Provisioning Software with Hardware-Software Binding

Robert P. Lee, Konstantinos Markantonakis, Raja Naeem Akram

robert.lee.2013@live.rhul.ac.uk,k.markantonakis@rhul.ac.uk,r.n.akram@rhul.ac.uk

ISG Smart Card and IoT Security Centre

Royal Holloway

UK

ABSTRACT

Smart cities are a concept of interest to many industrial, academic and government organisations. However, smart cities present a large attack surface to adversaries if every traffic light, power relay and water pipe are connected to the internet. This paper describes the problem of distributing software in a smart city when strong protection of device software, software installation and update provision are required. A set of requirements for a secure software provisioning system is presented and two models for the software distribution are proposed. Three protocols for distributing software are presented that meet the requirements stated. A formal analysis using Tamarin Prover is described that proves the security of the proposed protocols. Finally, an implementation has been developed using a laptop and Raspberry Pi 3 to demonstrate the proposed protocols in action and the performance of them.

1 INTRODUCTION

In this paper we are attempting to address the problem of securely provisioning software to devices in a "Smart City". "Smart Cities" are a concept that of interest to many groups including local and national governments, political unions and academics [3, 4, 6]. Smart cities are an approach to urban development that includes smart concepts and technologies to provide more effective services to the people who live or work in the city.

The UK department for Business Innovation and Skills (BIS) lists the following services that they wish to enhance in smart cities: Intelligent transport systems, Assisted or indepenent living, Water Management, Smart grids or energy networks and Waste Management [3]. These areas are all part of critical infrastructure so their availability and security are vital. Frequently, government systems such as healthcare, energy grids, water grids and other critical infrastructure systems transmit sensitive information that must be kept confidential. Transport management systems must also be secured because problems can cause significant delays or even physical harm to people. For example, interfering with traffic lights could easily cause an accident. With many risks in the smart city setting it is highly important that the networkeed devices be secured. In these settings it is critical devices behave as expected. Protecting devices from software tampering ensures they behave as designed. However, smart cities comprise many types of devices in a wide area that require software and software updates to be provisioned in the field instead of relying on secure manufacturing to prevent software compromise. A further complication is that the hardware may not be managed by the manufacturer of it but by a third party contracted by the organisation that owns/manages them.

The smart cities model can be abstracted into a theoretical model that also applies to other scenarios. A smart city can be considered as a wide network of computing devices that perform tasks and regularly communicate with a central server/s. In many large networks it may be impractical to rely on physical access to devices to change or update software. However, it is often important to ensure software can only be loaded by authorised parties. Examples of such networks include: industrial control systems, sensor networks and vehicular networks.

1.1 Contributions

The main contributions of this paper are:

(1) Modelling the security of provisioning software to large numbers of remote hardware devices (Section 2).
(2) Proposing a set of requirements for securing software provisioning (Section 2.3).
(3) Discussing the different potential models for managing the security of software provisioning (Section 3).
(4) Proposing a protocol to solve the problem of software provisioning (Section 5).
(5) Providing a security verification of the proposed protocol performed using Tamarin Prover (Section 6).
(6) Benchmarking the performance of the proposed schemes to show the performance of the proposals (Section 7).

1.2 Paper Structure

The paper is structured as follows. Section 2 contains an overview of the problem considered in this paper. Section 3 explores the possible models that could be used for software provisioning and draws the security requirements for a solution to the problem considered. Section 4 describes the related work in this area. Section 5 contains the solution proposed for securely provisioning software and updates to a network of devices. Section 6 analyses the solution proposed against the requirements set out in Section 3. Section 7 describes the implementations of the proposed protocols and includes performance data. Section 8 concludes the paper and describes the future work required in this area.

2 PROBLEM DESCRIPTION

This work considers how to securely provision software to devices in a wide, smart city network. This work will consider a simplified

problem including just one entity attempting to install software onto devices in a network belonging to a particular owner. This section explores the motivations of the entities involved and describes the attacker who seeks to undermine the security of the system. Finally, this section lists and describes the requirements for solving the problem posed.

2.1 Entities and Motivations

Provisioning software to devices includes four individuals: the Device (D), the Operator (O), the Manufacturer (M) and City Hall (CH). D wants access to an application provided by O. O is responsible for operating the Devices and is interested in loading their application onto them. However, O wants to control the spread of their application. M is a manufacturer of Devices who wants to create Ds able to securely receive and store software. CH is the organisation that owns the Devices and has contracted their operation to O.

The Device (D) is a generic Internet of Things (IoT) device that runs applications. D may be a set of traffic lights or a water valve in a treatment system that is loaded with an application from the operator/manager in order to let traffic/water flow. Alternatively, D could provide information back to a central computer system that is used to monitor flow through the network of pipes, wires or roads. D may be deployed in a public area that is considered to be an insecure environment where attackers have physical access to it.

The Operator (O) is a business or government agency that is contracted with operating/managing some or all of the infrastructure in the smart city such as the road or water network. To operate D, O loads it with software that dictates its behaviour and communications. Producing the software has required a large investment that O wishes to protect. Furthermore, as there is a risk to citizen safety if D is interfered with, protecting device software from tampering benefits O by preventing loss of revenue and any reputation damage caused by incidents.

The Manufacturer (M), produces hardware devices that are installed in smart cities. Different companies may need to manage the devices over their lifetimes, therefore they need to be able to be personalised by different Operators. M wishes to ensure their devices provide protection to the software installed and to provide guarantees as to the security of the software provisioning. If malicious parties replace the device software with their own this may violate citizen privacy or lead to accidents, injuries or lives lost. It is advantageous for M to produce devices that guarantee the security of installed software because it will make them more desirable to city planners.

Finally, City Hall (CH) is the government agency or department that holds overall responsibility for the smart city. This party is the owner of the Devices and is contracting O with running some city infrastructure. In practice, CH may be an IT department, a centralised organisation or an office of the regional or local government. However, for the context of this work the device owner is called City Hall.

2.2 Attacker Model

This work considers an Attacker (Att) that is attempting to interfere with the Devices with two main goals. Atts first goal is to steal the Software (SW) installed on the Device, this may be later used to produce duplicate Smart City devices without authorisation. Secondly, the Attacker wishes to force the smart city devices to execute their own (potentially malicious) software instead of that installed by O.

As stated previously, the Devices in the network may be in public places that Att has physical access to. This physical access allows Att to read the contents of Ds long term storage and RAM. However, Att is not able to read from any secure hardware present on D such as a secure element, TPM or HSM.

Communication between CH, M or O and D uses either wired or wireless connections. Therefore Att has the ability to eavesdrop on communications between these parties and send messages to any of these parties. Wireless communications can be easily overheard and created by an attacker so it is reasonable for Att to send and overhear messages to D. Furthermore, in the wired setting it is also reasonable for Att to hear and send messages as they have physical access to the device and so could tap any communication cabling.

2.3 Protocol Requirements

Based on the model described in Section 2.1 we propose the following requirements for a secure software provisioning scheme.

R1) Entity Authentication: All parties who receive or send applications must be authenticated to prevent application leakage and installing applications from unknown developers.

R2) Replay Resistance: Preventing message replays is required to prevent application leakage via overheard authentication messages.

R3) Perfect Forward Secrecy: Leakage of a session key must not allow attackers to access, deduce or compromise any sessions keys used in future protocol runs.

R4) Transferred Software Integrity: Guarantees will be provided that ensure the correct transfer of the application.

R5) Transferred Software Confidentiality: The application must only be transferred encrypted to prevent leakage.

R6) Secure Software Binding: Software must be securely bound to the device it is installed on to ensure it cannot be transferred to other devices.

R7) Compulsory Software Personalisation: The Device must only execute software provisioned be the authorised O.

R8) Secure Masking Agreement: Both the Device and Operator will contribute to securely establishing keys for application masking.

R9) Secure Key Establishment: Generation and use of session keys will ensure message and application confidentiality.

R10) Single Operator: Devices must only accept software from the Operator currently authorised to manage the Device.

R11) Operator Handover: It must be possible for the Operator of the Device to change.

Requirements R1, R2, R4 and R5 are concerned with ensuring software confidentiality and that D will only accept legitimate software. Requirement R1 ensures O only transfers software to

legitimate devices and that D only accepts software from the authorised O. Similarly, Requirement R2 ensures applications cannot be leaked to Att using captured messages between D and O. Requirement R5 prevents unauthorised installation of the software onto other Ds by ensuring it is protected when transferred to D. Finally, Requirement R4 allows the Device to check the software has not been tampered with in transit.

Requirements R2, R3 and R9 ensure the security of the software provisioning protocol. As stated in Section 2.2, Att is able to eavesdrop on communication between D and O. Requirement R2 ensures old versions of software cannot be reinstalled onto D and previous authentications cannot be reused by Att by replaying old messages. As a critical infrastructure component, D has a long lifetime including software updates, Perfect Forward Secrecy of software provisioning is required to ensure old keys do not leak latest software versions (Requirement R3). Requirement R9 ensures attackers are unable to predict the keys used for securing protocol messages and transferred software.

Requirements R6 to R8 protect the software stored on D. Securely binding the software to D (Requirement R6 ensures it cannot be transplanted from D onto different hardware. Similarly, Requirement R7 ensures that each device will require software to be specially loaded onto it, preventing unauthorised parties from installing software onto Devices. Requirement R8 requires the secret key protecting the software on D be agreed upon by D and the software provider.

D will have only one manager at a time, therefore only one entity should be able to provide software to D at any one time (Requirement R10). However, during the lifetime of D, the Operator may change so it must be possible for control of D to change from one O to another (Requirement R11).

3 MODELS FOR SOFTWARE PROVISIONING

We propose two models for provisioning software bound to Devices: *Device*-centric application provisioning and *Authority*-centric application provisioning. The difference between the two models is which party is responsible for generating the masking-keys and securely transferring the masked software to D.

Section 2.1 describes the entities in the scenario this paper considers. However, from this point we merge the Manufacturer and City Hall into one entity: the Authority (A). The Authority is equivalent to M and CH because A will have the same motivation and responsibilities as the entities it replaces. For example, M is interested in producing Devices that can be securely provided with software; CH also wishes for D to securely receive software from O. To allow D and O to communicate, they may be provided with certificates or pre-installed keys by M or CH. Whether M or CH is responsible for key/certificate provisioning may be determined by practicality/efficiency reasons or by regulations out of the scope of this work. Abstracting these roles into A will allow for any possible division of labour between M and CH to be allowed in the model proposed.
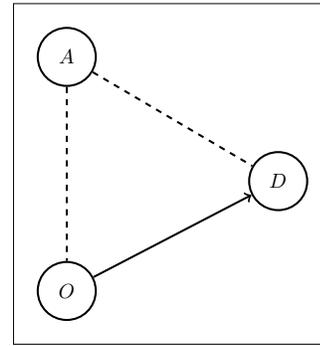
Figure 1: Device-centric application provisioning only requires connection between O and D, however this requires prior trust between D and A and between A and the O.

3.1 Device-centric Software Provisioning

This model divides almost all of the effort of provisioning software to Devices between D and O. In this Device-centric model, A facilitates software provisioning by providing certificates to the parties but does not take an active part in the protocol. It is assumed that both O and D can perform public-key cryptography to verify their identities. This model also assumes that O and D trust A; they are willing to communicate due to mutual trust in A. This setting is modelled in Figure 1; prior trust is shown with a dashed line and software transfer with a solid arrow.

In the Device-centric model, O will need a software provisioning certificate from A. This will be a public key, and an expiry time value signed by A. The expiry time value may be a start or finish time for the certificate depending on whether an O is always authorised for fixed period of time or for varying time periods.

For O to load SW onto D it will begin the protocol by contacting D. At the start of the protocol, O and D will perform a fresh mutual authentication. D then checks the certificate provided by O to ensure they are currently authorised to provide software to D. Next, O and D establish a session key for the protocol run to ensure message confidentiality. Once a session key has been established, a masking key is established, O then masks SW and transfers it to D.

3.2 Authority-centric Software Provisioning

Alternatively to the Device-centric model in Section 3.1 is the Authority-centric software provisioning model. In this model, O is prevented from directly communicating with D and instead transfers the application to D via A. This model addresses the possibility that O is trusted to provide software to D but is not permitted to directly modify its software. This may be necessitated by regulation preventing City Hall from granting complete control to O or simply to allow easier auditing of Devices. The Authority-centric approach comprises two main use-cases; the "Application-Relay" and "Application-Broadcast" models.

The first model is the "Application-Relay" model shown in Figure 2. This model is similar to the Device-centric model however it involves O transferring the SW to D via A instead of directly. In this use case one D is provided with a software update or installation from O. Like in the previous model, O will establish communication
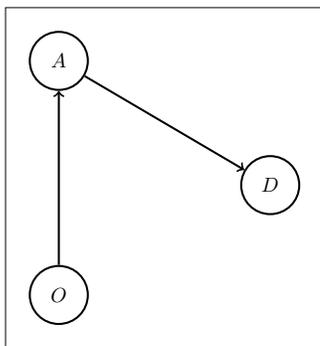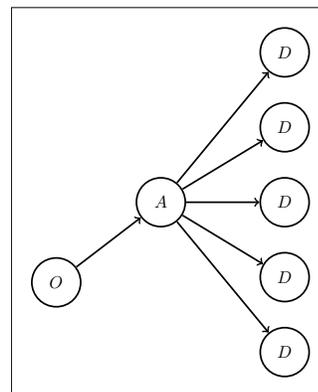
Figure 2: The Application-Relay model allows an Authority to monitor application provisioning from O to D.

Figure 3: Application-Broadcast model considers the situation in which O is loading a large number of Devices with a single application.

with A, the parties will authenticate, establish a session key and the application will be transferred. Next, A and D will communicate, first authenticating themselves then establishing keys for communication and masking. Finally, A will mask the application before transferring it to D.

The second model is the "Application-Broadcast" model depicted in Figure 3. This use case is an extension on the "Application-Relay" model and considers a set of identical Devices, $D_{1...n}$, managed by O. When O wishes to install or update software on the Devices then they establish communication with A and transfer the software to them as in the previous model. However, once A has received the software, it establishes communication with all of the Devices and transfers it to them. In practice, this setting could apply when a smart city contains a number of the same model of traffic lights or a different smart city device. O may need to update the software on the Devices to update the communications protocol to a different cipher or to expand the data being sent from D to O. In the A-centric model, software transfers must be performed via A, however if many devices require the same software update it would be more efficient to allow A to receive the software once and then update the devices. Once A has received the application they establish sessions and masking keys with each of $D_{1...n}$, masks the application for each Device and transfers each version to $D_{1...n}$.

An advantage of the manufacturer-centric model is that it is compatible with less powerful devices. Low-end devices have less computational power, sources of randomness and sometimes battery life than a server ran by A that is better equipped to generate strong masking keys.

One argument against the Authority-centric models proposed is that they both require O to transfer the unmasked application to A. Initially, it may seem as if this requires an unreasonable amount of trust in the Authority. However, as A is responsible for producing the keys used by D and owns the devices so also has physical access to them it is always in a position to extract the application. If A would always be able to access the unmasked application via D then it requires no more trust to be placed in A to transfer the application to D via them than to use Authority-centric software provisioning at all.

4 RELATED WORK

This section describes previous work in the areas of securely binding hardware and software (Section 4.1). Later, this section describes two significant industry approaches in software provisioning: MULTOS (Section 4.2) and GlobalPlatform (Section 4.3).

4.1 Binding Hardware and Software

Binding hardware and software applies to various problems including: Digital Rights Management, preventing product counterfeiting and preventing unauthorised firmware modification [7, 8]. Hardware/Software bindings can either be created bi-directionally or uni-directionally. Krasinski and Rosner and Atallah et al. consider binding hardware and software uni-directionally as ensuring a piece of software requires a particular hardware device to execute [2, 7]. Alternatively, is the bi-directional approach taken by Lee et al. who describe binding hardware and software as creating a bond that ensures that both require the other for execution [8].

Due to the different bonds formed, the binding schemes described by Krasinski and Rosner, Atallah et al., and Lee et al. approach the problem in different ways. All three schemes make use of a device specific item to bind software and hardware, however the use and source of the items differ. Krasinski and Rosner use a device specific method for generating a key that is checked whenever the copy protection software they are binding executes, however they are not clear how the key is used [7]. Atallah et al. use a Physically Unclonable Function to provide response data to bind an instantiation of RSA to a particular hardware device that ensures it is being executed on the intended hardware [2]. Both of the uni-directional schemes are limited to checking the software has not been transplanted to a different device, however the Lee et al. scheme also ensures the hardware instance only executes software bound to it. Lee et al. bind the hardware and software at an instruction level that ensures that neither the hardware or application can be changed [8].

Due to Requirement R6, a solution is required to prevent software being transferred from one D to another by *Att*. Furthermore, Requirement R7 states that D must only execute software that

has been provisioned to it by the authorised O. Therefore, the binding scheme that provides the level of security required is that of Lee et al. However, their proposed scheme only offers secure binding between hardware and software and has no mechanism for deploying the scheme in the real world. In their paper, Lee et al. list updates as future work and assume that installation takes place in a secure environment [8]. Therefore, their proposal requires an extension before it can be applied to the problem considered in this work.

4.2   MULTOS

MULTOS is an operating system for multi application devices developed and standardised by the MULTOS Consortium. Historically, it has been used in smart cards for payment, ID, passports and transport ticketing [9]. MULTOS does not provide protection for installed applications, however the specification requires that development of MULTOS devices be security tested to at least EAL4+ Common Criteria or C.A.S.T. standard [11].

The security of MULTOS is primarily constructed using public key cryptography to provide secure communication channels between Devices, Issuers and Application Providers. A mechanism for securely provisioning software and firmware updates to devices is included in the MULTOS specification. To install or update an application on a MULTOS device an application provider must work with the device Issuer to produce an Application Load Unit (ALU) and an Application Load Certificate (ALC). The ALU is a container object comprising the application, any application personalisation data that is needed and a hash of the application signed by the application provider. ALUs can be either public or confidential depending on the security required, confidential ALUs are protected using the public key installed during the enablement of the MULTOS device [10]. The ALC is an authorisation certificate generated by the device issuer containing the public key of the Application Provider and the application header containing the application ID, hash, and storage requirements. Before an application can be installed on a device, the installation must be authorised with an ALC.

MULTOS software provisioning considers a similar problem to that described in this paper, however the solution does not address several points of the problem we are interested in solving. Firstly, MULTOS does not provide any solution to the problem of how to protect applications installed on a MULTOS device. Instead, the MULTOS specification requires that installed software is protected on the device but does not define how it is to be protected. Therefore, it does not meet with Requirements R6 and R7.

Secondly, it does not fit with either of the two models proposed in Section 3 as it considers a model that is a combination of the two proposed in this work. MULTOS software provisioning is most similar to the Authority-centric model, however the issuer is not aware of the application being installed on the device. In producing the ALC, the issuer approves installation of an application matching the provided application header, but the actual application is unknown. Mass updating of all managed Devices (Figure 3) is not possible in MULTOS; each device must be contacted by the application provider separately as no broadcast model is supported.

4.3   Global Platform

GlobalPlatform is a smart card specification concerned with allowing smart cards to run applications from multiple different providers in a secure manner. The standard has a very wide scope and covers system, card and security architectures, life cycle models, card and application management, and secure communications. Both the secure provisioning of applications and how to protect them in storage are included in the standard so GlobalPlatform [1] may solve some of the problems considered in this work.

In GlobalPlatform, applications are protected using the GlobalPlatform Runtime Environment. The Runtime Environment provides an API for applications, secure storage and secure execution space for applications that separates each applications code and data from other applications. Finally, the Runtime Environment provides communication between the card and off-card entities [12].

GlobalPlatform provides a strong mechanism for secure application installation, management and control via Security Domains. Security Domains are on-card representatives of the Card Issuer and the Application Providers and provide secure services such as key storage, encryption, decryption and signing/verifying digital signatures. Each GlobalPlatform device contains at least one Security Domain: the Issuer Security Domain. The Issuer Security Domain can be responsible for all the security critical tasks listed above or it can allow Application Providers their own on-device space by authorising Application Provider Security Domains to perform the listed tasks [1].

Security Domains allow application installation by verifying the Data Authentication Patterns (DAP) that are provided in the Load File Data Blocks used to transfer applications to GlobalPlatform devices. If a Security Domain has the relevant permissions it is able to accept applications that are accompanied by a DAP using a shared symmetric or asymmetric key from the Application Provider. Security Domains can be locked to prevent their use and their permissions can be limited to ensure the Issuer retains overall control of the device [1].

With the correct configuration a GlobalPlatform based solution could meet with almost all of the requirements described in Section 2.3 and fit with the models in Section 3. The only requirements that native GlobalPlatform does not meet are Requirements R6 and R7. Although as GlobalPlatform is hardware agnostic a hardware using the system described in Section 4.1 could allow the solution to meet all requirements. The hybrid provisioning required with hardware/software binding could be implemented using a custom protocol to include masking key establishment as GlobalPlatform permits custom protocols.

However, while GlobalPlatform could be combined with a hardware/software binding scheme to address the problem considered in this work, any solution based on GlobalPlatform would almost certainly be not optimal. GlobalPlatform is a large, complicated specification designed to apply to many different scenarios and using it in solving the problem described would introduce unnecessary complication to the system. A bespoke solution would offer more simplicity and efficiency than using a large standard such as GlobalPlatform for a small amount of the functionality offered.

Table 1: Protocol Notation

n_x	A nonce with id x.
g^x	A Diffie-Hellman key share.
$\text{Enc}_k(m)$	The encryption of the message m using the key k.
O-cert	The O-Cert of the Device Operator.
sk_X	The signing key of X.
$\text{Sig}_k(m_x)$	The hash-based signature of a message x produced using the key k.
$\text{H}(m)$	The hash digest of a message m.
$M_k(App)$	The software App masked using a key k.
$\text{MAC}_k(m)$	A Message Authentication Code calculated for a message m using a key k.

5 PROPOSED SOLUTION

As described in Section 4, no solution exists to solving the problem of provisioning software to devices with hardware/software binding. This section describes a new solution that draws on previous work by Lee et al.

5.1 Overview of Solution

The scheme by Lee et al. is the only existing scheme providing hardware/software binding compliant with Requirements R6 and R7. Therefore it will be used to protect software installed on the Devices.

As stated in Section 4.3, a new protocol is required to provision software to the Devices in the network. A solution using a bespoke protocol will be more efficient than one based on a niche configuration of an existing standard and will also be easier to analyse. However, the different models of software provisioning described in Section 3 will require separate protocols due to the significantly differing roles of A and O. The new protocols for the different software provisioning models are described in Section 5.2 and Section 5.3.

The notation used to denote the proposed protocols is listed in Table 1.

5.2 Device-centric Software Provisioning

This section describes the protocol for provisioning software in the Device-centric model presented in Section 3.1. The developed protocol is presented in Table 2.

The D-centric protocol is a three message protocol that establishes freshness and authentication by O and D signing freshly generated nonces. This allows the protocol to meet Requirements R1 and R2.

Keys for the session and masking are established using Diffie-Hellman key agreement. These are carried out by the exchanging of g^a, g^b, g^c, g^d; allowing the mutual agreement upon the session and masking keys: g^{ab} and g^{cd}. This generation and use of session and masking keys for each protocol run ensures the scheme satisfies Requirements R3, R5, R8 and R9.

Requirements R10 and R11 dictate that only one, authorised O may install software on D at a time and that it must be possible for one O to cease provisioning SW and another to start. In the Device-centric protocol this is achieved using the O-cert. The O-cert is a

Table 2: Device-centric Software Provisioning

Device-centric Protocol		
1.	$O \rightarrow D$	$O\|D\|n_1\|g^a\|O\text{-cert}\|\text{Sig}_{\text{sk}_O}(m_1)$
2.	$D \rightarrow O$	$D\|O\|n_1\|n_2\|g^b\|\text{Enc}_{g^{ab}}(g^c)\|\text{cert}_D\|\text{Sig}_{\text{sk}_D}(m_2)$
3.	$O \rightarrow D$	$O\|D\|n_2\|\text{Enc}_{g^{ab}}(M_{g^{cd}}(SW)\|\text{H}(SW)\|g^d)$ $\|\text{Sig}_{\text{sk}_O}(m_3)$

certificate that is provided to O by A and includes pk_O signed by A's public key as well as a certificate expiry time and information describing the devices O is permitted to manage. Certificate expiry times and only granting one valid O-cert at a time will ensure the protocol meets Requirements R10 and R11. Alternatively, the O-cert could contain a start time and always have a fixed lifetime, however specific expiry times allow certificate expiry to more easily align with trial periods/Operator contracts. The certificate of D in the Device-centric protocol is a normal certificate containing the public key of D signed by A.

Finally, the inclusion of the application signature ensures that Requirement R4 is met by the protocol.

5.3 Authority-centric Software Provisioning

This section will describe the protocols for provisioning software according to the Authority-centric software provisioning model in Section 3.2. The main difference between the Authority-centric and Device-centric protocols is that the former requires a two stage process as the application must be transferred to D via A instead of directly from O to D. Therefore, the Authority-centric approach is longer and less efficient than the Device-centric protocol proposed in Section 5.2. The Authority-centric protocols proposed (Table 3) is a two stage protocol consisting of transferring the application from O to A (messages 1-3) and then from A to D (messages 4-6). This allows for Application Relay as depicted in Figure 2 and Application-Broadcast as depicted in Figure 3. The Application-Broadcast would require running the latter half of the protocol multiple times, once for each device.

The second half of the protocol in the A-centric model can be approached in two ways depending on the relationship between A and D: with public key cryptography to build trust or using pre-existing trust. In D-centric software provisioning O makes contact with D to install software. It is assumed that both parties are communicating based on their mutual trust in the Authority (A) rather than an existing trust relationship. By contrast in the A-centric model it is plausible that the existing trust relationship between D and A may include having a pre-shared symmetric key. In such a case D may not be required to do as many public key cryptographic operations which is advantageous in resource-constrained settings. Therefore a second A-centric protocol is proposed here that uses a pre-shared key to replace some signatures with MACs (Table 4).

Requirements R1 to R3, R5, R8 and R9 are all achieved in the Authority-centric protocols in the same manner as described in Section 5.2 for the Device-centric protocol. However, Requirements R10 and R11 do not require a solution in the Authority-centric model as A is able to trivially ensure by only accepting software from the currently authorised O and no others. Finally, Requirement R4 is

Table 3: Authority-centric Software Provisioning (with Asymmetric Cryptography capable Devices)

Authority-centric Provisioning

1. $O{\to}A$ $O\|A\|n_1\|g^a\|O\text{-cert}\|\text{Sig}_{\text{sk}_O}(m_1)$
2. $A{\to}O$ $A\|O\|n_1\|n_2\|g^b\|\text{cert}_A\|\text{Sig}_{\text{sk}_A}(m_2)$
3. $O{\to}A$ $O\|A\|n_2\|\text{Enc}_{g^{ab}}(SW\|H(SW))\|\text{Sig}_{\text{sk}_O}(m_3)$
4. $A{\to}D$ $A\|D\|n_3\|g^c\|\text{Sig}_{\text{sk}_A}(m_4)$
5. $D{\to}A$ $D\|A\|n_3\|n_4\|g^d\|\text{Enc}_{g^{cd}}(g^e)\|\text{cert}_D\|\text{Sig}_{\text{sk}_D}(m_5)$
6. $A{\to}D$ $A\|D\|n_4\|\text{Enc}_{g^{cd}}(M_{g^{ef}}(SW)\|H(SW)\|g^f)$
   $\|\text{Sig}_{\text{sk}_A}(m_6)$

Table 4: Authority-centric Software Provisioning (with Pre-Shared Keys

Authority-centric Provisioning with Pre-Shared Keys

1. $O{\to}A$ $O\|A\|n_1\|g^a\|O\text{-cert}\|\text{Sig}_{\text{sk}_O}(m_1)$
2. $A{\to}O$ $A\|O\|n_1\|n_2\|g^b\|\text{cert}_A\|\text{Sig}_{\text{sk}_A}(m_2)$
3. $O{\to}A$ $O\|A\|n_2\|\text{Enc}_{g^{ab}}(SW\|H(SW))\|\text{Sig}_{\text{sk}_O}(m_3)$
4. $A{\to}D$ $A\|D\|n_3\|g^c\|\text{MAC}_{PSK}(m_4)$
5. $D{\to}A$ $D\|A\|n_3\|n_4\|g^d\|\text{Enc}_{g^{cd}}(g^e)\|\text{MAC}_{PSK}(m_5)$
6. $A{\to}D$ $A\|D\|n_4\|\text{Enc}_{g^{cd}}(M_{g^{ef}}(SW)\|H(SW)\|g^f)$
   $\|\text{MAC}_{PSK}(m_6)$

met using digital signatures in the Public-Key scheme and by using a MAC in the Pre-Shared Key protocol.

6 ANALYSIS

Further to the analyses of the proposed protocols in Sections 5.2 and 5.3 a formal, mechanical analysis of the protocols has been carried out using Tamarin Prover [13]. This section describes the protocol models, the analysis carried out and the security results proven.

6.1 Description of Protocol Models

The protocol analysis was carried out using three models: one for each protocol proposed. The models represent the protocols as state machines that are progressed through as messages are sent and received by the Device, Authority and Operator.

A digram of the state machine for analysing the D-centric protocol is included in Figure 4. This model contains two types of transitions between the rules representing the states of the protocol: messages and entity-states. Both the correct message and entity-state are required before a rule can be executed. The messages are the outputs of the $D1$, $O1$ and $D2$ rules that are sent between the Operator and Device that are also sent to the attacker.

The entity-states are facts that store the information that a particular entity in the protocol run is aware of and needs to retain. These states are created as private facts that the attacker is not shown and are used to retain information such as Diffie-Hellman exponents. This state machine representation of the protocol allows a legitimate execution path to be represented with the rules evaluated in the order: $O1$, $D1$, $O2$, $D2$. It also limits attacker behaviour to only realistic attacks such as trying to forge a session with a legitimate
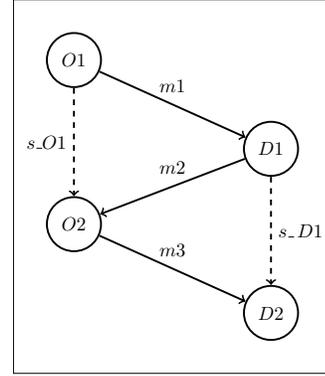
Figure 4: Modelling Device-centric application provisioning requires a state machine of four states with five connections.

A for part or all of a protocol run. The attacker can also try to break the scheme security by eavesdropping a legitimate protocol run. This is reasonable as it ensures the attacker cannot start the protocol by forging $m2$; a legitimate Operator would notice that $m1$ had not been sent.

Similar state models for the Authority-centric protocols were developed to produce Tamarin models for the proposed protocols, however they are omitted for brevity.

6.2 Scope of Mechanical Analysis

The Tamarin models prove the communication requirements of the protocol. Therefore, rules and lemmas were included in the model that tested Requirements R1 to R5 and R8 to R11. Requirements R6 and R7 were not included in the model and it was assumed that installed software is securely stored.

The method for the analysis was to first prove that the protocols satisfy Requirements R1 and R2. Intuitively, some requirements (such as Requirements R8 and R9) rely on fresh authentication. Therefore the Tamarin model first proves Requirements R1 and R2 and then uses those to verify that the other requirements tested have been met.

To limit the complexity of the model it is assumed that an entity in the model is either a Authority, Device or an Operator and they do not switch roles. However, a result of this is that different messages in the protocol will only be created by certain entities such as message 5 is only ever output by a Device.

6.3 Modelling Assumptions Required

To model the proposed protocols several assumptions were made. Some assumptions were required to allow Tamarin Prover to reason about the protocols. However, most assumptions were chosen to limit model complexity without compromising its value. An example is preventing entities from switching roles. This is not unreasonable because a traffic light in a city is not going to change into a different device or become an Authority or Operator during its lifetime. This assumption did not weaken the security proven by the model but did simplify the protocol verification.

Other assumptions were made to prevent Diffie-Hellman key-exchange exponent edge cases that could theoretically lead to attacks. Examples of these are assuming all parties will only generate safe Diffie-Hellman shares. This includes excluding all shares or share combinations that are equal to the generator as these would allow for a trivial break by the attacker. The chance of these scenarios is negligible so preventing them by assumption is reasonable and allows the model to prove the security properties without weakening the result.

Another example of a simplifying assumption is to assume that all generated nonces are unique. In practice, an implementation would check that received nonces are fresh so this assumption is reasonable as it removes the need to check nonce freshness from the model.

6.4 Results of Analysis

When verifying the protocols met Requirement R10, the model showed that implementations of this protocol will need to avoid problems with O-Certs expiring between messages. Sending the O-Cert in the initial message makes sense in terms of practicality: D or A need not continue to communicate with O if they do not have a valid O-Cert. However, the Tamarin model highlighted the possibility of a time of check, time of attack vulnerability. Preventing this vulnerability simply requires implementations to check the O-Cert is still valid at install time and has not expired since receipt.

After including extra O-Cert validity check the protocol models met all security requirements. Therefore, when implemented correctly, the three protocols described in Section 5 will meet the requirements described in Section 2.3.

One limitation of the analysis carried out is that it does not consider the consequences of any compromised devices. The current model assumes the parties are honest but communicating over a channel controlled by the attacker. This allows the model to cover any combination of honest cases but it does not consider if individual devices are compromised. The security of the key establishment has been proven in the context of the attacker being able to break some session keys. However, a malicious device can always leak the current key or even the application being transferred so some requirements will not be able to be met by the current scheme.

A further limitation to the analysis is that it is only interested in full breaks or application leaks. If one version of a piece of software is leaked to the attacker, this might be of use to an attacker in guessing parts of the next version of the code. However, the existing model has only considered proving the security of the protocol against total software leakage and does not model partial information leakage.

7 IMPLEMENTATION

This section describes the implementation developed to demonstrate and evaluate the protocols proposed. All three protocols were implemented and execution times for each entity involved were recorded.

7.1 Test Setup

The protocols were implemented using a Raspberry Pi 3 Model B as the Device and a laptop in place of the Operator and Authority. The laptop is a 2013 Macbook Pro with a 2.5 GHz Intel Core i5 Processor and 8GB DDR3 RAM. The Raspberry Pi 3 has a 1.2GHz Quad-Core ARM Cortex-A53 Processor, 1GB LPDDR2 RAM and 802.11 wireless LAN connectivity. The communication between Raspberry Pi and laptop was carried by a normal WiFi network. For the Authority-centric protocols, the laptop executed the code for the Authority and Operator and the two communicated using the loopback network interface.

The implementations of the protocols were written in Python and the cryptographic functions were provided by the library *cryptography.io* [14]. Several standardised cryptographic algorithms were used including: AES-128 in CBC mode, SHA-256, ECDSA and ECDHE using the SECP384r1 curve, PKCS7 padding. Encryption and signing algorithms were chosen based on recommendations made by various organisations compiled by Damien Giry [5].

7.2 Results

The protocols were ran multiple times to measure the execution time of each entity in each protocol. Each protocol was executed ten times and the results presented are the mean average and sample standard deviation values for each set of protocol timings, the results are provided in Table 5. The number of sample execution times was low, however the low sample standard deviation values suggest that the timings provided are indicative of the performance of the protocols.

The results show the execution time of each protocol is very low despite each involving the verification of MACs and/or digital signatures. By comparing the performance results for the Device-centric protocol to those of the Authority-centric protocols it seems that using the Raspberry Pi caused a delay. This is indicated by the significantly lower execution times of the Operator when it only communicated with the Authority. However, it is not clear as to if the delay is caused by a lack of computing power on the Raspberry Pi or if the delay was due to the latency of the wireless communications between systems.

One unexpected result is that the A-centric protocol using pre-shared keys appears to be slightly slower than the more public key reliant version. This was not expected as the pre-shared keys protocol was proposed to be a more efficient option. Messages m4 and m6 were the same length in both although m5 was much larger in the public key reliant version. Therefore, the slower execution cannot have been caused by the time for communication. It may be possible that the *cryptography.io* library has an efficient ECDSA algorithm that allows it to execute more quickly than the HMAC algorithm. However, this seems unlikely so further optimisation and study is needed to explain this result.

8 CONCLUSION

In this paper we have presented a new problem setting for provisioning software to devices in smart cities. The new scenario applies to the real world and includes the important issue of protecting the software that is being installed on the devices.

Table 5: Performance Results

Protocol	Execution Times (in seconds)					
	Device		Operator		Authority	
	μ	σ	μ	σ	μ	σ
D-centric	0.187	0.00955	0.183	0.0103	-	-
A-centric	0.115	0.0102	0.0197	0.00104	0.118	0.00623
A-centric (with PSK)	0.135	0.0142	0.0189	0.00106	0.153	0.0121

Two models for distributing software to devices in a smart city are described and protocols that solve the problems they present have been suggested. The protocols have been proven to be secure using a formal analysis tool that considers large numbers of possible combinations of messages when looking for flaws.

The future work in this area would be to the community would be to expand the analysis to check the security of the scheme when some devices are compromised. The attacker described in Section 2.2 has physical access to the Devices so could potentially mount an attack to gain control over some Devices. Alternatively, another area for future work would be examining the performance cost of the proposed protocols by implementing them using various microcontrollers. This would provide cost measurements in terms of time and energy required to use the proposed scheme that would allow for easier comparison with other protocols.

REFERENCES

[1] 2015. *Global Platform Card Specification*. Version 2.3. GlobalPlatform.
[2] Mikhail J. Atallah, Eric D. Bryant, John T. Korb, and John R. Rice. 2008. Binding Software to Specific Native Hardware in a VM Environment: The PUF Challenge and Opportunity. In *Proceedings of the 1st ACM Workshop on Virtual Machine Security (VMSec '08)*. ACM, New York, NY, USA, 45–48. DOI : http://dx.doi.org/10.1145/1456482.1456490
[3] Department for Business Innovation and Skills. 2013. *Smart Cities: Background Paper*. Technical Report. HMG.
[4] European Commission. 2015. Smart Cities. (2015). https://ec.europa.eu/digital-single-market/en/smart-cities
[5] Damien Giry. 2017. Keylength - Cryptographic Key Length Recommendation. (2017). https://www.keylength.com/en/
[6] IEEE. 2017. Readings on Smart Cities. (2017). www.smartcities.ieee.org/articles-publications/ieee-xplore-readings-on-smart-cities.html
[7] R. Krasinski and M. Rosner. 2003. Method for binding a software data domain to specific hardware. (May 2003).
[8] Robert P. Lee, Konstantinos Markantonakis, and Raja Naeem Akram. 2016. Binding Hardware and Software to Prevent Firmware Modification and Device Counterfeiting. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical System Security, CPSS 2016, Xi'an, China, May 30, 2016*, Jianying Zhou and Javier Lopez (Eds.). ACM. DOI : http://dx.doi.org/10.1145/2899015.2899029
[9] MAOSCO Ltd. 2014. Securing Smart Meters with MULTOS - Technical Overview. (2014). https://www.multos.com/uploads/MULTOS_in_Smart_Meters_Technical_Overview.pdf
[10] MAOSCO Ltd. 2015. Guide to Generating Application Load Units. (2015). https://www.multos.com/uploads/GALU.pdf
[11] MAOSCO Ltd. 2017. MULTOS Explained. (2017). http://www.multos.com/multos_explained.htm
[12] Konstantinos Markantonakis and Keith Mayes. 2003. An overview of the GlobalPlatform smart card specification. *Information Security Technical Report* 8, 1 (2003), 17 – 29. DOI : http://dx.doi.org/10.1016/S1363-4127(03)00103-1
[13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*. Springer Berlin Heidelberg, Berlin, Heidelberg, 696–701. DOI : http://dx.doi.org/10.1007/978-3-642-39799-8_48
[14] Python Cryptographic Authority. 2017. cryptography.io. (2017). https://cryptography.io/