Remote Attestation Mechanism for Embedded Devices based on Physical Unclonable Functions

Raja Naeem AKRAM [a] Konstantinos MARKANTONAKIS [b] Keith MAYES [b]
[a] *Cyber Security Lab, Department of Computer Science, University of Waikato, Hamilton, New Zealand. Email: rnakram@waikato.ac.nz*
[b] *ISG Smart Card Centre, Royal Holloway, University of London, Egham, United Kingdom. Email: k.markantonakis, keith.mayes@rhul.ac.uk*

Abstract. Remote attestation mechanisms are well studied in the high-end computing environments; however, the same is not true for embedded devices - especially for smart cards. With ever changing landscape of smart card technology and advancements towards a true multi-application platform, verifying the current state of the smart card is significant to the overall security of such proposals. The initiatives proposed by GlobalPlatform Consumer Centric Model (GP-CCM) and User Centric Smart Card Ownership Model (UCOM) enables a user to download any application as she desire - depending upon the authorisation of the application provider. Before an application provider issues an application to a smart card, verifying the current state of the smart card is crucial to the security of the respective application. In this paper, we analyse the rationale behind the remote attestation mechanism for smart cards, and the fundamental features that such a mechanism should possess. We also study the applicability of Physical Unclonable Functions (PUFs) for the remote attestation mechanism and propose two algorithms to achieve the stated features of remote attestation. The proposed algorithms are implemented in a test environment to evaluate their performance.

1. Introduction

Fundamentally, both the GlobalPlatform Consumer Centric Model (GP-CCM) [1] and User Centric Smart Card Ownership Model (UCOM) [2] are similar in a sense that they both advocate for the user's "*freedom of choice*" – the users can install or delete any application as they please on their smart cards. In this paper, we focus on the UCOM; however, the proposed solutions are also applicable to the GP-CCM. The UCOM requires that smart cards must have 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 overall UCOM ecosystem.

In the UCOM, the card manufacturers make sure that smart cards have adequate security and operational functionality (i.e. firewall, application sharing, tamper-evidence and secure execution environment etc.) to support the user ownership. 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 [3]. This assurance will be achieved through a third party security evaluation of the smart cards before they are issued to individual users [4]. Furthermore, to provide a dynamic security validation [4], 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 briefly describe the core architecture of the remote attestation mechanism followed by a discussion regarding the fundamental design requirements for such a mechanism. Subsequently, we describe the proposed remote attestation mechanism, and two algorithms based on PUFs. Finally, we discuss an attestation protocol for the proposed remote attestation mechanism and present the implementation details and performance measurements.

Section 2, discusses the core architecture of the attestation mechanism that provides security and reliability assurance to (remote) requesting entities. Subsequently, we extend the discussion to the remote attestation framework and the proposed algorithms in section 3. In section 4 we propose an attestation protocol; in section 5 we detail the test implementation results of the attestation protocol and proposed algorithms.

2. Attestation Mechanism Framework

In this section, we discuss the core architecture of the attestation mechanism followed by the discussion on the design requirements and possible solutions.

2.1. The Core Architecture for the Attestation Mechanism

On a typical smart card, several mechanisms are in place to test and verify the state of the platform (both software and hardware). At the software level, GlobalPlatform card specification has proposed the controlling authority (termed CA in the GlobalPlatform card specification) [5] and the Mandated Data Authentication Pattern (Mandated DAP) mechanism [5,6]. In the DAP mechanism, an off-card entity (controlling authority) signs applications that are being loaded onto a smart card, and this approval of the applications is verified by an on-card entity referred to as the GlobalPlatform card manager [6]. At the hardware level, the Known Answer Test (KAT) for cryptographic modules mandated by FIPS [7] and similar mechanism are deployed by the smart card manufacturers (i.e. RAM test, and checksum of non-volatile memory) [8].

However, there is no proposal for remote attestation of the smart card and applications installed on it. The UCOM proposes the Trusted Environment & Execution Manager (TEM) as a trusted module for embedded devices like smart cards. The TEM is fundamentally different from the Trusted Platform Module (TPM) [9] and Mobile Trusted Module (MTM) [10] in two respects. Firstly, the TEM implements a self-test mechanism that includes hardware parameters to provide remote attestation and a dynamically configurable integrity measurement mechanism that is based on a challenge-response framework. Secondly, the TEM is not based on a static architecture; in fact, it enforces platform security policies during the application execution rather than just generating the hash (once) at the start of the application execution.

The concept of TEM is to group/provide similar and enhanced functionality that provides assurance and validation of the platform to requesting on-card or off-card entities. The TEM is independent of the platform configuration that is mainly concerned with the smart card runtime environment, which can be based on a technology such as Java Card or Multos. A detailed discussion on the TEM, its features and comparison with other trusted modules (i.e. TPM, and MTM) is beyond the scope of this paper.

The proposed remote attestation mechanism is based on the following two TEM features; the attestation handler and the self-test manager, which are the core components of the attestation mechanism and are discussed in the following sections.

2.2. The Attestation Handler

The attestation handler and the self-test manager are part of core architecture and the difference between these two modules is that the attestation handler focuses on the software and the self-test manager 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.

During the attestation mechanism, 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 or requesting entity that the platform is as secure and reliable as it is claimed to be in the security evaluation certificate discussed in [4]. In addition, respective SPs can ask the TEM to generate the state validation of their applications (e.g. a signed hash of the application) after they have been installed, ensuring that the application is downloaded without any errors. This function of the attestation handler is similar to the Data Authentication Pattern (DAP) [5,6].

2.3. Self-test Manager

The self-test mechanism checks whether the smart card is tamper-resistant as certified by a security evaluation certificate [4]. The aim of the self-test mechanism is to provide a remote hardware validation in a way that enables a requesting entity (e.g. an SP) to independently verify that the smart card tamper-resistance mechanism is still secure and reliable. As our focus not at the hardware end of the smart card technology, we do not propose any hardware-based mechanism in this paper, which is one of the possible directions for future research.

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

1. Robustness: On input of certain (random) data, it should always produce associated (random) output in an efficient manner. If on input of 'i' it generates 'z', the next time 'i' is used as input the output should be the same 'z'.
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 sefl-test function should cause irreversible changes which render the device unusable/inaccessible.
5. Unforgeable: It should be computationally difficult to simulatethe self-test mechanism.
6. Assurance: the self-test mechanism should provide assurance (either implicitly or explicitly) to independent verifiers. It should not require an active connection with the device manufacturer to provide the assurance.

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

Features	**Active-Shield**	**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 support this feature if adequately catered for during the design.

There are several possibilities for a self-test mechanism for smart cards including using active (intelligent) shield/mesh [11], the Known Answer Test (KAT) [7], and the Physical Unclonable Function (PUF) [12].

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 manufacturer (or associated partners).

Another potential protection strategy is to utilise Physical Unclonable Functions (PUFs) [12] to provide hardware validation. It is difficult to find a single and consistent definition of PUF in the literature [13]. Usual applications of the PUF described in the literature are in anti-counterfeiting [14], Intellectual Property protection [15], tamper-evident hardware [16], hardware based cryptography [17] and secure/trusted processors [18].

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 that makes it difficult to be different on individual smart cards of the same batch. It also requires other features to provide tamper evidence, like active-shield. Whereas, PUFs and adequately designed Pseudorandom Number Generators (PRNGs) can provide assurance that the platform state and the tamper-resistant protections of a smart card are still active.

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 the industry [19]; for completeness, we use them as a self-test mechanism in our proposals because they meet most of the requirements listed in table 1.

3. Attestation Mechanisms

In this section, we discuss the two attestation mechanisms based on non-simulatable PUFs that combines the functionality of the attestation handler and self-test manager discussed in previous section.

3.1. Non-simulatable PUFs

A non-simulatable PUF is a PUF that is computationally difficult to simulate including the device manufacturer, user (device owner) and malicious entities. This has made PUF a suitable candidate for the true/pseudo random number and secret key generators [20,21].

Algorithm 1: Self-test algorithm for offline attestation based on a PUF

```
Input  : $l$; list (array) 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 $k$.
Notation:
x ⟵ y+z: first the operation on the right of the arrow will be performed and the
result will be stored in $x$. This notation is common for all algorithms in this paper.
1  SelfTestOffline ($l$) begin
2      while $i < n$ do
3          $a \longleftarrow$ ReadAddressList ($l$,$i$)
4          $seed \longleftarrow$ Hash (ReadMemoryContents ($a$), $seed$)
5          $i \longleftarrow i+1$
6      if $seed \neq \emptyset$ then
7          $k \longleftarrow$ nmPUF ($seed$)
8      else
9          return testfailed
10     $S \longleftarrow$ DecryptionFunction ($k$, $S_e$)
11     return S
```

Based on non-simulatable PUFs, we describe two algorithms 1 and 2 that take into account the offline and online modes of the attestation mechanism. In the offline mode, the communication is only between the requesting entity (i.e., application provider) and the respective smart card. In the online mode, the card manufacturer or management authority is included in the attestation mechanism.

The offline algorithm is based on the function `SelfTestOffline` that takes a list of selected memory addresses (l) stored on the card by the card manufacturer. The list has memory addresses of critical components related to the security and reliability of the given smart card platform. The function `SelfTestOffline` iterates through the 'l' and generates a hash of the contents of the given memory location. The generated hash value is then stored as a *seed*. After traversing through the 'l', the `SelfTestOffline` checks the value of the *seed*. If the seed value is zero then throw an attestation fail exception; otherwise, proceed to the next step. The generated *seed* value is then input to the PUF that produces a sequence referred as 'k' in algorithm 1. Using the generated 'k', the `SelfTestOffline` will decrypt the signature key for the given device, then return the signature key to the attestation handler. The handler will generate a signature and send it to the requesting entity (e.g. the SP) along with the relevant cryptographic certificate. If the signature is verified correctly then the smart card state is in conformance to the evaluation state.

The PUF based secret key and associated public key pair and certificate is generated at the time of card manufacturing. The private key is certified by the evaluation authority that also provides the security and reliability details of their evaluation as part of the certificate [4]. The certificate hierarchy and associated keys in a smart card are discussed in section 3.2.

Algorithm 2: Self-test algorithm for online attestation based on a PUF

```
Input     :
$c$; challenge sent by the card manufacturer.
$n$; random number send by the card manufacturer.
Output  :
$r$; hash value generated on selected memory addresses, set at zero.
$p$; response part of the CRP for the implemented PUF.
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 $seedfile$.
$s$; unique reference to an entry in the $seedfile$.
$nc$; number of bytes in the $n$.
$i$; counter set to zero.
$l$; upper limit of memory address defined by the card manufacturer.
$m$; memory address.
$mK$; shared secret between a smart card and respective card manufacturer.
Notation:
x % y: represents x modulus y. This notation is common for all algorithms in this paper.

SelfTestOnline ($c, n$) begin
    $mK \longleftarrow$ nmPUF($c$)
    while $i < nc$ do
        $s \longleftarrow$ ReadSingleByte($n, i$) % $ns$
        $seed \longleftarrow$ ReadSeedFile($s$)
        $m \longleftarrow$ GenPRNG($seed$) % $l$
        $r \longleftarrow$ Hash(ReadMemoryContents($m$), $r, mK$)
        if $(nc - i) = 1$ then
            $p \longleftarrow$ nmPUF($r$)
        $i \longleftarrow i+1$
    return $r$, $p$
```

For online attestation, the card manufacturers will have to generate (limited) Challenge-Response Pairs (CRPs), which will be unique to the individual device. The rationale behind this is based on the design of a non-simulatable PUF in which the designer tries to make the CRP space sufficiently large to make it difficult for an adversary to simulate the PUF [22,20]. This design decision even makes it difficult for the card manufacturer to simulate the PUF. The limited set of generated CRPs will lead to a limited number of device validations (before they start to repeat), which is not a scalable solution. Therefore, we use a rolling update mechanism in which at the end of each successful device validation (section 4) a new CRP will be generated for future use. A valid CRP response can also help the card manufacturer to ascertain that the device is not counterfeit as only the issued device's CRPs are registered in its CRP database.

The PUF-based online attestation mechanism represented in algorithm 2 implements a function `SelfTestOnline` that takes two parameters: a challenge 'c' and random number 'n' from the respective card manufacturer. The challenge 'c' is input to the PUF at line two and a response is generated, which is the response to the challenge 'c' and we treat it as a shared secret (mK). The function `SelfTestOnline` then treats the random number 'n' as a collection of bytes, reading one byte at a time and taking modulus of the byte with the length of the $seedfile$. By doing so, we generate an index to the $seedfile$ and in the next step we read a $seed$ value from that index. The $seed$ value is used to generate a new random number, whose modulus with upper memory limit (l) defined by the manufacturer gives us a memory location. In the next step (line seven), we read and hash the memory contents from the memory location, and the result is stored in 'r'. This process is repeated for the number of bytes the random number 'n' has, which is represented by the nc. At $nc-1$ iteration, the value of "r_{nc-1}" (the value of r at the iteration "$nc-1$") is used as input to the PUF again to generate a new CRP.

In function `SelfTestOnline`, the generated 'r' and 'p' are then securely communicated back to the smart card manufacturer, which can verify the generated 'r' and stores the CRP. The card manufacturer can verify the 'r' by executing instructions from lines three to seven of the algorithm 2. The function `SelfTestOnline` does not send the challenge ("r_{nc-1}") which was used to generate the response 'p' because the card manufacturer can also generate the value of 'r' at iteration "$ns-1$".

3.2. Key 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 [8] the card manufacturer initiates the key generation process.

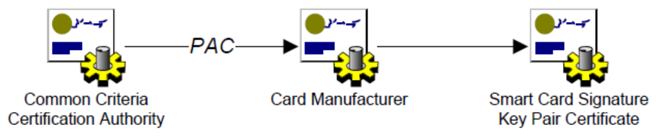
PAC
Common Criteria Certification Authority
Card Manufacturer
Smart Card Signature Key Pair Certificate

Figure 1. Certificate hierarchy in the UCOM.

Each smart card will generate a signature key pair (SC_{Sign}) that does not change for the lifetime of the smart card. The SC_{Sign} is certified by the card manufacturer, and it is used to provide offline attestation. Furthermore, in the certificate hierarchy shown in figure 1, the SC_{Sign} is linked with the Platform Assurance Certificate (PAC) [4] via the card manufacturer's certificate. The PAC is a cryptographic certificate that certifies the card manufacturer's key for the particular batch of smart cards that are evaluated by a third party like Common Criteria (CC) for their security and reliability. At present, such certificates are not issued however, the framework proposed in [4] can be adopted for such purposes.

As discussed in section 3.1, the evaluation authority issues a certificate (e.g. PAC) 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.

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 for generating MAC communication messages between the smart card and the card manufacturer.

4. Attestation Protocol

The Attestation Protocol (ATP), involves the card manufacturer in 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) 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. 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).

4.2. Protocol Notation and Terminology

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

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 a MAC algorithm on data Z with key K.
$Sign_X(Z)$	Represents 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.
$CertS_{X\leftarrow Y}$	Represents the certificate for the signature key belonging to an entity X, issued by an entity Y.
$CertE_{X\leftarrow Y}$	Certificate for the public encryption key belonging to an entity X, issued by an entity Y.
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 [4].
$X \rightarrow Y : C$	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).

4.3. Protocol Description

In this section, we describe the attestation protocol, and each message is represented by *ATP-n*, where n represents the message number. The structure of this representation would be the protocol acronym (i.e. ATP for attestation protocol) followed by the message number.

Table 3. Messages in the Proposed Protocol.

ATP-1.	$\mathcal{SC}$	:	$mE = e_{k_{SC-CM}}(SC_i \| N'_{SC} \| CM_i \| ReqVal)$
	$\mathcal{SC} \rightarrow \mathcal{CM}$	:	$SC_{i'} \| mE \| f_{mk_{SC-CM}}(mE) \| SID$
ATP-2.	$\mathcal{CM}$	:	$mE = e_{k_{SC-CM}}(CM_i \| N'_{SC} \| N_{CM} \| Challenge)$
	$\mathcal{CM} \rightarrow \mathcal{SC}$	:	$mE \| f_{mk_{SC-CM}}(mE) \| SID$
ATP-3.	$\mathcal{SC}$	:	$mE = e_{k_{SC-CM}}(N'_{SC} \| N_{CM} \| N_{SP} \| N_{SC} \| Response \| Optional)$
	$\mathcal{SC} \rightarrow \mathcal{CM}$	:	$mE \| f_{mk_{SC-CM}}(mE) \| SID$
ATP-4.	$\mathcal{CM}$	:	$VM = Sign_{CM}(CM_i \| SC_i \| N_{SP} \| N_{SC} \| PAC)$
	$\mathcal{CM}$	:	$mE = e_{k_{SC-CM}}(N'_{SC} \| VM \| SC^{+}_{i'} \| SID^{+} \| CertS_{CM})$
	$\mathcal{CM} \rightarrow \mathcal{SC}$	:	$mE \| f_{mk_{SC-CM}}(mE) \| SID$

ATP-1 Before issuing the smart card to the user, the $\mathcal{SC}$ and $\mathcal{CM}$ will establish two secret keys; an encryption key K_{SC-CM} and a 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 pseudo-identifier $SC_{i'}$ is used in the ATP-1, which is generated by the $\mathcal{SC}$ and the 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 the MAC is valid, it will then decrypt the encrypted part of the message.

ATP-2 After the message is successfully decrypted, 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; however, in case of PUF-based attestation mechanism it would be the pre-calculated challenge part of the CRP.

ATP-3 After generating the $Response$ using the algorithms discussed in section 3, 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 $Response$. 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 $Optional$ element is to accommodate the CRP updates if the $\mathcal{CM}$ implements a PUF-based attestation process.

While the $\mathcal{SC}$ was generating the $Response$ 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 If the attestation response is successful then the $\mathcal{CM}$ will take the random numbers generated by the $\mathcal{SP}$ and the $\mathcal{SC}$ during the Secure and Trusted Channel Protocols (STCPs) discussed in paper [23] 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 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 be used to authenticate in the subsequent attestation protocol runs. 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 detail the test performance results.

5.1. 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) [27] 128-bit key symmetric encryption with Cipher Block Chaining (CBC) [28] without padding for both encryption and MAC operations. The signature algorithm is based on the Rivest-Shamir-Aldeman (RSA) [28] 512-bit key. We used SHA-256 [29] 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 [30]. The average performance measurements in this paper is rounded up to the nearest natural number.

The attestation mechanism implemented in our test environment executes all the instructions on a pair of Java Cards, except for the PUF whose execution time from [22] was added later. The performance measurement taken from two different 16-bit Java Cards are listed in table 4. The offline attestation mechanism based on PUF takes in total 2292 bytes of memory space. Similarly, the online attestation mechanism and associated attestation protocol based on PUF takes in total 6392 bytes.

Table 4. Test performance measurement (milliseconds) for the attestation protocol.

Measures	**Offline Attestation**		**Online Attestation**	
Card Specification	C1	C2	C1	C2
Average	532.75	584.26	1128.65	1284.85
Best time	506	495	992	1075
Worse time	749	838	1312	1638
Standard Deviation	53.22	83.31	103.62	112.72

6. Summary

In this paper, we discussed the overall architecture of the attestation mechanism that includes hardware validation with the traditional software attestation. We proposed two modes for the attestation process: offline and online attestation. In designing the attestation process, we based our proposal on PUFs. To have an online attestation, we proposed the attestation protocol that communicates with the card manufacturer to get a dynamic validation of assurance (i.e., 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 and detailed the performance measurements.

References

[1] "GlobalPlatform A New Model: The Consumer-Centric Model and How It Applies to the Mobile Ecosystem," GlobalPlatform, Whitepaper, March 2013.

[2] R. N. Akram, K. Markantonakis, and K. Mayes, "A Paradigm Shift in Smart Card Ownership Model," in *the 2010 Int. Conf. on Computational Science and Its Applications (ICCSA 2010)*, B. O. Apduhan and O. Gervasi, Eds. Fukuoka, Japan: IEEE CS, March 2010, pp. 191–200.

[3] R. N. Akram, K. Markantonakis, and K. Mayes, "Application Management Framework in User Centric Smart Card Ownership Model," in *The 10th Int. Workshop on Information Security Applications (WISA09)*, H. Y. YOUM and M. Yung, Eds., Busan, Korea: Springer, August 2009, pp. 20–35.

[4] R. N. Akram, K. Markantonakis, and K. Mayes, "A Dynamic and Ubiquitous Smart Card Security Assurance and Validation Mechanism," in *25th IFIP Int. Information Security Conf. (SEC 2010)*, K. Rannenberg and V. Varadharajan, Eds. Brisbane, Australia: Springer, September 2010, pp. 161–172.

[5] "The GlobalPlatform Proposition for NFC Mobile: Secure Element Management and Messaging," GlobalPlatform, White Paper, April 2009.

[6] *GlobalPlatform: GlobalPlatform Card Specification, Version 2.2,*, GlobalPlatform Std., March 2006.

[7] *FIPS 140-2: Security Requirements for Cryptographic Modules*, Online, NIST Federal Information Processing Standards Publication, Rev. Supercedes FIPS PUB 140-1, May 2005.

[8] W. Rankl and W. Effing, *Smart Card Handbook*, 3rd ed. NY, USA: John Wiley & Sons, Inc., 2003.

[9] *Trusted Module Specification 1.2: Part 1- Design Principles, Part 2- Structures of the TPM, Part 3- Commands*, Trusted Computing Group Std., Rev. 103, July 2007.

[10] "TCG Mobile Trusted Module Specification," Trusted Computing Group (TCG), V1.0, June 2008.

[11] 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 Int. Conf. on Information Security Theory and Practices*, ser. WISTP'07. Springer, 2007, pp. 161–174.

[12] 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*, NY, USA: ACM, 2002, pp. 148–160.

[13] H. Busch, M. Sotáková, S. Katzenbeisser, and R. Sion, "The PUF Promise," in *Proceedings of the 3rd Int. Conf. on Trust and Trustworthy Computing.* Springer, June 2010, pp. 290–297.

[14] D. Kirovski, "Anti-Counterfeiting: Mixing the Physical and the Digital World," in *Foundations for Forgery-Resilient Cryptographic Hardware*, ser. Dagstuhl Seminar Proceedings, J. Guajardo, B. Preneel, A.-R. Sadeghi, and P. Tuyls, Eds., Germany, 2010.

[15] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended Abstract: The Butterfly PUF Protecting IP on every FPGA," in *Proceedings of the 2008 IEEE Int. Workshop on Hardware-Oriented Security and Trust.* Washington, DC, USA: IEEE CS, 2008, pp. 67–70.

[16] 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, vol. 4249. Springer, October 2006, pp. 369–383.

[17] R. S. Pappu, "Physical One-way Functions," Ph.D. dissertation, MIT, March 2001.

[18] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas, "Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Function," vol. 33, pp. 25–36, May 2005.

[19] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-Channel Analysis of PUFs and Fuzzy Extractors," in *Trust and Trustworthy Computing*, ser. LNCS, J. McCune, Eds. Springer, 2011, vol. 6740, pp. 33–47.

[20] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *the 44th Annual Design Automation Conf.*, NY, USA: ACM, 2007, pp. 9–14.

[21] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical Unclonable Function and True Random Number Generator: a Compact and Scalable Implementation," in *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, ser. GLSVLSI '09. New York, NY, USA: ACM, 2009, pp. 425–428.

[22] S. Schulz, C. Wachsmann, and A.-R. Sadeghis, "Lightweight Remote Attestation using Physical Functions," Technische Universitat Darmstadt, Darmstadt, Germany, TR-2001-06-11, July 2011.

[23] R. N. Akram, K. Markantonakis, and K. Mayes, "A Secure and Trusted Channel Protocol for the User Centric Smart Card Ownership Model," in *12th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-13).* Australia: IEEE CS, 2013.

[24] G. Lowe, "Casper: a compiler for the analysis of security protocols," *J. Comput. Secur.*, vol. 6, Jan 1998.

[25] C. A. R. Hoare, *Communicating Sequential Processes.* NY, USA: ACM, 1978, vol. 21, no. 8.

[26] P. Ryan and S. Schneider, *The Modelling and Analysis of Security Protocols: the CSP Approach.* Addison-Wesley Professional, 2000.

[27] Joan Daemen and Vincent Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard.* Berlin: Springer, 2002.

[28] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC, October 1996.

[29] *FIPS 180-2: Secure Hash Standard (SHS)*, NIST Spec., 2002.

[30] M. Lepinski and S. Kent, "RFC 5114 - Additional Diffie-Hellman Groups for Use with IETF Standards," Tech. Rep., January 2008.

[31] "Trusted Computing Group, TCG Specification Architecture Overview," The Trusted Computing Group (TCG), Beaverton, Oregon, USA, revision 1.4, August 2007.

[32] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure Code Update by Attestation in Sensor Networks," in *the 5th ACM workshop on Wireless security*, NY, USA: ACM, 2006, pp. 85–94.

[33] Y. Li, J. M. McCune, and A. Perrig, "SBAP: Software-based Attestation for Peripherals," in *Proceedings of the 3rd Int. Conference on Trust and Trustworthy Computing*, Berlin, Springer, 2010, pp. 16–29.

[34] 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.

[35] 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.

[36] H. Busch, S. Katzenbeisser, and P. Baecher, "PUF-Based Authentication Protocols - Revisited," in *Information Security Applications*, ser. LNCS, H. Youm and M. Yung, Eds. Springer, 2009, pp. 296–308.