the $Resp$ using the PRNG-based algorithm discussed in section 2, the $\mathcal{SC}$ will proceed with message three. It will concatenate the random numbers generated by the $\mathcal{SC}$, $\mathcal{CM}$, and $\mathcal{SP}$, with the $Resp$. The rationale for including the random number from the SP in message three is to request $\mathcal{CM}$ to generate a validation message that can be independently checked by the $\mathcal{SP}$ to ensure it is fresh and valid. The function of the Opt element is to accommodate the CRP updates if other algorithms are used (e.g. PUF-based attestation).

While the $\mathcal{SC}$ was generating the $Resp$ based on the $Challenge$, the $\mathcal{CM}$ also calculates the correct attestation response. When the $\mathcal{CM}$ receives message

three, it will check the values and if they match then it will issue the validation message. Otherwise the attestation process has failed and $\mathcal{CM}$ does not issue any validation message (VM).

ATP-4.
$$\mathcal{CM} : VM = Sign_{CM}(CM_i||SC_i||N_{SP}||N_{SC}||PAC)$$
$$\mathcal{CM} : mE = e_{k_{SC-CM}}(N'_{SC}||VM||SC^{+}_{i'}||SID^{+}||CertS_{CM})$$
$$\mathcal{CM} \rightarrow \mathcal{SC} : mE||f_{mk_{SC-CM}}(mE)||SID$$

If the attestation response is successful then the $\mathcal{CM}$ will take the random numbers generated by the $\mathcal{SP}$ and the $\mathcal{SC}$ and include the identities of the $\mathcal{SC}$ and $\mathcal{CM}$. All of these items are then concatenated with the $\mathcal{SC}$'s evaluation certificate PAC and then signed by the $\mathcal{CM}$. The signed message is then communicated to the $\mathcal{SC}$.

In the ATP-4, the $\mathcal{CM}$ will also generate a SID and $SC_{i'}$ that will be used in the subsequent execution of the attestation protocol between the $\mathcal{SC}$ and $\mathcal{CM}$. The SID and $SC_{i'}$ for the subsequent run of the attestation protocol is represented as SID^{+} and $SC^{+}_{i'}$. The SID^{+} is basically a (new) random number that is associated with the pseudo-identifier of the smart card that it will use to authenticate in the subsequent attestation protocol. Furthermore, the $SC^{+}_{i'}$ is generated as $SC^{+}_{i'} = f_{mK_{CM}}(CM_i||N_{SC}||N_{CM}||SID)$, where mK_{CM} is the MAC key that the $\mathcal{CM}$ does not share.

5 Protocol Analysis

In this section, we analyse the proposed attestation protocol for given goals and provide details of the test performance results.

5.1 Informal Analysis

In order to meet the goals PG-1 and PG-2, all messages communicated between the $\mathcal{SC}$ and $\mathcal{CM}$ are encrypted and MACed using long term secret encryption and MAC keys; k_{SC-CM} and mK_{SC-CM}, respectively. The $\mathcal{A}$ has to compromise these keys in order to violate the PG-1. If we consider that the symmetric algorithm used (e.g. AES) is sufficiently strong to avert any exhaustive key search and robust enough to thwart any cryptanalysis then it is difficult for the $\mathcal{A}$ to break the protocol by attacking the used symmetric algorithms. A possibility can be to perform side-channel analysis of the smart card and attempt to retrieve the cryptographic keys; however, most modern smart cards have adequate security to prevent this attack, which are evaluated and certified by the third party evaluation (e.g. Common Criteria evaluation). Nevertheless, these assurances can only be against the state-of-the-art attack methodologies at the time of manufacturing/evaluation. Any attacks which surface after manufacture and evaluation may render both the assurance and validation mechanisms useless.

The smart card identity is not used as plaintext during the communication between the $\mathcal{SC}$ and the $\mathcal{CM}$. Instead of using the $\mathcal{SC}_i$, the $\mathcal{SC}$ uses a psuedo-identity $\mathcal{SC}_{i'}$ which changes on every successful completion of communication

with the respective $\mathcal{CM}$. Therefore, a particular $\mathcal{SC}$ will only use $\mathcal{SC}_{i'}$ once during its lifetime.

5.2 Protocol Verification by CasperFDR

The CasperFDR approach is adopted to test the soundness of the proposed protocol under the defined security properties. In this approach, the Casper compiler [14] takes a high-level description of the protocol, together with its security requirements. It then translates the description into the process algebra of Communicating Sequential Processes (CSP) [15]. The CSP description of the protocol can be machine verified using the Failures-Divergence Refinement (FDR) model checker [16]. A short introduction to the CasperFDR approach to mechanical formal analysis is provided in appendix A. The intruder's capability modelled in the Casper script (appendix A) for the proposed protocol is as: 1) An intruder can masquerade as any entity in the network. 2) It can read the messages transmitted by each entity in the network. 3) An intruder cannot influence the internal process of an agent in the network.

The security specifications for which the CasperFDR evaluates the network are as shown below. The listed specifications are defined in the # Specification section of appendix A: 1) The protocol run is fresh and both applications are alive. 2) The key generated by a smart card is known only to the card manufacturer. 3) Entities mutually authenticate each other and have mutual key assurance at the conclusion of the protocol. 4) Long term keys of communicating entities are not compromised.

The CasperFDR tool evaluated the protocol and did not find any attack(s). A point to note is that in this paper, we provide mechanical formal analysis using CasperFDR for the sake of completeness and we do not claim expertise in the mathematical base of the formal analysis.

5.3 Implementation Results & Performance Measurements

The test protocol implementation and performance measurement environment in this paper consists of a laptop with a 1.83 GHz processor, 2 GB of RAM running on Windows XP. The off-card entities execute on the laptop and for on-card entities, we have selected two distinct 16bit Java Cards referred as C1 and C2. Each implemented protocol is executed for 1000 iterations to adequately take into account the standard deviation between different protocol runs, and the time taken to complete an iteration of protocol was recorded. The test Java Cards (e.g. C1 and C2) were tested with different numbers of iterations to find out a range, which we could use as a common denominator for performance measurements in this paper. As a result, the figure of 1000 iterations was used because after 1000 iterations, the standard deviation becomes approximately uniform.

Regarding the choice of cryptographic algorithms we have selected Advance Encryption Standard (AES) [17] 128-bit key symmetric encryption with Cipher

Block Chaining (CBC) [18] without padding for both encryption and MAC operations. The signature algorithm is based on the Rivest-Shamir-Aldeman (RSA) [18] 512-bit key. We use SHA-256 [19] for hash generation. For Diffie-Hellman key generation we used a 2058-bit group with a 256-bit prime order subgroup specified in the RFC-5114 [20]. The average performance measurements in this paper is rounded up to the nearest natural number.

The attestation mechanism implemented for emulating the practical performance is based on the PRNG design. The PRNG for our experiments was based on the AES [12] and it has been implemented such that it allows us to input the seed file. The performance measures taken from two different 16-bit Java Cards are listed in table 3. The offline attestation mechanism based on PRNG take in total (excluding PRNG seed file) 2084 bytes. Similarly, the online attestation mechanism and associated attestation protocol based on PRNG take in total (excluding PRNG seed file) 5922 bytes.

Table 3. Test performance measurement (milliseconds) for the attestation protocol.

Measures	**Offline Attestation**		**Online Attestation**	
Card Specification	C1	C2	C1	C2
Average	408.63	484.55	1008	1284
Best time	367	395	930	1075
Worse time	532	638	1493	1638
Standard Deviation	41.82	59.43	87.68	92.29

5.4 Related Work

The basic concept of remote attestation and ownership acquisition came from the TCG's specifications [21]. The user takes the ownership of the Trusted Platform Module (TPM) and in return, the TPM generates a unique set of keys that are associated with the respective user. The remote attestation mechanism described in the TPM specification [22] provides a remote system attestation (only software). The attestation mechanism is designed so that if the software state is modified, the TPM cannot generate a valid report.

The TPM does not provide an attestation that includes the hardware state. Furthermore, the attestation defined in the TPM specification is more like the offline attestation. However, the offline attestation mechanism (algorithm 1) is different to the one used by TPM, whereas the online attestation is not part of the TPM specifications.

Similarly, other proposals concentrate on the software attestation without binding it to a particular hardware. Such proposals include SCUBA [23], SBAP [24], and SWATT [25]. These protocols utilise execution time as a parameter in the attestation process. This is difficult to guarantee remotely, even with the delegation of time measurement to neighbouring trustworthy nodes [23]. Other

mechanisms that use trusted hardware are proposed by Schellekens et al. [26] and PUF-based protocols [27].

There is no such proposal for remote attestation in smart card frameworks like Java Card, Multos, or GlobalPlatform. The nearest thing is the DAP in the GlobalPlatform card specification that checks the signature on the downloaded application (if the application provider chooses this option). Furthermore, we have opted out of having execution measurement as part of the attestation process as it is difficult to ascertain the trustworthiness of the remote device that measures it. However, unlike other proposed protocols we have an explicit requirement that third party evaluation is used to provide an implicit trust in the attestation process. Furthermore, our proposal binds the software attestation with the hardware protection (tamper-evident) mechanism to provide added assurance.

6 Conclusion

In this paper, we briefly discussed the generic architecture of the UCSC and its components. Later, we extended the discussion to the security assurance and validation framework that requires a third party evaluation and an attestation process. The attestation process includes hardware validation with the traditional software attestation. We proposed two modes for the attestation process: offline and online attestation. In designing the attestation processes, we based our proposal on the PRNG algorithms. To have an online attestation, we proposed the attestation protocol that communicates with the card manufacturer to get a dynamic certificate of assurance (a signed message from the card manufacturer) that the smart card is still secure and reliable. We implemented offline and online attestation mechanisms, along with an attestation protocol on 16-bit Java Cards. We also detailed the performance measurements of the implemented mechanisms and protocols.

References

1. R. N. Akram, K. Markantonakis, and K. Mayes, "A Paradigm Shift in Smart Card Ownership Model," in *Proceedings of the 2010 Intl. Conf. on Computational Science and Its Applications (ICCSA 2010)*, B. O. Apduhan, O. Gervasi, A. Iglesias, D. Taniar, and M. Gavrilova, Eds. Fukuoka, Japan: IEEE Computer Society, March 2010, pp. 191–200.
2. R. N. Akram, K. Markantonakis, and K. Mayes, "A Dynamic and Ubiquitous Smart Card Security Assurance and Validation Mechanism," in *25th IFIP Intl. Information Security Conf. (SEC 2010)*, ser. IFIP AICT Series, K. Rannenberg and V. Varadharajan, Eds. Brisbane, Australia: Springer, Sep 2010, pp. 161–172.
3. R. N. Akram, K. Markantonakis, and K. Mayes, "Coopetitive Architecture to Support a Dynamic and Scalable NFC based Mobile Services Architecture," in *The 2012 Intl. Conf. on Information and Communications Security (ICICS 2012)*, K. Chow and L. C. Hui, Eds. Hong Kong, China: Springer, October 2012.

4. "The GlobalPlatform Proposition for NFC Mobile: Secure Element Management and Messaging," GlobalPlatform, White Paper, April 2009.
5. *GlobalPlatform: GlobalPlatform Card Specification, Version 2.2,* March 2006.
6. K. Eagles, K. Markantonakis, and K. Mayes, "A comparative analysis of common threats, vulnerabilities, attacks and countermeasures within smart card and wireless sensor network node technologies," in *the 1st IFIP Intl. Conf. on Information Security Theory and Practices*, Berlin: Springer, 2007, pp. 161–174.
7. *FIPS 140-2: Security Requirements for Cryptographic Modules*, Online, National Institute of Standards and Technology (NIST) Federal Information Processing Standards Publication, Rev. Supercedes FIPS PUB 140-1, May 2005.
8. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the 9th ACM Conf. on Computer and communications security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 148–160.
9. P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-proof Hardware from Protective Coatings," in *Cryptographic Hardware and Embedded Systems Workshop*, ser. LNCS, Springer, Oct 2006, pp. 369–383.
10. H. Busch, M. Sotáková, S. Katzenbeisser, and R. Sion, "The PUF promise," in *the 3rd Intl. Conf. on Trust and Trustworthy Computing.* Springer, June 2010.
11. D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-Channel Analysis of PUFs and Fuzzy Extractors," in *Trust and Trustworthy Computing*, J. McCune, B. Balacheff, A. Perrig, A.-R. Sadeghi, A. Sasse, and Y. Beres, Eds. Springer, 2011.
12. R. N. Akram, K. Markantonakis, and K. Mayes, "Pseudorandom Number Generation in Smart Cards: An Implementation, Performance and Randomness Analysis," in *5th Intl. Conf. on New Technologies, Mobility and Security (NTMS)*, A. Mana and M. Klonowski, Eds. Istanbul, Turkey: IEEE CS, May 2012.
13. W. Rankl and W. Effing, *Smart Card Handbook*, 3rd ed. NY, USA: John Wiley & Sons, Inc., 2003.
14. G. Lowe, "Casper: a compiler for the analysis of security protocols," *J. Comput. Secur.*, vol. 6, pp. 53–84, January 1998.
15. C. A. R. Hoare, *Communicating sequential processes.* New York, NY, USA: ACM, 1978, vol. 21, no. 8.
16. P. Ryan and S. Schneider, *The Modelling and Analysis of Security Protocols: the CSP Approach.* Addison-Wesley Professional, 2000.
17. Joan Daemen and Vincent Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard.* Berlin, Heidelberg, New York: Springer Verlag, 2002.
18. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC, October 1996.
19. *FIPS 180-2: Secure Hash Standard (SHS)*, National Institute of Standards and Technology (NIST) Std., 2002.
20. M. Lepinski and S. Kent, "RFC 5114 - Additional Diffie-Hellman Groups for Use with IETF Standards," Tech. Rep., January 2008.
21. "Trusted Computing Group, TCG Specification Architecture Overview," The Trusted Computing Group (TCG), Oregon, USA, revision 1.4, August 2007.
22. *Trusted Module Specification 1.2: Part 1- Design Principles, Part 2- Structures of the TPM, Part 3- Commands*, TCG Std., Rev. 103, July 2007.
23. A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure Code Update By Attestation in sensor networks," in *Proceedings of the 5th ACM workshop on Wireless security*, ser. WiSe '06. NY, USA: ACM, 2006, pp. 85–94.
24. Y. Li, J. M. McCune, and A. Perrig, "SBAP: software-based attestation for peripherals," in *Proceedings of the 3rd Intl. Conf. on Trust and trustworthy computing*, ser. TRUST'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 16–29.

25. A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: SoftWare-based ATTestation for Embedded Devices," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 272, 2004.
26. D. Schellekens, B. Wyseur, and B. Preneel, "Remote attestation on legacy operating systems with trusted platform modules," *Sci. Comput. Program.*, vol. 74, pp. 13–22, December 2008.
27. S. Schulz, C. Wachsmann, and A.-R. Sadeghis, "Lightweight Remote Attestation using Physical Functions," Technische Universitat Darmstadt, Darmstadt, Germany, Technical Report, July 2011.

A Attestation Protocol

The Casper script in this section corresponds to the attestation protocol described in section 4.

```
#Free variables
SC, CM : Agent
ns, nsp, nt, c, r : Nonce
S1, S2 : Num
VKey: Agent -> PublicKey
SKey: Agent -> SecretKey
InverseKeys = (sKey, sKey), (VKey,
SKey)

#Protocol description
0.  -> SC : CM
1. SC -> CM : S1,{SC, ns, CM,}{sKey}
2. CM -> SC : {CM, ns, nm, c,
S2}{sKey}
3. SC -> CM : {ns,nm,nsp,r}{sKey}
4. CM -> SC : {ns,{CM,SC,ns,nsp}
               {Skey{CM}}}{sKey}

#Actual variables
SmartCard, CardManufacturer, MAppl
: Agent
Ns, Nsp, Nt, Nm, Challenge, Response
: Nonce
SOne, STwo : Num

#Processes
INITIATOR(SC, CM, ns, nsp, r) knows
sKey, VKey
RESPONDER(CM, SC, nm, c) knows
sKey, SKey(CM), VKey

#System
INITIATOR(SmartCard, CardManufacturer,
Ns, Nsp, Response)
RESPONDER(CardManufacturer, SmartCard,
Nm, Challenge)

#Functions
symbolic VKey, SKey

#Intruder Information
Intruder = MAppl
IntruderKnowledge = {SmartCard,
CardManufacturer, MAppl, MAppl, Nm,
Nsp, SKey(MAppl), VKey}

#Specification
StrongSecret(SC, sKey, [CM])
StrongSecret(SC, r, [CM])
Aliveness(SC, CM)
Aliveness(CM, SC)
```