HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

ONBOARD CREDENTIALS: HARDWARE ASSISTED SECURE STORAGE OF CREDENTIALS

AISHVARYA KUMAR SHARMA

Master's thesis
17-January-2007

Supervisor: Professor N. Asokan
Advisor: Kari Kostiainen, M.Sc. (Tech.)

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Aishvarya Kumar Sharma
Title:	Onboard Credentials: Hardware Assisted Secure Storage of Credentials
Date:	17-01-2007
Pages:	66
Department:	Department of Computer Science and Engineering
Professorship:	T-110
Supervisor:	Professor N. Asokan
Instructor:	Kari Kostiainen, M.Sc. (Tech.)

There is a plethora of services offered on the internet and a user may access many of these disparate services simultaneously. The user possesses multiple credentials for accessing these growing numbers of services and credential storage mechanisms have been devised to address this problem. Software credentials such as applications storing passwords are vulnerable to dictionary attacks and malwares. Storing credentials using secure hardware is expensive and inflexible. Secure hardware modules such as Trusted Platform Modules (TPM) are becoming increasingly available and a system can be designed that utilizes such commodity hardware.

In this thesis, a novel architecture called "Onboard Credentials" is presented. In this architecture, a credential consists of a secret and an accompanying algorithm. The algorithm is expressed as scripts, which are executed to encrypt the secret in an interpreter running in a secure execution environment. Such a design allows third parties to develop new credential types and utilize the secure environment. We implement a proof-of-concept of the proposed architecture on two platforms: Linux using TCG/TPM as the secure hardware and on Symbian OS utilizing its Platform Security and evaluate the design and implementation.

Keywords:	security, secure environment, virtual machine, credentials, provisioning
Language:	English

Preface

The work related to this thesis was part of the NEON project in the Networking Technologies Laboratory at Nokia Research Centre, Helsinki. The funding for writing this thesis was received through research assistantship from Telecommunications and Communications Laboratory, Helsinki University of Technology.

I offer my profound gratitude to Professor Antti Ylä-Jääski for giving me this awe-inspiring opportunity to work in this esteemed project. I would like to express my sincere gratitude to my supervisor, Professor N. Asokan, whose expertise, understanding and patience were pivotal to the timely and successful completion of this thesis. I am grateful to Kari Kostiainen, my advisor, for his kind help and feedback during the project and for his immense patience while reading the countless drafts of this thesis. Jan-Erik Ekberg has been a great source of inspiration and I sincerely thank him for his much valued support and guidance. Aarne Rantala kept the work place alive and was always there for help. Thanks to Lucia Tudose and Kalle Kuismanen for their support and troubleshooting problems with me. This thesis would not have been possible without the help and cooperation extended to me by my other numerous colleagues at Nokia and at TML, HUT.

I also express my thanks to my family in particular my grandfather, for the love and support imparted to me all these years. I am grateful to my friend Raghav Karol for helping and encouraging me to come to Finland for studies. I thank Deepak and Ishvar for patiently listening to my stories at the end of the day and my flat mates Antti, Marri and Mikko for making my stay in Finland a fun and memorable experience.

Aishvarya Sharma
Espoo, Finland
17-January-2007

Table of Contents

Abstract i
Preface iii
Table of Contents v
Table of Figures vii
Glossary viii
1 Introduction 1
1.1 Motivation 1
1.2 Problem Statement 1
1.3 Organization of this thesis 3
2 Background 4
2.1 Secure hardware modules 4
2.1.1 Trusted Computing 4
2.1.2 ARM TrustZone 8
2.1.3 Java Cards 10
2.2 PKCS#11 – Cryptographic Token API 11
2.3 Platform Security 12
2.3.1 SELinux 12
2.3.2 LIDS – Linux Intrusion Detection System 13
2.3.3 Symbian Platform Security 13
2.4 Trusted Path 14
2.5 Related work 15
2.5.1 Solutions using secure hardware 15
2.5.2 Applications storing passwords 18
2.5.3 Single Sign-On 20
3 Criteria 23
4 Design 25
4.1 Onboard Credentials 25
4.1.1 Secure Environment 25
4.2 Architectural Overview 26
4.2.1 Usage 26
4.2.2 Credentials Interpreter 30
4.2.3 Credential Server 31
4.2.4 Database storage 32
4.3 Protecting the Credentials 33
4.3.1 Storage Key Initialization 33
4.3.2 Provisioning 34

4.3.3 Local Access Control 36
5 Implementation 38
5.1 Lua Interpreter 38
5.1.1 Working of the Lua Interpreter 38
5.2 Linux Implementation 39
5.2.1 Linux Architecture 39
5.2.2 Lua Interpreter in Linux 40
5.2.3 Protecting the Credentials 41
5.2.4 Credential Server 44
5.2.5 Modified application - Mozilla Firefox Browser 45
5.3 Symbian Implementation 47
5.3.1 Symbian Architecture 47
5.3.2 Lua Interpreter in Symbian 49
5.3.3 Protecting the Credentials 49
5.4 Contribution in this thesis 50
6 Analysis 52
6.1 Security Evaluation 52
6.2 Usability Evaluation 54
6.3 Criteria Evaluation 55
6.4 Future Work 60
7 Conclusion 62
8 Bibliography 63

Table of Figures

Figure 1: Existing Credentials Storage Systems 2
Figure 2: TPM Chip [9] 5
Figure 3 Secure Boot / Signatures [9] 8
Figure 4: ARM TrustZone's Parallel Secure Side [16] 9
Figure 5: ARM TrustZone Software Architecture [15] 10
Figure 6: Trusted UI in PassPet Tool [49] 19
Figure 7: CardSpace Identity Selection Screen [55] 21
Figure 8: Onboard Credentials 25
Figure 9: Onboard Credentials Architecture - Provisioning Use case 28
Figure 10 Onboard Credentials Architecture - Credentials Usage 29
Figure 11 Credential Interpreter inside secure execution environment 30
Figure 12: Initialization of the Original Platform Key (OPK) 34
Figure 15: Memory buffer layout for the Lua Interpreter 39
Figure 13: TLV Format 39
Figure 14: Data layout in TLV format as sent to Lua Interpreter 39
Figure 16: The Basic Linux Architecture 40
Figure 17: Platform Key hierarchy in Linux with Provisioning 43
Figure 18: First time access to website. Username/password prompt 45
Figure 19: User asked to select access level 46
Figure 20: On subsequent visits, only PIN is asked while displaying the custom text. 46
Figure 21: Login is successful 47
Figure 22: Symbian Architecture 48

Glossary

AES-CBC	Advanced Encryption Standard - Cipher Block Chaining
AIK	Attestation Identity Key
API	Application Programming Interface
ASP	Authentication Service Provider
BIOS	Basic Input Output System
COMKey	Communication Key
DAA	Direct Anonymous Attestation
DLL	Dynamic Link Library
DM	Device Management
DNS	Domain Name Service
EK	Endorsement Key
FVEK	Full-Volume Encryption Key
GSM	Global System for Mobile communication
GTK	GNU Took Kit
HMAC	Hash based Message Authentication Code
HTTP	Hyper Text Transfer Protocol
IdP	Identity Provider
IMSI	International Mobile Subscriber Identity
InitRD	Init Ram Disk
IPC	Inter Process Communication
LED	Light Emitting Diode
LIDS	Linux Intrusion Detection System
MAC	Message Authentication Code
MAC	Mandatory Access Control
NGSCB	Next Generation Secure Computing Base
OMA	Open Mobile Alliance
OPK	Original Platform Key
PID	Process Identifier
PIN	Personal Identification Number

PKI	Public Key Infrastructure
PPK	Provisioned Platform Key
RDBMS	Relational Database Management System
S/MIME	Secure / Multipurpose Internet Mail Extensions
SAML	Security Assertion Markup Language
SIM	Subscriber Identity Module
SK	Script specific Key
SOAP	Simple Object Access Protocol
SP	Service Provider
SRK	Storage Root Key
SSL	Secure Socket Layer
SSO	Single Sign-On
TCG	Trusted Computing Group
TLS	Transport Layer Security
TLV	Type Length Value
TMSI	Temporary Mobile Subscriber Identity
TPM	Trusted Platform Module
TrouSerS	TCG Software Stack
UI	User Interface
USB	Universal Serial Bus
VMK	Volume Master Key
VSIM	Virtual Subscriber Identity Module
WLAN	Wireless Local Area Network
WWAN	Wireless Wide Area Network

1 Introduction

1.1 Motivation

A credential is any information about the user that is asserted either by the user himself or by another party; examples are the user's name, preferences, access rights, etc. In modern electronic transactions, trust is established with digital credentials, which often consist of a secret that is used for security related operations such as authentication, authorization and encryption. A traditional example of a credential is username and password. A more comprehensive example of a credential is a SecurID [1] token used in many organizations as means of authentication for VPN access. A SecurID token consists of a cryptographic key as a secret and a set of algorithms to operate on that key.

There is wide assortment of services offered on the Internet, such as e-mail, internet banking, internet games, online shopping, etc. The user is required to present a different credential for accessing each of these, often disparate, services. For example, one may need to supply credentials once for checking e-mail, then present another credential for accessing bank account, another for shopping on e-bay and another on amazon.com.

1.2 Problem Statement

Mechanisms to securely store credentials have been devised to address the problem of managing multiple credentials. Using dedicated terminals is not a scalable solution and is used in situations where the security demands are very high (example: ATM machines). One may protect the credentials in any generic device using two methods as illustrated in Figure 1:

- **Software Credentials:** In this method, the credentials are stored as software tokens and the security is provided by the underlying operating system and the application. Browsers such as *Firefox* [2] and plug-ins like *PwdHash* [3] and *Password Multiplier* [4] offer to encrypt the passwords with a key (such as a local PIN or short passphrase) before storing the passwords on the local file system of the computer. No binding exists between the secret and the algorithm that prohibits a weaker algorithm to be used for encryption. Further, user passwords and PINs are susceptible to guessing attacks [5] and thus make software credentials vulnerable.

  Although most operating systems provide isolation between different processes in memory, they seldom provide application specific access control on persistence storage. Therefore, no protection exists from a malware running with the user's privileges to access the

credentials stored on persistence storage. The malware may also modify the application to gain access to the stored credentials. It may also indulge in spoofing an application in an attempt to obtain, from the user, the PIN that has been used to encrypt the credentials. Credentials stored as software tokens are thus problematic in spite of offering ease of use, cost-effectiveness and flexibility.

- **Hardware Credentials:** Credentials may also be stored as hardware tokens. A separate dedicated hardware module provides security and as the access is more tightly controlled and defined, the overall attack surface is reduced [6]. The most common examples of hardware credentials are SIM cards for mobile telephones, newer credit cards and SecurID Cards [1] for VPN access. The credentials are stored inside these hardware devices, which can only be used through specially designed hardware and software interfaces. The internal design of such hardware devices may be publicly known or hidden. Usually the hardware devices are made tamper resistant. Some devices destroy the credentials and themselves if the detect any tamper attempts.

  Hardware credentials may also use PIN for access control mechanisms. The PIN is also vulnerable to guessing attacks but obtaining the PIN does not yield the secret, though the adversary may be able to use the credentials. Hardware tokens are more secure than software credentials as they offer protection against malware, but are also more difficult and expensive to use in comparison to software credentials. Hardware credentials exhibit inflexibility and upgrading hardware (or credentials) is inconvenient. Further, distribution of hardware tokens is also a big problem. Generally, each hardware token contains credential that caters to only one application. In mobile phones, for e.g., it is possible to use only one SIM at a time, thus making it impossible to access multiple services simultaneously.

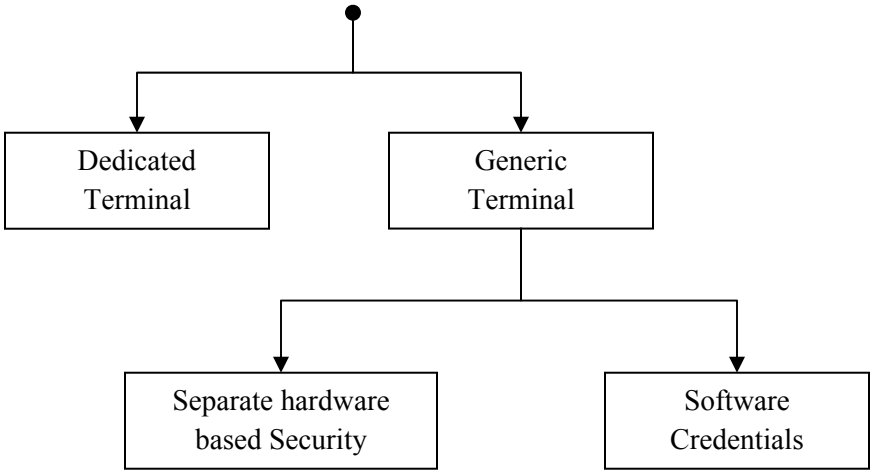

Figure 1: Existing Credentials Storage Systems

The current means of storing credentials as hardware tokens or software tokens is unsatisfactory. However, general-purpose secure commodity hardware modules such as TPM [7] [7] is becoming

widely available. The problem addressed in this thesis is how to design a scheme that uses such commonly available hardware to securely store and use the credentials while addressing the shortcomings mentioned above.

1.3 Organization of this thesis

The rest of the thesis is organized as follows: Chapter 2 presents the necessary technical background for this thesis. Chapter 3 defines the evaluation criteria that allow us to compare our work with other related works. Chapter 4 describes the general architecture of the system used for securely storing and using credentials using general-purpose secure commodity hardware modules. Chapter 5 deals with the implementation of the architecture in the two different platforms. In Chapter 6, the observations and analysis of the system is discussed followed by concluding remarks in Chapter 7.

2 Background

2.1 Secure hardware modules

In this section, we describe some of the prevalent hardware modules that are used for secure storage and execution. Until a few years back, such specialized hardware that assisted in cryptographic computations was very expensive and not common. With recent advancements in technology, these equipments have been much cheaper and more commonly available. Many leading laptop manufactures have started shipping their notebooks with secure environment such as the TPM [7] embedded on the motherboard.

2.1.1 Trusted Computing

Trusted computing is a technology developed and promoted by Trusted Computing Group (TCG) [8]. TCG aims to provide a secure platform that can be trusted upon for critical operations needed by organizations and individuals. The purpose of the TCG is to develop, design and promote trusted computing and security technologies including hardware building blocks and software interface specifications [8]. Supporters of this technology claim that it makes computers safer and less prone to attacks and malware, thus making them more reliable, for both end-users and service providers.

Trusted Platform Module (TPM)

TCG has designed the Trusted Platform Module (TPM) [8] to achieve the goals mentioned above. TPM is a hardware component that provides a mechanism for secure storage of digital keys, certificates and passwords. TPM chips are used to protect key operations and other cryptographic tasks such as key generation, signing, encryption, decryption, etc that would otherwise be performed in an normal operating system environment.

The operating system and applications can access these features only through a well-defined interface. For example, an application may encrypt some data with an RSA key residing inside the TPM but may not be allowed to read the content of this key.

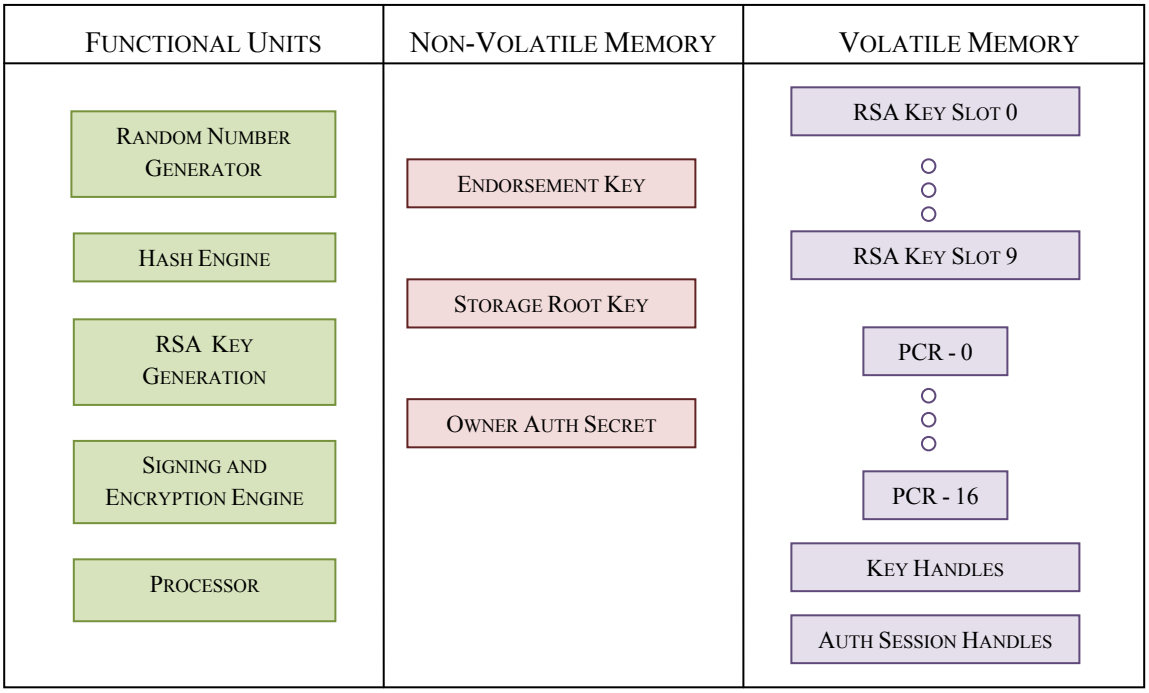

Figure 2: TPM Chip [9]

Figure 2 illustrates the inner details of the TPM. The TPM offers cryptographic functions such a hardware random number generator and hash computing algorithm. The TPM chip is made tamper resistant and designed to avoid the possibility of extraction of keys by hardware analysis. TPM contains a set of 20-byte registers called Platform Configuration Registers (PCR) that store the cumulative digest of the state of the system. The PCR 'extend' mechanism is an exclusive method to update the values of the PCR. It is expressed as:

$$PCR[n] \leftarrow \text{SHA-1}\ (PCR[n] \,|\, \text{measured data})$$

n: number of the PCR being updated.

A Stored Measurement Log (SML) is responsible for maintaining an ordered database of integrity changing events; the PCRs store digests of the measured data while SML provides an organized storage for the events taken place. As SML may grow quite large, these may be typically stored outside the TPM. Consistency is maintained as the PCRs are internal and may be modified only by the 'extend' mechanism and can show any evidence of tampering, thereby protecting the SML. The integrity protection of the SML comes from the fact that the sequence of events in the SML can be rehashed and compared to the value denoted by the PCR.

Key Concepts of TCG/TPM

The major features offered by a TPM based on TCG are remote attestation, sealing and binding. It is also used for identification, where the service provider may like to identify and authenticate the end-host before starting a transaction. Trusted Computing Group [7] [7] encompasses five key technology concepts, all of which are required for a fully trusted system.

1. **Endorsement:** Each TPM contains an RSA key called the Endorsement Key (EK), which may be initialized inside the chip only once. This may be done either by the manufacturer or by the user. The private part of the key is guaranteed to never leave the chip and is used to provide proof of origin.

2. **Secure Input and Output:** It is relatively easy for malicious software to intercept information as it travels between a user and a software application. Secure I/O uses a protected and verified channel using checksums (assisted by software and hardware) to verify that the software performing the I/O has not been tampered with. Although effective against software attacks, Secure I/O does not provide much protection against hardware-based attack.

3. **Memory curtaining / Protected Execution:** TPM provides full separation of sensitive areas of memory, for e.g. memory where cryptographic keys are stored. Inside the TPM, a process can be executed with the assurance that no other process can interfere with it. Operating system also does not have full access to the TPM's memory, so that secrecy and privacy is maintained even if the operating system has been compromised.

4. **Sealed storage:** TPM can be used to encrypt data using a key that is bound to an assessment of the software and hardware used. The key is extractable only when the state of the system returns to the same state to which the key was sealed.

5. **Remote attestation:** Remote attestation is a means to create a non-forgeable summary of the software of the computer, allowing a third party to verify that the software on the computer has not been compromised. Service providers may use this feature to identify and authenticate the end-host before starting a transaction. It works by having the TPM generate a certificate stating the PCR values that may represent the current state and the past history of the system. The user can present this certificate to a remote party as evidence that their computer has not been tampered with. Remote attestation information is generally encrypted with a public-key so that secrecy is maintained between the entity that presented the attestation and the entity that requested the attestation.

To maintain anonymity (as EK uniquely identifies a platform), TPM generates another RSA key pair called Attestation Identity Key (AIK). The AIK's private key pair also never leaves the TPM. The AIK public key is sent to a trusted Certificate Authority (Privacy CA) that knows the public

key of the TPM's EK. The Privacy CA can verify the origin of the AIK as it is signed by the EK of the TPM. The Privacy CA then issues a certificate on the TPM's AIK. This certificate is presented by the TPM to a verifier to authenticate itself with the AIK. Thus, the TPM can use different AIKs each time to sign PCR values thus making the attestation un-linkable. This method is described in more detail in [10].

Secure and Authenticated boot

Secure boot is a process that validates the integrity of the platform and ensures that the platform always boots into a previously "known good" configuration. Secure boot is based on the transitive trust model that states that the integrity of a layer can be guaranteed if and only if the integrity of the lower layer is checked and that the transition to the higher layer occurs only after the integrity checks on the next layer are complete [11]. The resulting integrity chain then warrants the system's integrity. The Root-of-Trust-for-Measurement (RTM) contained in the trusted platform constitutes as the first layer in the chain for secure boot [12]. This immutable code is executed first, even before the BIOS, implying the dependence on TPM aware BIOS, such as [13], for secure boot. During the secure boot, the RTM performs a cryptographic measurement of the platform's code objects and compares the measured value with a 'previously known' value computed during a previous boot that was agreed to be trustworthy. If the two values do not match, the boot is aborted. The 'previously known' state of the system is a set of PCRs that may be stored as a table in unsecured persistence storage. A protected non-volatile register – Data Integrity Register (DIR) inside the TPM chip stores the hash of the table, thereby ensuring the integrity of the PCR values stored in unsecured persistence storage. ARM TrustZone [14] provides secure boot functionality that ensures that the system always boots into a predefined secure state.

Authenticated boot shares the same concept as secure boot except that it does not stop the device from booting into an arbitrary non-secure state. It allows a mechanism for an external party to verify the integrity of the system through remote attestation. The TPM follows a similar authenticated boot concept as explained in [11]. Trusted boot is a combination of secured boot and authenticated boot that allows an external party to verify that the system has indeed booted into a predefined secure state.

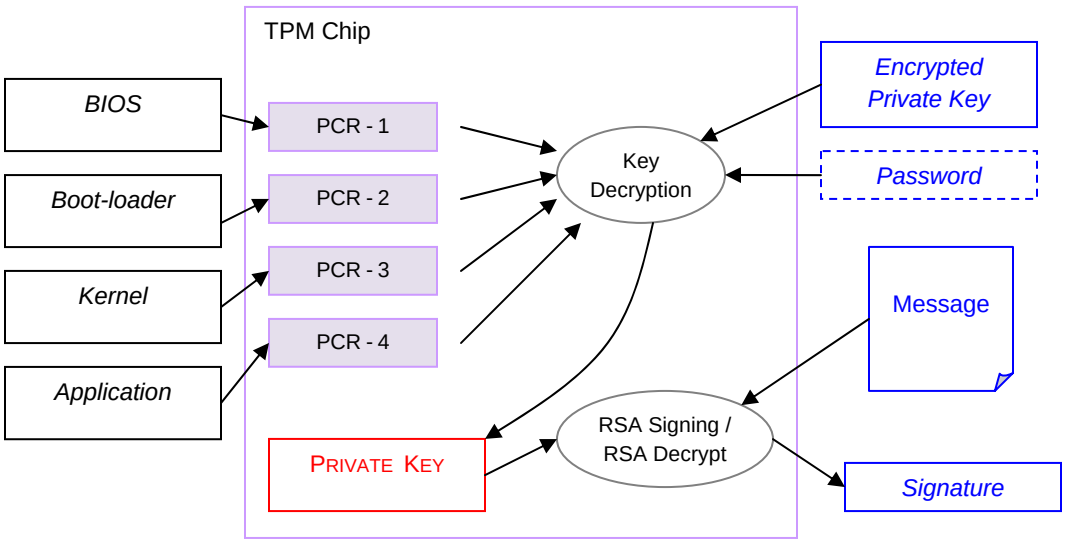

Figure 3 Secure Boot / Signatures [9]

Figure 3 illustrates a part of the secure boot. Prior to executing the BIOS, a cryptographic hash is computed and verified against a stored digital signature for the BIOS code. If the signature is valid, the control is passed to the BIOS. After another integrity check over the boot-loader, the BIOS passes control to it. The bootstrap code finds the bootable device and proceeds to verify the boot block, which then proceeds to verify the kernel image. The kernel verifies the application space of the operating system before giving control to it and hence completing the boot process.

2.1.2 ARM TrustZone

ARM's TrustZone [15] is an initiative to provide trusted computing in embedded systems with minimal impact on important performance criteria in embedded systems such as power consumption, processing capability or size.

While off-chip hardware solutions such as separate co-processors enable acceleration of computationally extensive cryptographic operations, it adds to the cost, size and power consumption. Further, the data flowing in the bus between the core processor and the off-chip device is vulnerable. One example of such a hardware solution is Subscriber Identity Modules (SIM) cards that have been predominant in the handset domain.

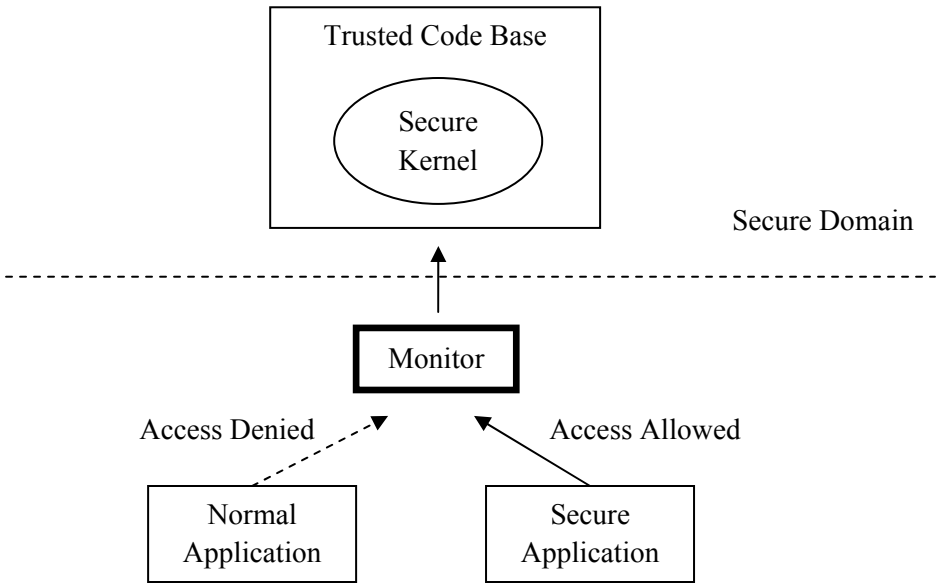

Figure 4: ARM TrustZone's Parallel Secure Side [16]

TrustZone consists of a single processor with a separate secure area. It provides a separate access to sensitive information and other hardware and software portions of the ARM-based system-on-chip (SoC) [16]. The security is achieved by creating a secure domain using "trusted code base" residing in the secure area of the processor. This ensures the trustworthiness of the system starting from boot (similar to secure boot described earlier in Section 2.1.1). The trusted code base is protected by implementing a separate secure domain as shown in Figure 4. A secure monitor triggered by the Secure Monitor Interrupt (SMI) switches between secure and non-secure regions. Before switching to the secure mode, the monitor disables all virtual memory and clears all the data and code caches. Only applications identified as trusted are allowed access to the secure domain.

TrustZone provides features such as space for secure storage of data, performance enhancement with full bus-bandwidth to all storage areas and allows customization and upgrades to secure system even after SoC finalization. TrustZone also provides a Trusted Interpreter based on Small Terminal Interoperability Platform Specifications (STIP) for execution of secure code that is independent of operating system or device [15]. Figure 5 describes the software architecture when using ARM TrustZone CPU. The demarcation between the normal and secure side is done by adding an extra flag (the S-bit) throughout the architecture. This flag can be accessed only through the secure monitor mode and it defines which parts of the system (CPU core, memory, peripherals, etc.) are in secure mode.

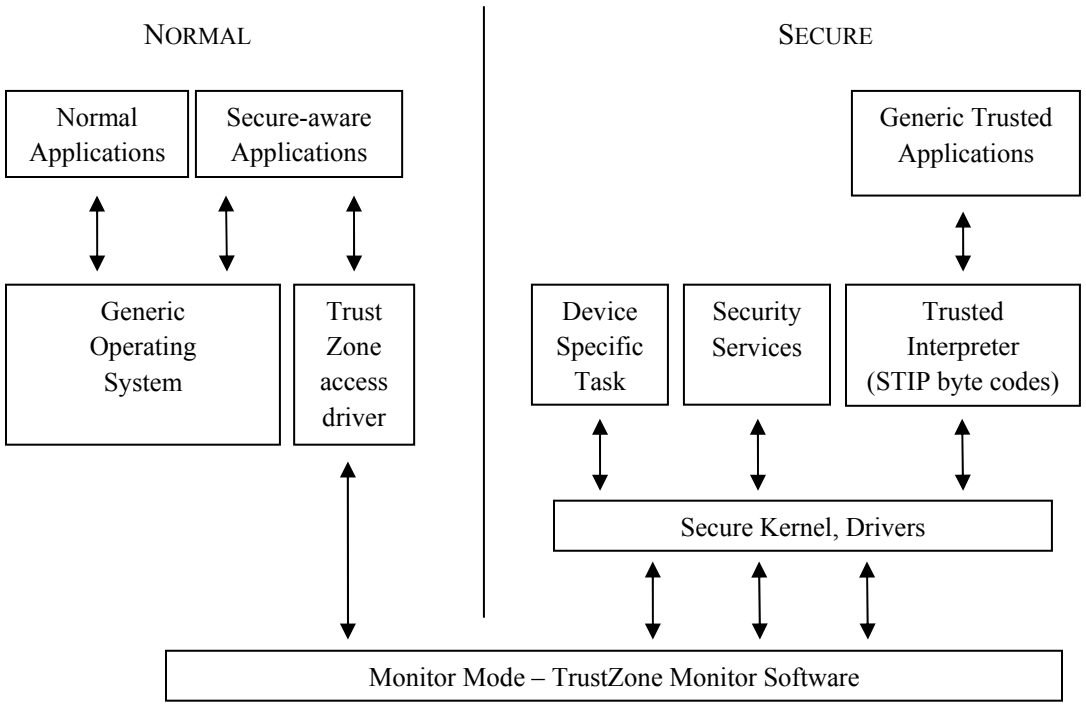

Figure 5: ARM TrustZone Software Architecture [15]

Trusted parties such as service providers may develop secure applications that are to be executed in the secure side. Alternatively, security aware applications may also be able to access the secure side of TrustZone using the TrustZone access driver. Secure platform independent applications such as SIM-Lock may bypass the operating system completely by invoking the Trusted Interpreter.

2.1.3 Java Cards

Smart cards are small computing devices that act as tokens to enable services that require security. Java Cards [17] platform provides a secure execution environment that isolates applets, running inside the card, from each other. Each applet is given a private partition of the card memory and a firewall feature provides a detailed control over shared data between the applets. The Java Card virtual machine further separates the applications from the underlying hardware and operating system.

Java Card has a "split virtual machine" architecture [17]. The off-card virtual machine executes on a workstation computer and links the classes by resolving the references. The off-card virtual machine also converts the code into a format suitable for execution on the Java Card and provides a mechanism to verify that the content of the smart card conforms to the Java Card specifications. An applet created may be cryptographically authenticated to ensure the contents are not changed before the installation of the applet on the card. The on-card components consist of a Java virtual machine, a runtime environment and Java Card APIs.

The Java Card platform leverages a few security benefits from the Java programming language and adds some significant security enhancements. Creating applications in Java language is also easier than developing applications using traditional embedded programming languages. Java Card provides transaction atomicity that guarantees that all memory changes are written to persistence storage at once, thereby maintaining consistency of data, in the event of sudden removal of a card while a transaction is in progress [17]. Java Card also provides classes that can be used for cryptographic computations. These classes include key encryption and decryption algorithms, signature generation and verification, message digest, random key generation and PIN management.

Java applets with cryptographic signatures can be distributed securely to third party developers and smart card operators. A Java Card Technology Certification Kit (TCK) [17] is supplied for testing the compatibility and to ensure conformance to Java Card behavior standards.

2.2 PKCS#11 – Cryptographic Token API

Public Key Cryptography Standards (PKCS) are developed by RSA Security Inc. [18] in co-operation with other organizations and sources. PKCS#11 [19] defines cryptographic Token Interface Standard APIs and was designed to provide a standard interface between applications and portable cryptographic devices such as smart cards and allow simultaneous sharing of the devices by these applications. Some applications, libraries and protocols have been built upon PKCS#11 and key products such as Mozilla and SSL hardware accelerators have long supported this standard.

In PKCS#11, a token is a device that stores objects (e.g.: key data and certificates) and performs cryptographic operations. This is a logical classification, as one physical device may comprise of distinct logical tokens. Token objects are permanent and are accessible to all applications connected to the token and the user must authenticate and establish a session before using a token. Session objects are volatile and exist only during the duration of the session between the application and the token. Each object, such as, a key, has attributes that identifies its access as public, private or secret. Private and secret keys may have properties that may make them sensitive and never extractable. Sensitive keys cannot be revealed in plaintext outside the token and non-extractable keys cannot be disclosed off the token at all.

PKCS#11 describes two categories of users. The administrators may only perform operations such as changing passwords. Normal users are allowed to perform only cryptographic operations. The standard mandates that all users must login before they can access the objects through PIN or through other custom augmented mechanisms.

PKCS#11 provides a set of APIs for accessing functions for access control mechanisms such as “login” and changing PIN of a token. The standard further provides a set of APIs for key management functionality such as:-

- Secret key generation
- Key pair generation for public/private keys
- Wrapping (encryption) of private or secret key
- Decryption of the wrapped key
- Deriving a key from the base key.
- Keyed Encryption of data
- Keyed Decryption of data
- Signing and verification of data
- Hash algorithms such as SHA-1 and MD5
- MAC computation

2.3 Platform Security

Operating systems are often the critical point of failure and are thus, fundamental to the security of the overall computing system. Mandatory Access Control (MAC) is an essential step towards providing a sufficiently secure operating system. Discretionary Access Control (DAC) based on identity and ownership in operating systems such as Linux is inadequate until every program continues to inherit all the privileges associated with its owner [20]. The following section describes some of the relevant enhancements to native features of the operating systems to provide a stronger platform security.

2.3.1 SELinux

SELinux [21] in an implementation of Flask – a flexible and fine-grained mandatory access control architecture [22] by National Security Agency (NSA) [23]. SELinux is implemented using Linux Security Modules (LSM) [24] and a Linux kernel integrating SELinux enforces mandatory access control policies that confines user programs and system servers to just the minimum amount of privilege needed for functioning. This eliminates any ability of a compromised process or object to cause harm to other components.

SELinux security server maintains three security attributes – *identity, role* and *type*. Combinations of these attributes form security contexts, which are used in computing access decisions. Each process is associated with an *identity*. Changes to the *identity* are strictly controlled. Each object in the system is assigned a *type* that determines the access permissions to other types. *Roles* determine allowable user actions. Users are assigned *roles* and the actions and transitions between roles are governed by the Role-based Access Control Policy (RBAC). *Role* transitions always require explicit user consent and authentication. SELinux defines *domains* that can be entered by set of *roles* where the Type Enforcement (TE) assigns the actual permissions.

The policy decision-making logic is encapsulated within a single Security Server component. Access Vector Cache (AVC) component minimizes the performance overhead of the access controls by caching the access decisions computations obtained from the security server. The policy enforcement code is integrated into the subsystems (e.g. the process management code, file system code, IPC code, etc) of the operating system. SELinux provides flexibility by keeping the policy design making logic separate from the policy enforcing logic.

However, SELinux is designed to enforce mandatory access controls in systems and configuration of such policies are prone to human errors [25]. It does not protect against kernel vulnerabilities and it lacks an efficient resource consumption control. The SELinux code is quite complex and increases the chances of bugs in the SELinux package itself. To achieve good level of protection, SELinux requires that applications be recompiled. This is a problem as recompilation may not always be feasible on a system that has been already deployed.

2.3.2 LIDS – Linux Intrusion Detection System

LIDS [26] is another enhancement for the Linux kernel that implements security features that are not part of the native Linux kernel. The need for LIDS arose as the current Linux setup has a problem of having a "root" account that can interfere with any aspect of the system. LIDS offers mandatory access control, port scan detectors and process and file protection from root.

LIDS is a patch on the Linux kernel and installs itself as a kernel module. The version of LIDS for kernel 2.6 is based on LSM [24] and it cannot work with other security modules like SELinux (discussed in Section 2.3.1). This could be possible as both LIDS and SELinux modify the kernel and as such may require contradicting functionality in the same feature or module in the kernel.

LIDS provides protection for several boot stages – boot, post-boot and shutdown and one may specify an ACL for each stage. One can give processes or binaries certain capabilities for a particular stage or globally throughout the system. For example, by using LIDS, it is possible to disable a kernel capability for all processes. One may then define an exception list in an ACL that allows only specific programs to utilize those kernel capabilities. Additionally, LIDS may be used to make all system files read-only from super-user. The system files may be modified only after a special authentication with LIDS.

2.3.3 Symbian Platform Security

Symbian OS v9.0 onwards introduced Platform Security [27] that addresses the problem of ensuring the security and integrity of data and applications. Application developers must take this concept into account when designing and developing Symbian C++ applications. The security is achieved with the introduction of the following:-

1. **Capabilities:** The APIs offered by Symbian are divided into different groups depending upon their sensitivity. Each executable in Symbian operating system is assigned certain capabilities, which determines its access rights to the APIs provided by the operating system. This implies that any code that needs to access some capability protected functionality must go through an authorizing process [28] in order to gain the privilege of using the capability. Authorizing the code confirms the trustworthiness of the code.

2. **Secure IDs / Vendor IDs:** All executables from Symbian v9.0 onwards must possess a Secure Identifier (SID), which is assured to be locally unique. There may be two kinds of SIDs - normal SIDs and protected SIDs. As an example of its use, SID may be used to protect an API by restricting the access to some particular applications. A protected SID, which is guaranteed to be globally unique, requires the presence of a digital certificate authorizing its use. Similarly, Vendor ID (VID) is used to determine the origin of the executable. VID if required, must be always signed. This feature is beneficial for device manufacturers who may want to restrict access to the API released by them.

3. **Signed Software:** Symbian requires that software with capabilities must be signed after a certification process as described in [28]. Although an unsigned application is allowed to be installed and executed, it would not be able to access the resources that mandate possession of certain capabilities. This makes it impossible for untrusted applications to access protected resources of the operating system.

4. **Data Caging:** The new platform security provides isolation of process data, i.e. other running processes cannot access other's runtime memory. Each process additionally has an exclusive directory on persistent storage that can only be accessed by itself or by those with higher capabilities.

2.4 Trusted Path

Typically, PIN and fingerprints are used as authentication mechanisms for local access control. Therefore, it is important that software receiving this authentication information ensure that the correct party has presented the credentials. A malware may spoof the genuine software and phish the local access control credentials. Once obtained, it may present these credentials to the software to gain access to the resources.

A solution to this problem is a "trusted path". A trusted path is a mechanism that provides confidence that the user is communicating with the party the user intended to communicate with and ensures that the attackers cannot intercept or modify information that is being communicated [29]. Before the user submits his credentials, the application must authenticate itself to the user to establish the trust that the credentials are indeed being presented to the genuine entity. It must also be ensured that the correct channels and mechanisms are used to input the access control information.

One method to implemented trusted path is to have a non-forgeable key-press before login. For example, in the Windows environment, the Ctrl + Alt + Del sequence always displays a trusted user interface that is guaranteed to belong to the operating system. The operating system does not allow any application to trap this particular key sequence. This ensures that no malicious application or process can trap this key sequence and spoof the UI to get passwords from the user.

Another approach for implementing a trusted path locally is to control a separate display that only the login program can perform. For example, if only trusted programs could modify the keyboard lights (the LEDs showing Num Lock, Caps Lock, and Scroll Lock), then a login program could display a running pattern to indicate that it is the real login program. Unfortunately, LEDs cannot currently be used to confirm a trusted path since in operating systems such as Microsoft's Windows or Linux normal users can change the LEDs. Moreover, it is highly unlikely that the users would actually check the status of these LEDs. Such LEDs are generally unavailable on mobile devices and this scheme thus, cannot be applied universally.

2.5 Related work

There has been considerable effort in the area of storing and managing credentials. Many of them offer security by using assisted hardware such as the TCG/TPM. Some of the projects are mentioned in this section.

2.5.1 Solutions using secure hardware

Central Credential Repositories

Lorch, Basney, Kafura [31] use secure hardware for protecting private keys in a central credential repository. The authors discuss about the vulnerability of the private key in a PKI infrastructure due to improper "key hygiene" which includes failing to encrypt the key with a strong password and storing the key on an insecure file system. Due to the distributed nature of PKI based solutions, it is difficult to enforce a key protection policy. The problem is further exacerbated in grid computing where users access services from different computers requiring the need for storing the credentials on each computer.

The authors examine the possibility of using a separate proxy server that issues short-term credentials derived from the original credentials and a user supplied passphrase. This way a client only obtains short-term credentials, which are given to it by the proxy server. Even if the credential is revealed, the risk is lessened, as the credential is valid for a short time period only. This scheme is applied for storing private-keys of users in a separate credentials repository. This makes this proxy server the central point of control. The authors propose protecting the users' private keys by using a crypto-processor to generate the public-private key pairs and store the private keys inside it. This yields three important features: 1) by storing the keys inside the

crypto-processor, key-hygiene issues are no longer relevant. 2) Administrators and third party cannot access the keys directly. 3) Tamper resistant properties of the secure hardware makes extensive physical security of the server unnecessary.

The above mentioned work was extended by Marchesini and Smith [32] to use secure hardware as key-stores both at repositories and at clients, if available. They introduce X.509 based proxy certificates, which are issued along with the short-term credentials. The proxy certificates contain the conditions in which the short-term credentials were created. Thus, the peer can make a decision whether to trust its counterpart based on the environment information present in the proxy certificate. If available at the client, this environment "measurement" may be attested using secure hardware such as Bear [33] based on TCG specifications [34]. The policy framework is also created that allows peers to express how their private key should be used in an XML based language. The policy framework, for example, may be used by a user to restrict the cryptographic operations that may be performed using his private key in the repository.

Such a method for secure storage of credentials is beneficial as it is compatible with existing PKI services. The private keys stored in the secure hardware and are thus protected from dictionary attacks. However, such a scheme required an additional infrastructure in the network. There is also the problem of protecting the access control credential that is used to login to the server to obtain the short-term credentials.

IETF's Securely Available Credentials (SACRED) work group in [35] and [36] has described the requirements and protocol for transfer of credentials from a secure repository. The trust between the repository (Credential Server) and the client is established using mutual authentication based on shared secrets. This trust establishment may be carried out by password protocols described in [37], [38], [39] and [40]. They further prescribe formats as specified in PKCS#12 [41] and PKCS#15 [42] standards to securely transfer the credentials from one location to another.

Microsoft - Bit Locker Drive Encryption

Microsoft's BitLocker [43] is a system involving TPM to protect data and is available in the premium editions of Windows Vista. The main goal of this project is to protect data on the operating system volume of the hard drive. To achieve this, BitLocker uses the v1.2 TPM security hardware to secure the encryption keys and prevent software-based attacks on the system integrity and security of the data. It includes integrity checks on critical early boot components. BitLocker uses the TPM to collect and store measurements from multiple sources within the boot process to create a fingerprint of the system. This fingerprint remains the same unless the boot system is tampered with. After the integrity of the boot process is proven, BitLocker uses the TPM to unlock the rest of the data.

The volume contents are encrypted with a Full-Volume Encryption Key (FVEK), which in turn is encrypted with a Volume Master Key (VMK). The VMK is sealed to the TPM to the corresponding fingerprint of the system, securing the system indirectly. The addition of the VMK facilitates

easy re-keying in the event of a security breach in the trust chain. The trust is in the fact that the VMK is decrypted only when the integrity of the system is ensured.

However, such a scheme encrypts the whole drive and is not suitable for encrypting individual credentials.

Intel - Manageable Identities

The Manageable Identities [44] research project from the Intel Systems Technology Labs is building a trusted access environment that fits into the existing network-based identities. The aim of this research project is to support coexistence and integration of hardware and software identities with user and provider content. The project involves sharing, transporting and managing of identities between different devices, depending upon policies dictated by service providers and users.

A TPM [8] enabled PC client is used to authenticate securely on public wireless spots using SIM credentials through a Bluetooth enabled handset. It is possible that a remote attestation is performed by the handset to verify the trustworthiness of the PC client before the SIM credentials are transmitted from the handset to the PC Client. Wireless operators could use remote attestation to verify the platform on the PC client and the SIM credentials presented by it. Such a mechanism could allow for SIM reuse for secure authentication on Wi-Fi networks. It is proposed that service providers and carriers could use such a mechanism to provide a consolidated bill for WLAN and WWAN usage.

Other projects

There have been considerable other projects related with providing security through Trusted Platform concepts. A TCG-based integrity measurement architecture in Linux is designed and implemented by Sailer, et al [12]. In this project, the authors have demonstrated an integrity measurement architecture where all executable content is measured by the TPM before execution. The authors have applied the measurement architecture to a web server application to detect and protect against undesirable invocations such as rootkit programs. Such a scheme protects the system from malware attacks and allows third parties to verify the integrity of the system before initiating a transaction.

TPM technology has been further extended into the mobile and cellular area by Kuntze and Schmidt [45]. The paper discusses business scenarios centered on the mobile network operator. The authors suggest using the functional restriction (SIM Lock) to implement a pre-paid mobile phone service. The functional restriction is secured by using trusted boot concepts. The mobile phone stores the total pre-paid cash value in its trusted storage area, which is modifiable only by trusted software. The access to the network is SIM based, but the Trusted Computing provides attestation and authentication that assures the network operator about the pre-paid cash value stored. According to the authors, this can avoid a centralized accounting for the prepaid cards,

realizing a cost benefit to the mobile network operator. It also describes the concept of virtual SIM cards (VSIM) and how the authentication on the GSM network by TMSI can be supplemented by attestation through the TPM's Attestation Identity Key (AIK) [7] [7]. However, the authors do not mention about any protection against rollback attacks, in which the value of the pre-paid card is reverted by restoring from a previous backup.

In another paper [46] by Shane, Amit and Kenneth, Trusted Computing is applied for securing Peer-to-Peer (P2P) networks. The absence of stable verifiable peer identities is mentioned as the central challenge in providing security services. The features of TPM such as Direct Anonymous Attestation (DAA) [47] are used to enforce the use of stable pseudonyms [48]. This prevents users from using multiple pseudonyms – *pseudospoofing* [48] or stealing other users' pseudonyms – *pseudotheft* [46]. This aims to prevent attacks such as *shilling* – generating false bids in online auction systems. Although this work is not related very much with our work, it does illustrate one possible use of TPM in a distributed network.

2.5.2 Applications storing passwords

Several applications store the passwords so that users may not have to type their passwords repeatedly. Some applications also generate stronger passwords from weaker but more easily remembered user-chosen passwords. In spite of the shortcomings of the password based authentication mechanism (prone to dictionary, phishing, social engineering attacks), it remains as the most widely adopted authentication mechanism. In this section, we examine some of the projects that aim to address the usability and security vulnerabilities of passwords based mechanisms.

PwdHash

PwdHash [3] is a browser plug-in that applies a cryptographic hash to generate strong passwords. The hash is computed from the user's entered password and the site domain and the new stronger password is sent to the target site. Such a scheme does not require any changes at the server side. Users do not know the actual password used and thus must always use the PwdHash application to login. On clients that do not have the plug-in installed, the authors provide a website where users may remotely generate the password again and use it.

For this scheme to work, the user must go to each individual website to change to the new password generated by the plug-in and may be repeated each time the user wishes to change the password. PwdHash also provides protection against phishing attacks by using a hash salt based on the domain name of the target website. This means that the password will not be correct if the user enters a password at a fraudulent site as the site's domain will be different.

Password Multiplier

Password Multiplier [4] intends to help users generate strong passwords for web accounts and is available as a plug-in for Firefox. Password Multiplier uses a single password (chosen by the user) to generate multiple passwords. A cryptographic hash is applied to derive the new stronger passwords from the username, password and domain name of the website.

Users must change each account's password in order to use the stronger protected passwords. However, for subsequent changes to a password for a particular website, the user must modify the label of the website provided by him for generation of the password. This implies that apart from the username and password, the user must remember the label for each website given by him. To prevent dictionary attacks, a two-stage process where the first stage comprises of a computationally intensive hash function is used. This also introduces a delay thereby discouraging an attacker from testing combinations of master password. Unlike *PwdHash,* this plug-in does not provide any online service as an alternative for clients that do not have the plug-in installed.

PassPet

PassPet intends to improve protection from attacks related to using password by combining various schemes such as password hashing, custom names, password strengthening and UI customization [49]. PassPet attempts to reduce the burden on user's memory by making the user remember only one master secret. Each website is assigned a site-label by the user that is used to uniquely identify the site. The tool identifies sites by their SSL certificates or domain names to help prevent phishing attacks.

PassPet also utilizes a trusted path concept (see Section 2.4). On first use, the user is presented with a randomly chosen picture for the "pet animal" and another randomly chosen name for it. This information is presented each time the user is asked to input the master secret as shown in Figure 6. Thus, associating a persona with each user this makes it very difficult for a rogue application to spoof the PassPet tool.

Stronger passwords for websites are generated using master secret and site-labels by using computationally intensive hashing as described in [4]. User can generate new passwords for individual websites by changing the site-labels assigned to them. However, the user must visit each service's website to change the password. MAC based integrity protection is provided to the list of site-labels stored in the local persistence storage.

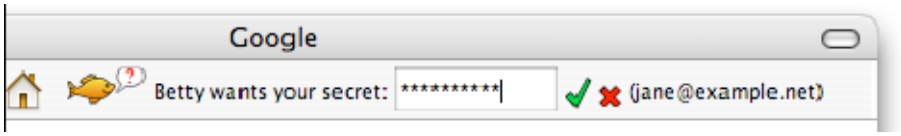

Figure 6: Trusted UI in PassPet Tool [49]

To regenerate correct passwords on another computer, PassPet needs to know the master address, master secret, site-labels for each website and two more tokens. The tokens are stored on the server and are not known by the user. Secure Remote Password (SRP) protocol [50] is used to retrieve the tokens based on the value of master address, master secret, site-label supplied.

Such a method of storing passwords is good as it implements a trusted UI and is compatible with existing services. However, the master password is vulnerable to dictionary attacks.

2.5.3 Single Sign-On

Another approach to manage credentials is to reduce the number of credentials themselves. Single Sign-On(SSO) is designed to centralize the authentication information on a single server, not only for the users' convenience but also to offer increased security by limiting the number of times sensitive information must be stored. It permits a user to authenticate only once to an Authentication Service Provider (ASP) and allows the user to subsequently use disparate Service Providers (SPs) without necessarily needing to re-authenticate. The information about user's authentication status (Authentication Assertions) is handled between the ASP and the desired SP transparently to the user.

Microsoft's Passport [51] SSO service is an example of a growing trend towards the use of web-based single sign-on that allow users to register their financial information once and shop at multiple web sites. It was one of the first SSO service implemented but it was not successful as the implementation required additional infrastructure. Privacy issues also surfaced as service providers were unwilling to share customer information with others.

Liberty Alliance Project

The Liberty Alliance [52], a consortium of more than 150 organizations, was established in 2001 to create an open authentication platform that would enable people to use a single sign-on for participating web sites. The federated identification push was a response to Microsoft's closed Passport technology [51]. Liberty supports SSO both within and across federation of service and identity providers. A single logout provides synchronized session logout functionality across all sessions that were authenticated by a particular identity provider.

An Identity Provider (IdP) is a Liberty-enabled entity that creates, maintains, and manages identity information and provides authentication to other service providers within a federation of service and identity providers. This does not require the users' personal information to be stored centrally. Federation works by having a Liberty-enabled client that knows about the identity provider of the entity that wishes to access a service. A Liberty-enabled proxy is an HTTP proxy that emulates a Liberty-enabled client. Anonymity is maintained by using arbitrary names assigned by the identity or service provider. This ensures that the pseudonym has meaning only in the context of the relationship between the relying parties.

In Liberty, the user is given security token the first time he logs onto the IdP. When the user next time wishes to access the service provider, he provides the Service Provider (SP) with the security token for authentication. The user is then asked to federate the accounts and is redirected to the IdP. The IdP issues the security assertion to the user in the form of an SAML [53] document. The user gives this to the SP as a SOAP message and is authenticated.

One apparent advantage of Single Sign-On is that only one credential is required to access multiple services. The disadvantage is that services need to be SSO compliant. There is also a problem of securely storing this one credential. SSO works on the assumption that the service providers are willing to trust a central authority. This may not always hold true and can be a deterrent to SSO.

Microsoft - CardSpace

Windows CardSpace [54] (formerly known as "Info Card") by Microsoft is a .NET technology component that provides access to the user's identities in a secure and easy manner through a desktop interface. The CardSpace project aims to provide a consistent control over a user's digital identity as a replacement to the password based web authentication mechanism. The project claims that it improves the user's confidence in the identity of the remote application. CardSpace is a user interface that presents users with an identity selector (a palette of cards) that is used to authenticate to various network resources. The interface is popped out in front of the desktop suspending other functionality on the desktop and shows the user, which kind of credentials are required to access a particular service or resource.

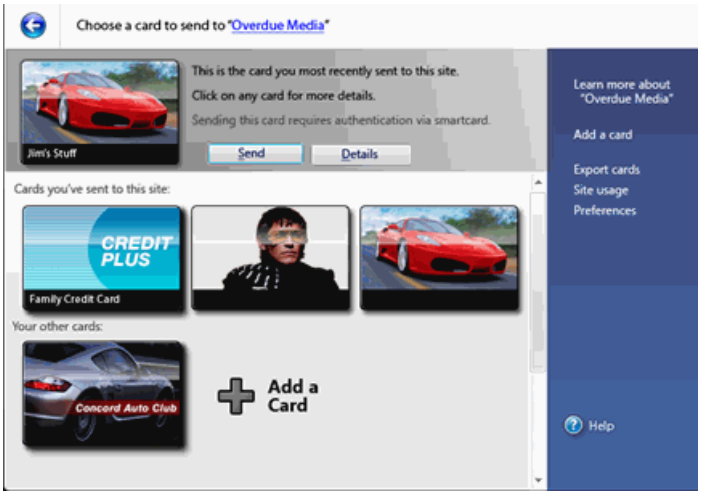

Figure 7: CardSpace Identity Selection Screen [55]

Each card represents a digital identity that the user can present to a relying party as shown in Figure 7. This card is actually created by an identity provider, and then installed on a user's machine. By selecting a particular card, the user chooses to request a specific security token with a specific set of claims created by a specific identity provider.

Authenticating users with security tokens than with passwords is a much more secure way of authentication, as the latter is prone to phishing and spoofing attacks. Typically, a website proves its identity by using certificate used for SSL communication. However, SSL certificates only prove that a given website has the particular DNS name. An attacker can spoof a website and use the certificate issued to him for the DNS name he owns. This way, it would be impossible to detect a phishing attack, as the SSL certificate will be valid. Thus, proving identity through SSL certificates yields little protection. To address this issue, the CardSpace project attempts to create a new level of certificate that contains more information than a traditional SSL certificate, such as company logo, etc. This higher-assurance certificate will also be a more authoritative source of information because it will be more difficult to get and would require a stronger agreement with the authority that issues it. Both identity providers and relying parties can use this new certificate type to prove their identities to users of CardSpace applications.

3 Criteria

Existing pure software solutions do not provide adequate security, as they are vulnerable to malware and dictionary attacks. Pure hardware solutions are expensive and do not offer much flexibility. Inexpensive, general-purpose, secure commodity hardware modules such as TPMs are becoming increasingly available. Our objective is to design a system that provides the means to securely store credentials using general-purpose secure commodity hardware modules. In this section, we present the desired qualities of the system for achieving our goal stated above.

We want that our system should be robust and fail-safe. It should not leak the stored secrets in spite of a crash or system failure. Moreover, the system should provide protection from misbehaving applications such as Trojans and viruses. It must also prevent any phishing or spoofing attempts and thus protect against unauthorized extraction or usage of the stored credentials. Hence, we define our first criterion for evaluation:-

1. ***The system should provide protection from malware.***

The system should be oblivious of the semantics of the credentials and should allow different credential types to be stored and used. One credential type should not interfere or compromise the security of other credential types instances stored in the system. The system should be preferably flexible and allow new credential types without requiring any modifications to the existing system.

2. ***The system should permit multiple credential types.***

With the increasing number of services offered electronically, the number of credentials in use is only going to increase. Our system should thus allow multiple credentials to be stored without compromising the security and usability of the system. The system should allow multiple instances regardless of its credential types. This brings us to our third criterion:

3. ***The system should permit multiple credential instances.***

Credentials may as such be created by the user himself on the device, or created externally by third parties such as service providers or other untrusted parties. The system should permit provisioning of these externally created credentials and allow them to be stored securely. The system should also allow users themselves to create credentials and store them using the system. Hence, we state our fourth criteria:

4. ***The system should allow credentials from multiple sources such as users, service providers, corporate IT departments, etc.***

In the event of a device failure or device change, the system should allow for easy migration of the credentials from one device to another. It would be very inconvenient for the user to input all the credentials onto the new device. It should be possible to create a secure backup of the credentials without compromising on the security of the credentials in any way.

5. ***The system should provide a mechanism of a backup of the credentials***

It is not viable for services to change their authenticating functionality to accommodate a new scheme to store passwords. Usually services require users to change the credentials (such as passwords or passphrases) at their own service portals, thus requiring the user to change them at each individual site; this makes it impractical for users to shift to the new authenticating mechanisms. Further, the system should work with the existing protocols and should not require any modifications. The system should not break any functionality on the client either.

6. ***The system should be compatible with existing services such as servers, protocols and clients.***

User may also additionally want to access the services from more than one device. The usage of the credential should not be limited to only one device, unless mandated by the service provider or another authority. Storing the credential on one device should not disable access of the server from another device. The system should allow individual transfer of credential to another device installed with the proposed system. Additionally, the user should be able to access the service from a device that does not implement the proposed system. Hence, the system should have the following requirement:

7. ***The system should allow services to be accessible from multiple devices.***

Service providers may want that a service is accessed only a certain number of times or for a fixed time period. They may also mandate that a service is accessed only from a certain device or may want to limit the number of devices from which the service is accessed. Service providers may wish to enforce a policy restricting the usage and copying of the credentials and the system should incorporate access control checks such as PIN or fingerprints. Hence, the criterion:

8. ***The system should allow policy based restriction of the usage of credentials.***

4 Design

4.1 Onboard Credentials

Virtual security tokens fill up the void between the two extremities of hardware secured credentials and software credentials (discussed in Section 1.2). These virtual credentials are protected by a secure environment (see Section 4.1.1) that utilizes underlying hardware security features. It combines the ease of use and deployment of software credentials with increased level of security provided by the specialized hardware.

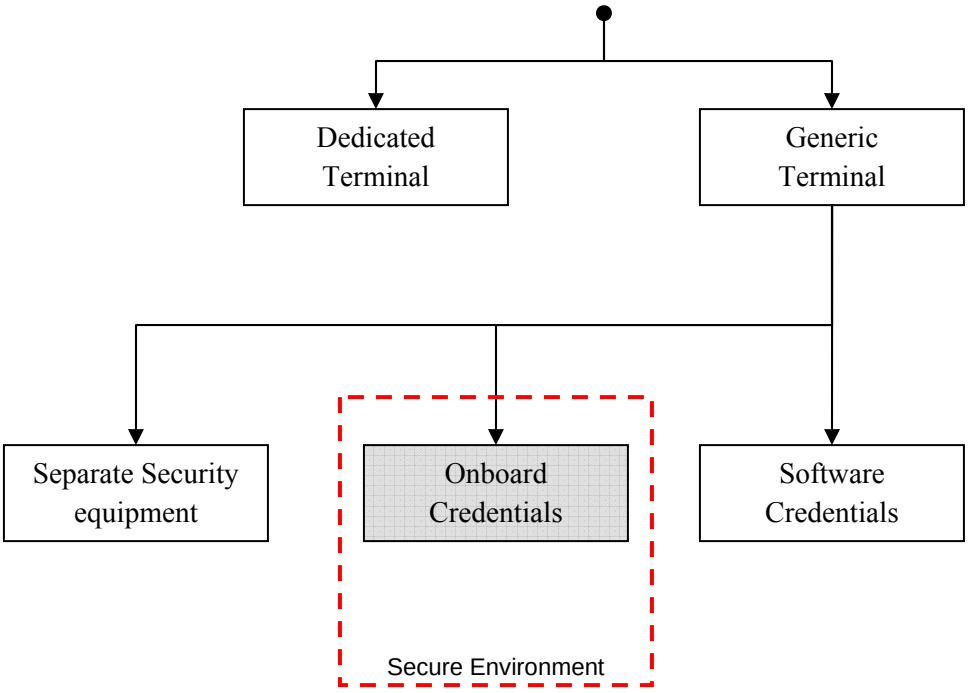

Figure 8: Onboard Credentials

4.1.1 Secure Environment

Secure environment at least provides secure persistence storage for keys and other data (at least 128 bits) and provides authenticated access to this secure storage. Additionally secure execution environment may be provided that provides per-process memory protection and ensures that a code running in a process does not interfere with or reads memory of another process.

Secure environment may be divided into three categories based on its constituents:

1. **Secure storage in hardware with secure execution environment provided by the operating system:** Secure hardware modules as such as TPM and ARM TrustZone provide secure storage and the operating system provides memory isolation and protection among processes.

2. **Both secure storage and execution environment in hardware:** The secure storage as well as secure execution environment is provided by using secure hardware modules such as ARM TrustZone.

3. **Both secure storage and secure execution environment provided by the operating system:** The operating system offers a security architecture that provides secure storage of data. Additionally it provides inter-process memory isolation of data as a means to provide secure execution environment. Example: Symbian Platform Security.

The fourth combination – with secure execution environment and operation system provided secure storage is of little use as generally the hardware, such as a microprocessor that supports secure execution environment is sophisticated enough to support secure storage of data.

4.2 Architectural Overview

In this section, we present an overview of the architecture for a credentials storage system based on secure environment. The architecture remains the same irrespective of the type of secure environment incorporated. A typical credential consists of a secret and a logic to operate on the secret. The credentials logic are expressed as scripts, which are dynamically loaded and executed (in an interpreter running in a secure environment) to encrypt the secret. The encrypted secret can be stored in a repository that may be kept in the normal user-space region of the operating system.

4.2.1 Usage

Adding credentials

In and the subsequent paragraphs, a typical usage of the Onboard Credential system is discussed. To best illustrate the usage of the Credential Server, we first take the scenario in which credentials are created or sent to the system. We consider a situation when a user accesses a web site for the first time and enters his/her username and password. Alternatively, this could also occur when a third party such as a device management server sends in credentials for a corporate user.

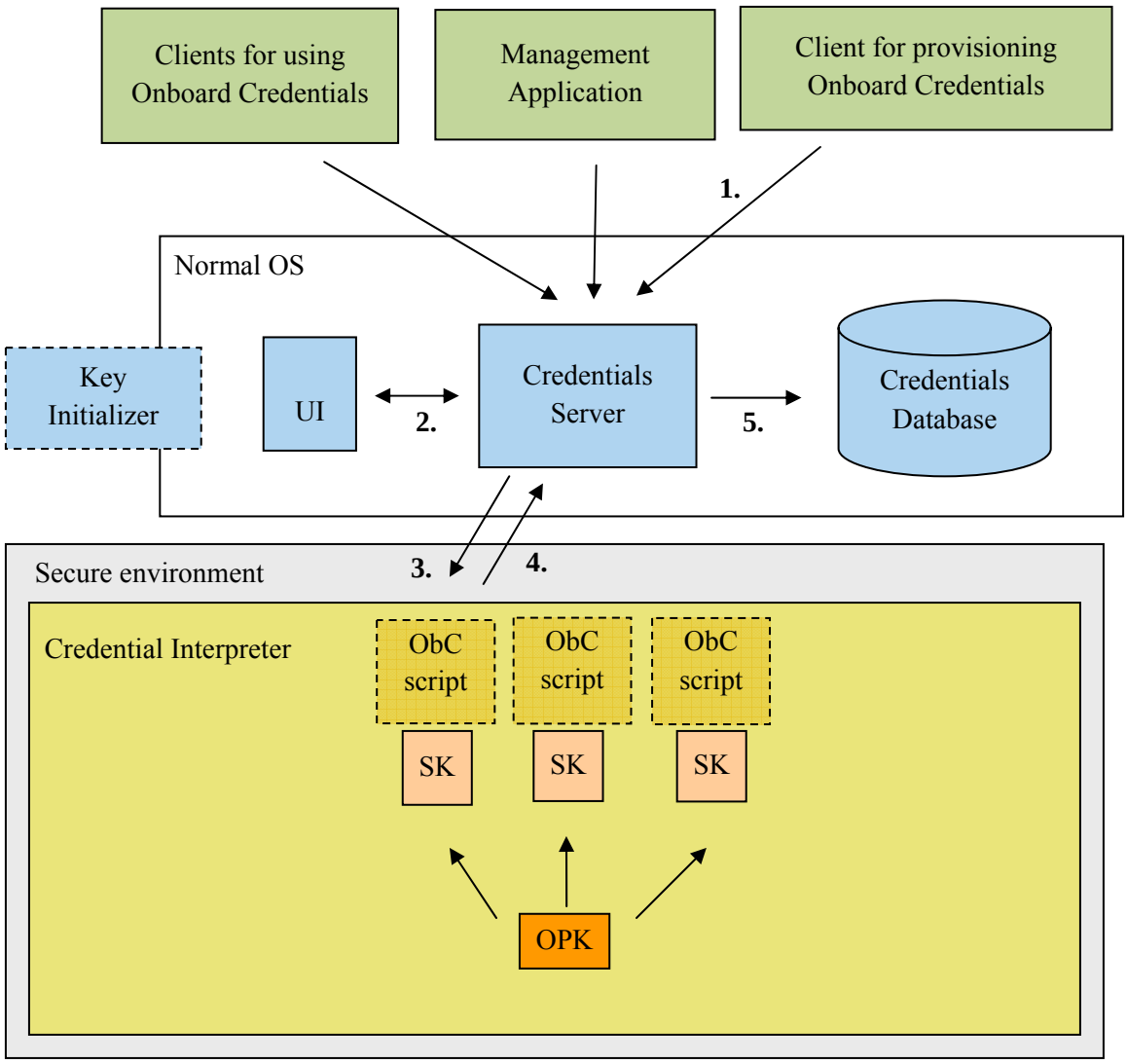

Figure 9: Onboard Credentials Architecture - Provisioning Use case

Here is a step-by-step use case:-

1. The credentials in the form of scripts are received by the system from a provisioning client or by the user himself if he created the credentials.
2. This information is presented to the user that a credential is being added to the system and the user is asked if this is acceptable. To restrict the usage of the credential, either a supplied policy may specify, or the user himself may choose to use additional local access control mechanisms such as PIN (see Section 4.3.3 about Local Access Control).
3. This credential along with the user-supplied information is given to the interpreter that resides inside the secure environment.

4. The logic portion of the credential, i.e. the script, encrypts the secret part of the credential using the script specific key (derived from the OPK) and returns back the script and the encrypted secret.
5. The encrypted secret along with the logic is then stored in a database that resides in the general unprotected environment.

Using credentials

The following paragraphs describe how the stored credentials are used by an application. Figure 10 illustrates an example in which the client wishes to authenticate to a server using a previously stored credential. The client must compute the correct response in order to authenticate itself. The server could be, for example, a web-server or a corporate VPN server.

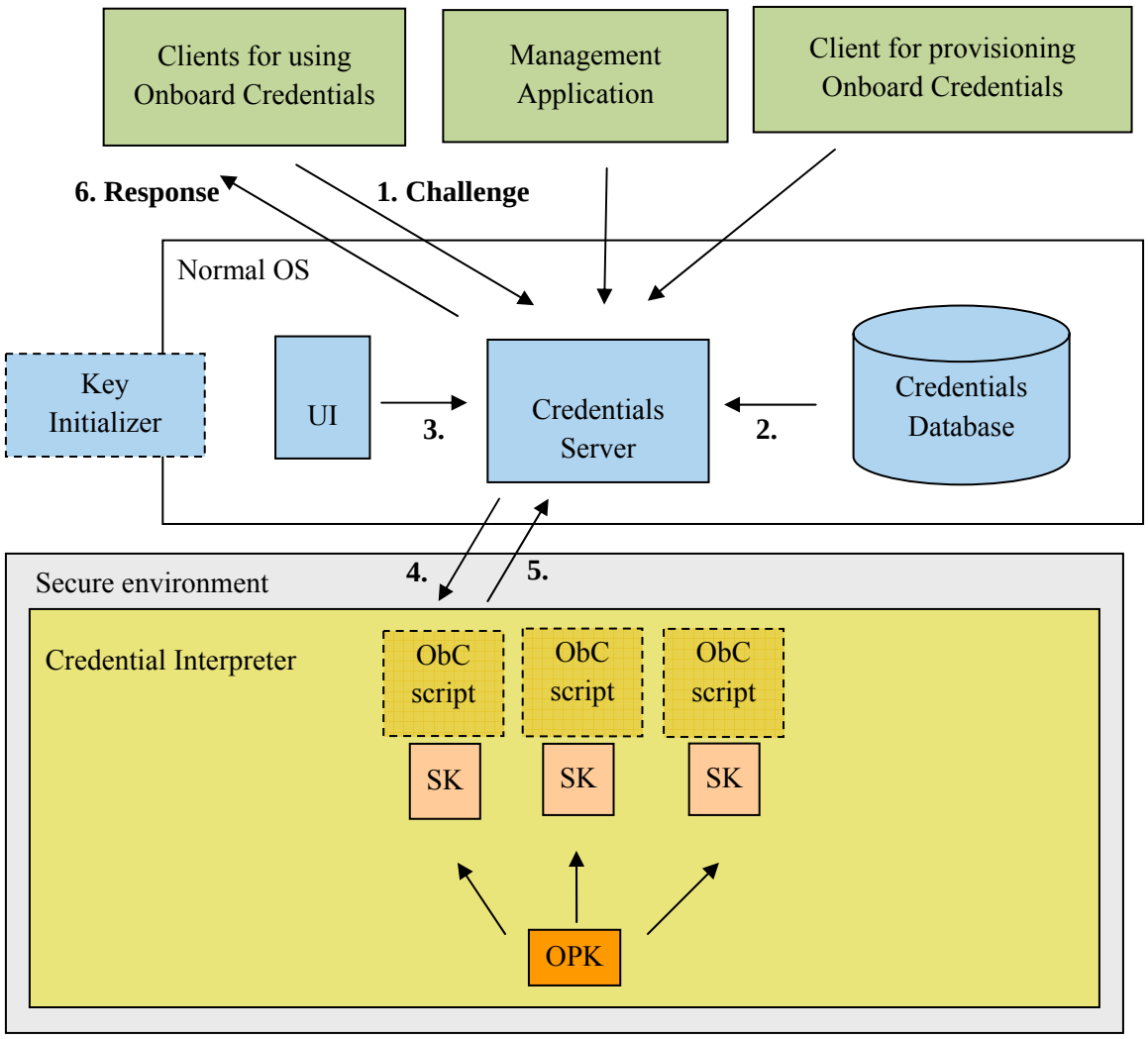

Figure 10 Onboard Credentials Architecture - Credentials Usage

1. The application requesting for a service is presented with a challenge. It sends a challenge to the Credential Server along with an identifier that specifies which credential to use.
2. The previously stored encrypted credential is retrieved from the database.
3. The Credential Server checks if any access control is required and asks for the user input such as PIN, etc.
4. The challenge, encrypted credential and any optional access control information is given to the interpreter that resides in the secure environment. The interpreter first authenticates the user if the optional access control data was present.
5. The credential interpreter internally decrypts the secret and executes the script with the challenge and the secret as an input to the script program. The script computes the response and returns this response back to the Credential Server.
6. The Credential Server returns this response back to the application. The application sends this response to the service provider for successful authentication.

4.2.2 Credentials Interpreter

As mentioned earlier, in the Onboard Credentials architecture, a credential comprises of a secret and an algorithm in the form of a script that operates on the secret. The script is pre-compiled into byte-codes. The Credentials Interpreter is a virtual machine that translates these byte-codes and executes them. Since the script is interpreted, there is no need for compiling the script, thus avoiding the dependence on a platform dependant compiler and other tools for development of credentials.

Credential Interpreter may reside in a secure execution environment. Applications data running in the secure environment are trusted and rarely isolated with each other. Therefore, it is important that the trusted applications be shielded from the scripts from untrusted third parties. Since the control of the scripts execution is entirely with the virtual machine of the Credential Interpreter, any harmful behavior of the script (intentional or due to a bug) is contained inside the interpreter itself. Figure 11 shows how the interpreter keeps the execution of the scripts contained.

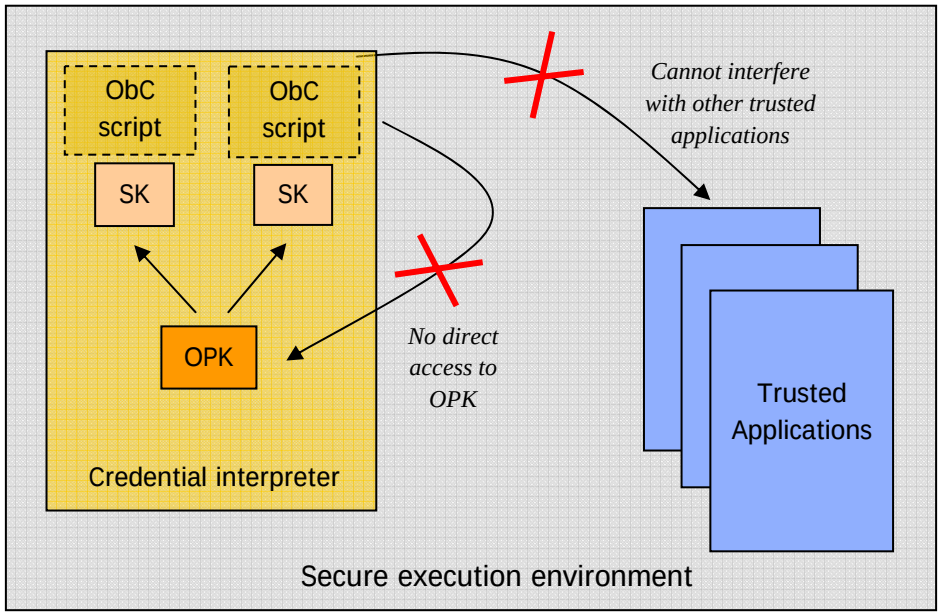

Figure 11 Credential Interpreter inside secure execution environment

The Credentials Interpreter internally uses a different encryption key for each script as illustrated in Figure 11. These script specific keys (SK) are derived from the main storage key by computing the keyed hash of the scripts.

$$SK = f\ (script, storage\ key)$$

This storage key may be available originally on the device itself, then known as Original Platform Key (OPK) or through provisioning – then known as Provisioned Platform Key (PPK) (discussed in more detail in Section 4.3). At least one Platform Key – the OPK must be available all the times. Using script specific keys also prevents direct access to the storage key (OPK or PPK) by the scripts as shown in Figure 11.

The Credentials Interpreter provides an interface through which the storage key (OPK) can be inserted into the interpreter module. To prevent possible replay attacks the Credentials Interpreter ignores further attempts at inserting the OPK. How the OPK is fetched and given to the Credentials Interpreter is platform specific and is discussed in more details in later sections (Section 5.2.3 and Section 5.3.3).

Controlling access to the Interpreter

The Credentials Interpreter in the proposed architecture is always trusted and is designed to be operated by only one application. Some operating systems such as Symbian [27], allow process authentication, which is used for access control to the Credential Interpreter. On platforms, such as Linux, that do not provide such process authentication, a shared-key based authentication mechanism is used. The Credential Interpreter then trusts the first application that attempts to communicate with it and establishes the shared key. The application must know of this protocol before attempting to talk to the Credentials Interpreter and therefore must generate a key and give it to the Credentials Interpreter. Code signing mechanisms using public-key encryption were not used, due to the problem of maintaining and distributing public keys.

The first application that talks to the Credentials Interpreter creates a 16-byte random key called the COMKey and gives it to the Credentials Interpreter. To prevent any replay attacks, the Credentials Interpreter does not accept any more keys once it has already received a key. In further interaction with the Credentials Interpreter, as a means to authenticate itself as the application that first initiated contact, the Credential Server computes a keyed hash of the total payload being sent using the COMKey. The Credentials Interpreter then re-computes the hash of the payload using the COMKey that was first given to it and compares it to ensure the authenticity of the message request.

The Credentials Interpreter however does not compute a hash of the output that it sends back because our basic presumption, as stated earlier, is that the Credentials Interpreter is trusted. This trust could be corroborated by sealing of the kernel by the Trusted Boot mechanism (as explained in Section 2.1.1) or by the security provided by the platform (assisted either by secure hardware [7] [7] or by built-in platform security as provided by some operating systems [27].

4.2.3 Credential Server

The Credential Server acts like the main engine and interacts with the other modules of the Onboard Credentials architecture. The Credential Server interfaces with the applications and accepts requests for adding new credentials, usage of credentials, etc. The Credential Server is constructed on the server-client paradigm. The server accepts requests and sends back the result to the client. The standard Inter Process Communication (IPC) architecture offered by the platform is utilized.

The Credential Server installs itself as a process and it is important that no other application or process pose itself as Credential Server. The security of the system relies greatly on the Credential Server and the Credential Interpreter. Therefore, it is important that neither of them is replaced by a malicious application. This can be also be ensured by platform through the security offered either by the operating system itself or by other means such as trusted boot. The communication between the Credentials Interpreter and the Credential Server is authenticated using the means provided by the operating system or by using a separate COMKey (as discussed earlier in Section 4.2.2).

The Credential Server manages the credentials and the corresponding access control and access policies. The Credential Server offers three kind of services through the API it exposes. The three types are:-

- **Provisioning Credentials:** The Credential Server provides a set of functions for inserting new credentials to the credentials Database. For example, the OMA Device Management application receives the credentials and requests the Credential Server to store the credentials. The server encrypts the secret part of the credential and stores along with the credential logic supplied by the application.
- **Using Credential:** The Credential Server also provides a set of methods to use the credentials that were previously stored in the Credentials Database. For example, a web browser may need to authenticate to a HTTP server using HTTP Digest authentication. The web browser asks the Credential Server to compute the HTTP digest based on the challenge. The Onboard Credentials system computes the digest from the previously stored credential and the corresponding logic.
- **Management methods:** The Credential Server provides various methods to perform management operations like defining and manipulating local accounts with different local access control settings. Local access control is discussed in more detail in Section 4.3.3.

4.2.4 Database storage

The credentials along with additional information such as local usernames, passwords, access control methods, policies, etc are stored in a relational database. The database consists of three tables:-

Credentials
Username
Realm
SealedSecret
LocalUserName
SealedLocalAuthLevel

LocalUsers
Username
CustomText
CustomPicture

LocalAccessControl
LocalUsername
Method
SealedVerificationData

The protected fields have been italicized. The credential secret and the verification data for local access control (e.g. PIN, biometric information) is sealed using the interpreter. Additionally information about the authentication level required for the credential must also be sealed so that a malicious user is not able to change the authentication level required for local access of the stored credential.

A relational database is used and the database file is stored on persistence storage. Open access to the database file is not considered a threat as the credential secrets are stored in encrypted form. Any reasonable encryption provided by the database engine for the meta-data is considered adequate.

4.3 Protecting the Credentials

This section describes the methods used to encrypt and provide secure storage of the credentials.

4.3.1 Storage Key Initialization

The OPK is the primary storage key and must be protected by secure environment. The Onboard Credentials system may be implemented on platforms offering different secure environments as discussed in Section 4.1.1. If the platform permits, OPK may be created during manufacture and stored in protected hardware. If the OPK is not provided by the platform, it must be initialized. The following sections discuss the situation where the OPK does not exist on the platform and thus, must be created.

On each boot, the Key Initializer checks if the Original Platform Key (OPK) exists. On a first boot, this check will fail. The Key Initializer then generates a random key, gives it to the Credential Interpreter and stores the OPK at a secure location (see secure storage in Section 4.1.1). Creation of the OPK ensures that the Credential Interpreter always works with a key that meets some minimum criteria (in this case, the criterion is that the key consists of enough random bytes).

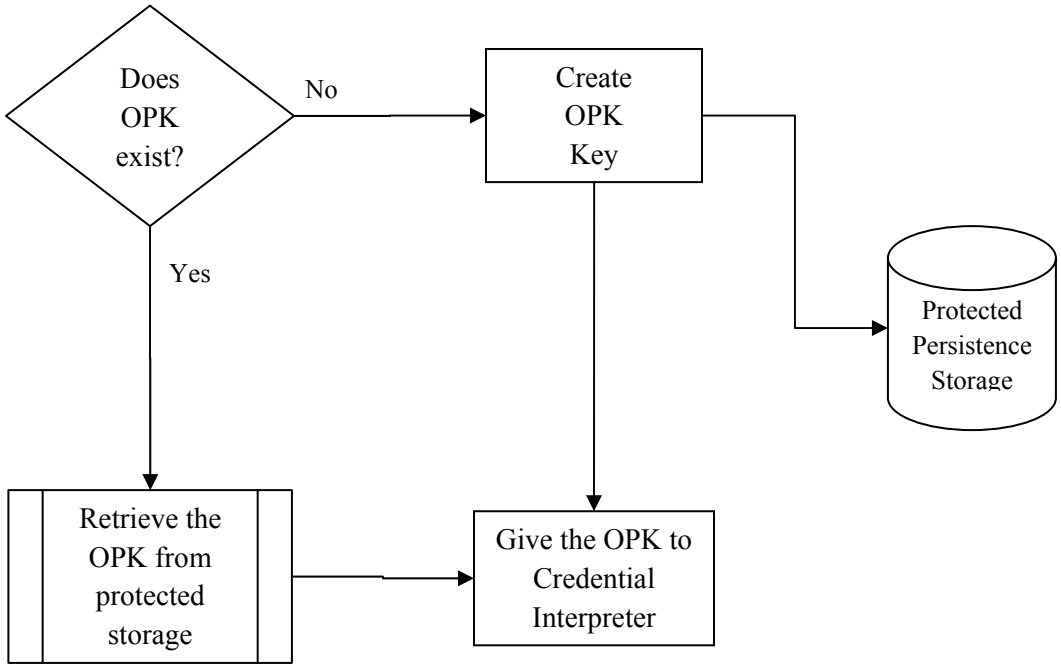

Figure 12: Initialization of the Original Platform Key (OPK)

On subsequent boots, the Key Initializer checks if the previously stored key exists. The keys existence tells the Key Initializer that a key has been generated before. On a successful retrieval of a key (which could include a possible platform specific decryption as discussed in Sections 5.2.3 and 5.3.3), it is given to the Credential Interpreter. As a general practice, the status about the crypto-function is not presented as it poses as a potential security risk. However, in our architecture, the Credential Server is trusted and is notified of any error that may occur during the initialization process. The decision to notify the user is left to the implementation of the Credential Server.

4.3.2 Provisioning

Reasons for Provisioning

The backup of the credentials consists of the credentials secrets encrypted with keys (SK) derived from the OPK (Section 4.2.2). This implies that the OPK is required to retrieve the credentials during a restore from the backup. Storing the OPK along with the backup is a security risk and should be avoided. It may also not be possible to transfer OPK from one device to another. Further, device manufacturers or corporate IT department may express their desire to keep a control of the storage key and may not like the key to be in possession of the user. Therefore, a

mechanism needs to be devised to allow the key for encryption to be provided by a trusted third party.

Provisioning may be useful for the following purposes:-

1. **Recovery:** As discussed above, in the event of a device reset or crash, it will be possible to get all the stored credentials from the stored backup as the key protecting the credentials would be supplied once again from a central trusted entity. In case the device is changed, provisioning shall provide the means to transfer the stored credentials to the new device.

2. **To communicate to individual scripts:** it provides a channel for the provisioning server to talk to the individual scripts (as script specific keys are derived from the provisioned key available both in the Lua Interpreter and at the provisioning server). Any other encrypted data may also be passed along to the script.

Protecting the provisioned key

The Provisioned Platform Key (PPK) is sent from a trusted server, such as a Device Management server. How this trust is brought about is beyond the scope of the thesis. The provisioning client residing on the device gives this PPK to the Credential Server. The Credential Server gives this PPK key to the Credential Interpreter. The Credential Interpreter on receiving the PPK derives two keys from the current OPK. One key is generated for authentication (AK) and another for encryption (CK).

The KDF is a generic key derivation function that is used to derive the keys for encryption and signing of the OPK. KDF used in our implementation was HMAC-SHA1.

$$AK = KDF\ (OPK, \text{“Authentication Key”})$$

$$CK = KDF\ (OPK, \text{“Encryption Key”})$$

The Credentials interpreter then computes MAC of the PPK using AK to ensure the integrity. Then it encrypts the PPK along with the computed MAC using CK.

$$PPK_{MAC} = MAC_{AK}\ (PPK)$$

$$PPK_{SEALED} = ENC_{CK}\ (PPK \,||\, PPK_{MAC})$$

The MAC function used in our implementation was HMAC-SHA1. The ENC algorithm used was AES-CBC.

The Credential Interpreter starts to use the PPK to derive the script specific keys (SK). The OPK continues to be the primary storage key. At the same time, the encrypted PPK (referred from now

on as PPK_{SEALED}) is returned to the Credential Server. The Credential Server receives PPK_{SEALED} and stores it in predefined location.

This sealing is implemented as a set of built-in commands and not as a script as is the usual way of sealing or unsealing using the Credential Interpreter. Having a script do the sealing makes it extensible, but there is the problem of storing the script itself. It was decided to use built-in commands, as it is more feasible and easier than to accommodate and design for secure storage of the sealing script.

The behavior remains mostly the same on subsequent boots. The Key Initializer program proceeds through its normal operation and gives the OPK to the Credential Interpreter. When the Credential Server starts up, it checks if PPK_{SEALED} is stored on the persistence storage; if found, it gives the PPK_{SEALED} to the Credential Interpreter.

The operation inside the Credentials Interpreter behaves in the reverse method as the provisioning process discussed in previous paragraphs. It generates the AK and CK keys from OPK and decrypts PPK_{SEALED} to obtain PPK and PPK_{MAC}. It verifies the authenticity by comparing the MAC with the one computed from HMAC of the PPK. On success, the Credential Interpreter starts to use PPK for deriving the script specific keys.

4.3.3 Local Access Control

Need for local user accounts

The credentials architecture could be implemented on a wide range of computing devices, some of which may be portable. Portable devices are small and can be easily lost or stolen, leading them into the hands of an adversary. Again, without any local access control, the adversary will be able to use the credential, even if he cannot extract the secret part of the credential. Therefore, there is a need to provide some kind of a local access control, based on authentication, for using the stored credentials.

Credentials may be of different levels of sensitivity. The user may require higher level of authentication of sensitive credentials for services such as online banking, VPN access, etc. At the same time, the user may not wish to have any access control for other less sensitive credentials, such as credentials for an instant messaging program. Multiple credentials for the same service may also exist. This could for be for example on a shared family computer accessed by everyone in the family. A single user may also choose to have multiple identities (and hence multiple credentials) on a chat server.

For credentials management and access control, local users are created under which the credentials are grouped and stored. Local access control checks may be performed using PIN or fingerprint or through other mechanisms. Each credential belongs to only one local user and access control to the credential may be optional, unless mandated by a policy accompanying the

credential. If multiple credentials are found, the user is asked to choose the credential by choosing the local user account.

Trusted Path

In the above paragraphs, we described the need for authentication for local user accounts. In Section 2.4, we discuss why there is a need to set up a trusted path while performing local access checks. A secure channel between the Credential Server and the user must be established, lest a malicious application steals the local access control information.

In Section 2.4, some schemes for trusted path are presented. Another approach is taken in the Onboard Credentials architecture. Instead of disabling a trap on a key sequence, a trusted UI in form of a predefined user interface is used as currently none of the operating systems (Linux or Symbian) offer a trusted UI concept. At the time of adding a local user in the Onboard Credentials architecture, a user is presented with a randomly selected image and asked to input some customized text. Each time the user is asked to input any information, the image and the custom text is presented. The user can be convinced that only the correct Onboard Credentials application can know the correct picture and custom text as an application spoofing the Onboard Credentials system will not be able to present this information. Our approach is similar to one described in [56] which talks about windows personalization a method to provide trusted path.

5 Implementation

In this section, we discuss how the Onboard Credentials architecture was implemented on two platforms – Linux with TCG/TPM and on Symbian OS using Platform Security. The same Credentials Interpreter implementation was used across the two platforms and is presented as a generic interpreter module. Platform dependant implementation details for the Credentials Interpreter are listed and discussed separately under the respective sub-sections.

5.1 Lua Interpreter

This section describes the implementation of the Credentials Interpreter as described in Section 4.2.2. Lua language [57] was selected to express the credentials as a script. The developer of the credential writes the script and gives to the Lua compiler to produce intermediate byte-code, which can then be executed by the Lua Interpreter. The version of Lua supported is limited to 2.4 (The latest version of Lua at the time of writing this thesis was 5.1.1, which was released on 09 January 2006). This was purposefully done to keep the Lua Interpreter simple and light-weight. It was further scaled down to support only 16-bit integer data type.

5.1.1 Working of the Lua Interpreter

The Lua Interpreter internally runs as a virtual machine with a multiplexer as a single entry point to the Lua core – which is the place where the script is actually executed. This single entry point into the virtual machine is used by both external and internal calls. Internal calls are differentiated with the presence of an "auth" field, which the virtual machine uses to authenticate itself to the Lua core. Alternatively, the same "auth" field could be used in the future to authenticate an external caller that may be utilizing Lua Interpreter's services.

The multiplexer invokes native functions as supported by the Lua core or external functions. External functions (as the name suggests) are outside of the core Lua functions and may be invoked from the script (as an external call) and from within the Lua core. These external functions are expressed as preprocessed macro identifiers and therefore, new external functions may be added without needing to change the Lua Interpreter internally. Each such external function thus corresponds to a Lua 'primitive' that forms part of the Lua interface API. Some examples of external functions are – ***`seal, unseal, add_secret`***, etc.

Any data given the Lua Interpreter is enveloped in Type-Length Value (TLV) format as illustrated in Figure 13. This format is beneficial as TLV sequences can be easily searched using a general parsing method and unknown TLV types can be easily ignored.

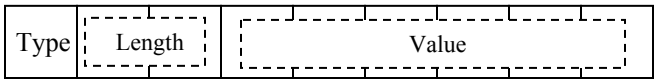

Figure 13: TLV Format

The data is enveloped as a Data TLV type and is put along with the executable script code and given to the Lua Interpreter, as shown in Figure 14.

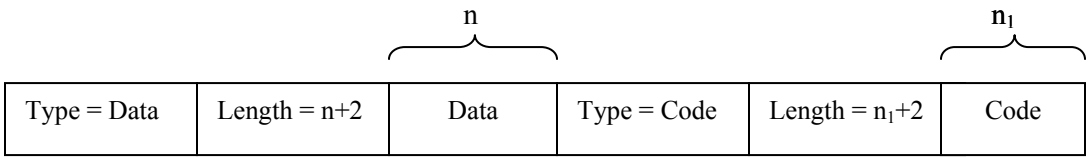

Figure 14: Data layout in TLV format as sent to Lua Interpreter

The application provides the memory to the Interpreter for execution and for storage of return values. This considerably reduces the memory requirements of the Lua Interpreter as an independent module. The layout of the buffer memory as given to the Lua Interpreter is shown in Figure 15.

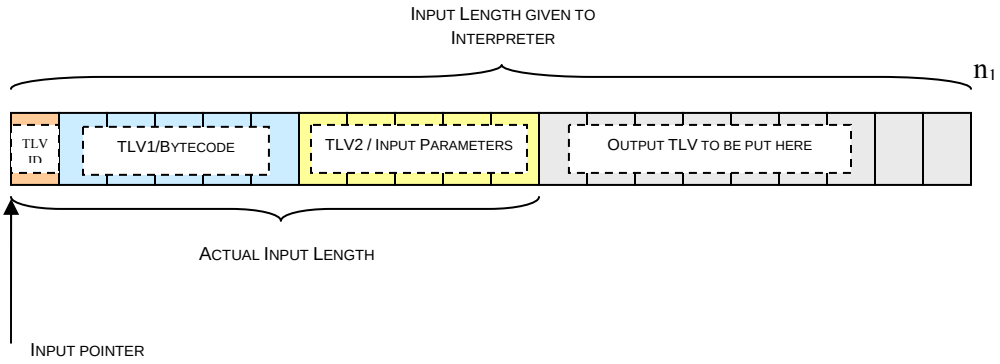

Figure 15: Memory buffer layout for the Lua Interpreter

5.2 Linux Implementation

5.2.1 Linux Architecture

Linux was chosen as one of the implementation platforms for Onboard Credentials as it is free and supports an open-source development environment.

TCG/TPM as means for secure storage along with the per-process memory isolation provided by Linux operating system constituted as the secure environment. IBM ThinkPad T42 notebook was used as the computing device with an Atmel TPM chip embedded in the motherboard. The open-source TCG Software Stack implementation – TrouSerS [58], created and released by IBM was used as software to access the TPM. The device drivers were compiled into the kernel, as it is easier to protect and verify the integrity of a monolithic kernel than protecting multiple kernel modules. TPM's Trusted Boot feature (Section 2.1.1- Secure and Authenticated boot) was used to verify the integrity of the system.

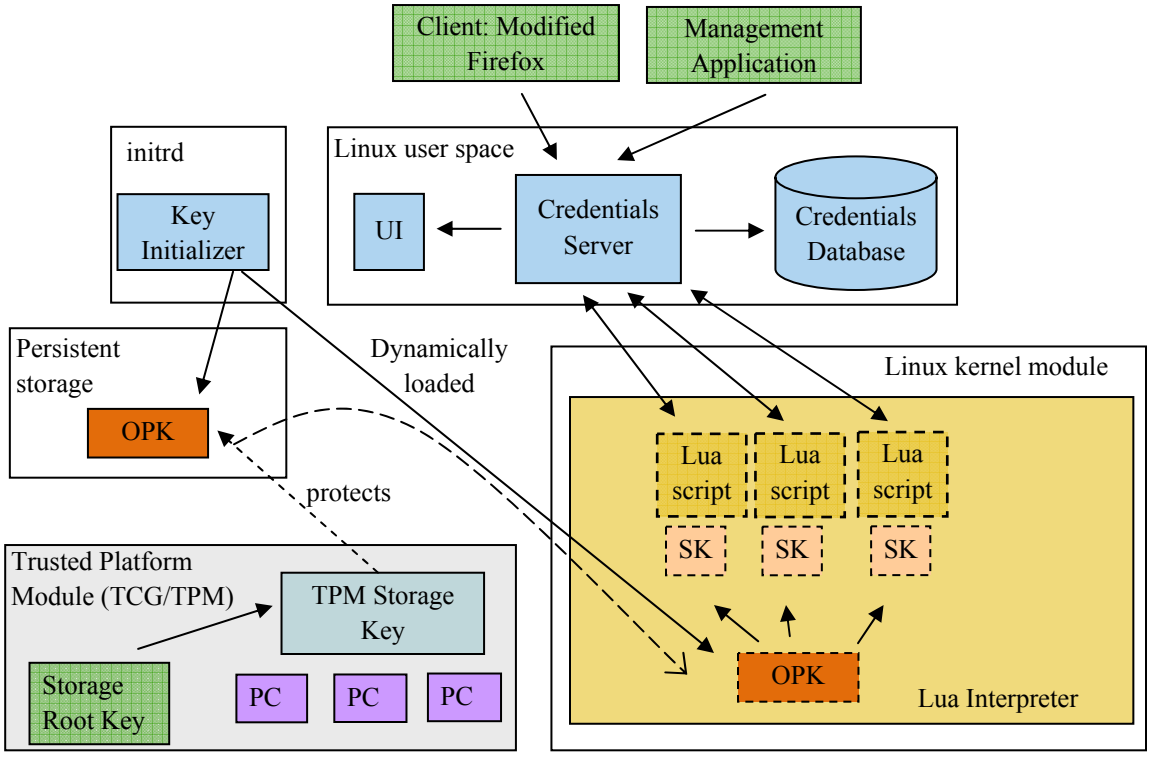

Figure 16: The Basic Linux Architecture

The Linux implementation deviated from the generic design, as the environment for the Linux platform was considerably different as shown in Figure 16. Firstly, the Credential Interpreter does not reside in the secure environment, as the TCG/TPM used in the Linux platform does not support a secure execution environment. Secondly, the Original Platform Key (OPK) is stored in the user-space and not in the secure environment. This led to additional steps being taken to secure the OPK. In the following sections, the deviations are described.

5.2.2 Lua Interpreter in Linux

The Lua Interpreter was implemented as an independent kernel module and is accessed as a character device. The Lua interpreter was put inside the kernel to ensure that no user-space process can interfere with it. Having a device driver as a kernel module is also a standard way of

providing kernel services to the user-space programs. Keeping the Lua Interpreter separate into the kernel module also provided good modularization and possibility of independent updates. The Lua Interpreter kernel module is loaded in the initrd [59] phase to ensure no user-space process can interfere with the loading of the kernel module. The Credential Server communicates with the Lua Interpreter by reading or writing to a character device - `/dev/lua`.

Access Control to the Interpreter

As discussed earlier in Section 4.2.2, to prevent any replay attacks Lua Interpreter does not accept COMKey more than once. Even if a malware is able to kill the Credential Server and attempt to masquerade as the Credential Server, it will not be able to communicate with the Lua Interpreter as it would need to know the COMKey. The only way Lua Interpreter can work again is after a reboot of the system. If the malware attempts to start its own Credential Server before the genuine one, it would need to modify the init scripts. The secure boot process ensures the integrity of the init scripts and would not boot the system if it finds that the init scripts have been changed. This makes sure that the genuine Credential Server would always be the first one to talk to the Lua Interpreter.

The COMKey support was required in Linux as the Linux platform does not natively provide any process authentication. Although identification through process ID (`pid`) was considered an option, it was soon rejected for the following reasons:-

a) The problem of Credential Server reliably communicating its PID to the Lua Interpreter.

b) Even if this could be done reliably, PIDs do not provide strong identification. A malware could kill the existing Credential Server and could continue to fork new processes until the system runs out of PIDs and rolls over. The evil application could install a new process with the old PID and pose itself as the Credential Server.

5.2.3 Protecting the Credentials

As discussed earlier in Section 4.2.2 the Lua Interpreter derives script specific key from the main storage key - OPK. This Original Platform Key (OPK) is given to the Lua Interpreter on system boot.

Storage Key Initialization

The Key Initializer in Linux was implemented as a small script – ‘keyinit’ that is put inside the Init Ram Disk (initrd) [59] image and is included in the measurements for trusted boot.

On every boot, this program checks if OPK_{SEALED} exists in the boot partition stored as `/boot/opk.sealed`. On a first boot, this check will fail. The keyinit program generates the 16 byte Original Platform Key (OPK) by reading from `/dev/random` and seals it using the TPM. The initrd (including the keyinit program) is used in the PCR measurements, which ensures the integrity of the keyinit program and the sealed key on subsequent boots. The sealing is required, as Linux does not provide any application specific access control that ensures exclusive read/write access by the Lua Interpreter. Sealing also ensures that even the root cannot extract the key. The security is in the fact that so early in the boot sequence no other application can unseal the key and extract it. If a malicious application wants to steal the key, it will have to modify the initrd to install itself. A modified initrd would yield different PCR values, thus failing the unseal operation.

The key is given to the Lua Interpreter using the primitive ***`add_secret`*** and the sealed key is written on to the file system in the path `/boot/opk.sealed`. The following are included in the measurements during sealing:-

1. BIOS
2. Trusted Grub bootloader and its configuration file (`/etc/grub.conf`).
3. Init ramdisk containing Lua Interpreter and TrouSerS libraries.
4. Linux kernel
5. `/etc/inittab` , `init` binary and all the libraries used by it.
6. The system loader – `ld.so` and its configuration file (`/etc/ld.so.conf`)
7. Credential Server binary and all the libraries used by it.

On every subsequent boot, the keyinit program checks if the `/boot/opk.sealed` file exists to know if a key has been generated before. If the file exists, it unseals the key file using TCG/TPM. On a successful operation, it gives the unsealed key to the Lua Interpreter using the ***`add_secret`*** primitive.

Protecting the provisioned key

The provisioning client on the computer gives the PPK to the Credential Server. The Credential Server gives this PPK key to the Lua Interpreter using the primitive ***`switch_to_ppk`***. The Lua Interpreter encrypts the PPK and computes a keyed-hash as previously discussed in Section 4.3.3.

The encrypted PPK (referred as PPK_{SEALED}) is returned to the Credential Server. The Credential Server receives PPK_{SEALED} and stores it in `/boot` directory with the name as `ppk.sealed`. At the same time, the Lua Interpreter switches the primary storage key to the PPK. The OPK still continues to be the master key and stays in the memory. The reason for not discarding OPK is that there could be another DM session before the next boot, although only one provisioning session is advised.

The behavior remains mostly the same on subsequent boots. The keyinit program proceeds through its normal operation and gives the unsealed OPK to the Lua Interpreter. When the Credential Server starts up, it looks for PPK_{SEALED} file at `/boot/ppk.sealed`. If found, it gives it to the Lua Interpreter using the ***`insert_ppk`*** primitive as described earlier in Section 4.3.2. Figure 17 shows the key hierarchy with provisioning support in Linux.

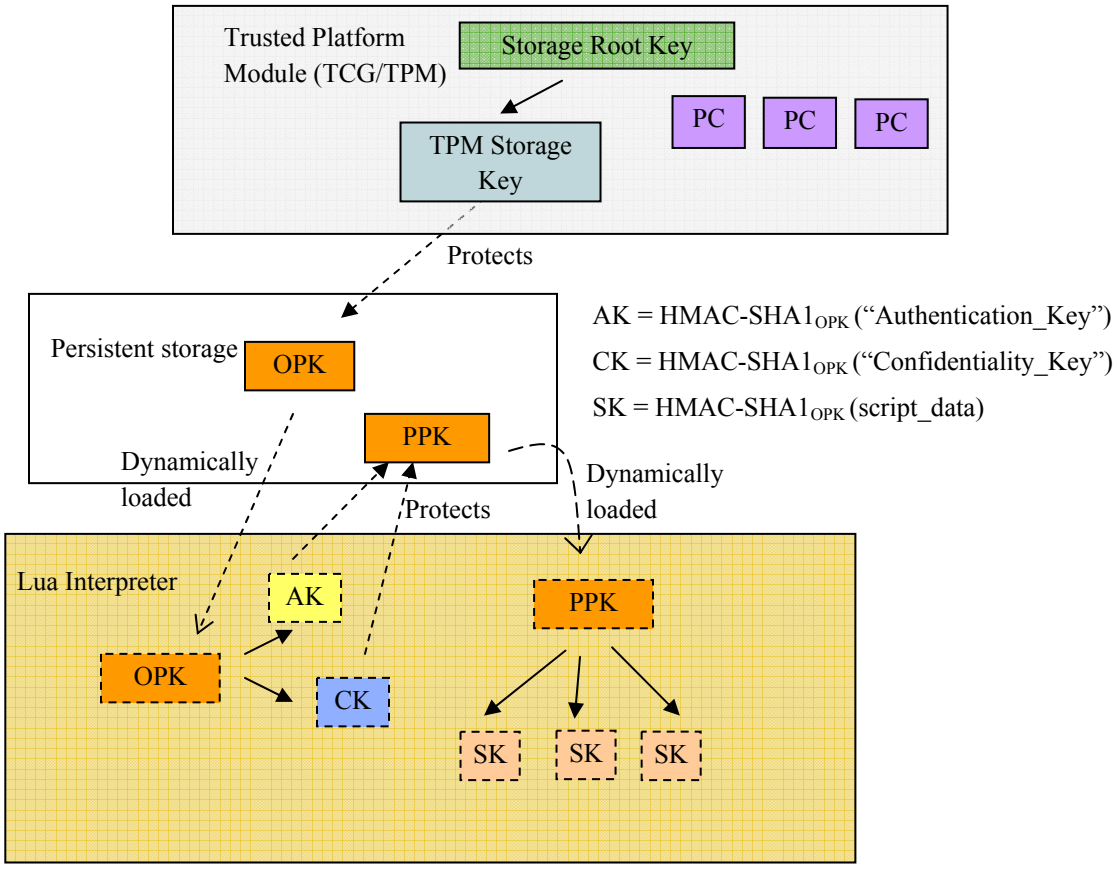

Figure 17: Platform Key hierarchy in Linux with Provisioning

Protecting from super-user access

The OPK and PPK reside in the Lua Interpreter's memory. Linux provides direct access to memory through the character devices `/dev/mem` and `/dev/kmem.` Although the access is restricted for all but the root, we wanted that the system is secured from a malicious root as well. It is possible for a hacker to get root access and read the memory using the two character devices to steal the storage keys.

Although access to raw I/O through the character devices can be disabled by removing the CAP_SYS_RAWIO capability in the kernel, some applications such as X server need access direct access to memory (via `/dev/mem`). LIDS [26] was used to implement application specific access control to this capability of the kernel. Using LIDS, The CAP_SYS_RAWIO capability was disabled globally and only X server binary was allowed access to it during the boot phase. In

principle, this means that the X server binary can still read the memory of the system and read the storage keys. Therefore, the X server binaries should also be included in the TPM measurements to ensure the integrity of the system.

5.2.4 Credential Server

The Credential Server is implemented as a standalone user-space process that is started in `/etc/inittab`. Sockets were used as means of Inter Process Communication (IPC) between the Server and the Credential Client (discussed in later section). The Credential Server uses an AF_UNIX socket instead of AF_INET to restrict the socket to the localhost. It was assumed that IPC between local sockets was relatively secure and only a kernel level process could read and interfere with the packets. Having the kernel included in the TPM measurements ruled out any attack possible by modifying the kernel code. LIDS further ensured that no arbitrary kernel module could be loaded. All the modules (except Key Initializer, Lua Interpreter and the graphics libraries) are linked together as one single binary that is also included in the TPM measurements for trusted boot.

As mentioned earlier, the Credential Server communicated with the Lua Interpreter through a character device (`/dev/lua`). Blocking read and write system calls as offered by Linux OS were used. This kept the overall complexity of the system relatively low. Performance was not a consideration as the scripts were small and the underling processor was capable of fast execution.

A trusted UI as described in Section 4.3.3 was created based on GTK2+ [60]. This increased the dependence on external X & GTK libraries. The security threat from X and GTK libraries was considered relatively low as these are open source projects and therefore the probability of a Trojan is negligible. If required, the X and GTK libraries could also be included in the platform measurements in the TPM to enhance the security.

SQLite v3 [61] was used as the database engine for storage of encrypted credentials and the other data. It was chosen as it is a portable, free ware, open-source implementation of an RDBMS, and did not add any bulk to the Credential Server binary. A free ware C++ wrapper was used on top of the SQLite library.

Applications access the Credential Server module using the Credential Client module. It resides as a separate object that an application must statically link with to utilize the services of the Credential Server. The Credential Client API exposes two types of API. Some illustrative examples of the API are:-

1. Credentials Usage API
    a. Encrypt/Decrypt Credential
    b. Calculate HTTP-Digest

2. Credentials Modification API
    a. Add Credential
    b. Delete Credential
    c. Modify Access Control
    d. Check if Credential Exists

The credentials stored in the database are indexed by server ID. The client uses the credentials by giving server ID as a parameter, such as in the `UseCredential()` function of the Credentials Client API. This prevents accidental use of an incorrect credential.

5.2.5 Modified application - Mozilla Firefox Browser

Mozilla Firefox 2.0 web browser was used as a client to demonstrate the usage of Onboard Credentials architecture. The browser uses the credentials stored in the Onboard Credentials database. First time a website is accessed requiring HTTP Digest authentication, the user inputs the username and password as is normally done (shown in Figure 18).

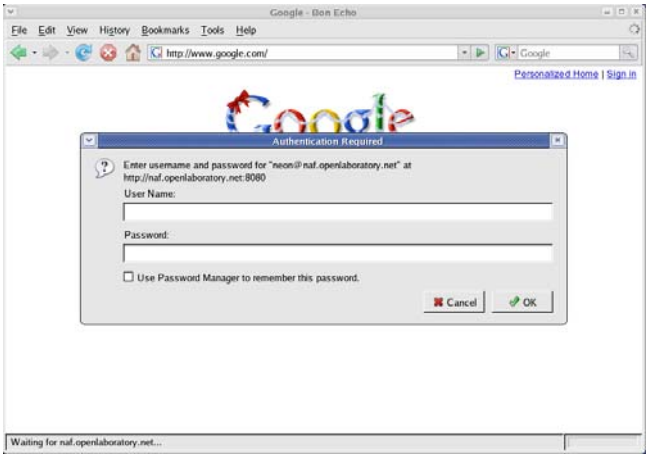

Figure 18: First time access to website. Username/password prompt

The modified browser stores a new credential that consists of the username and password. This is stored in the Onboard Credentials system as earlier described in Section 4.2.1. While adding, the user is asked to select the level of access control for this credential as shown in Figure 19.

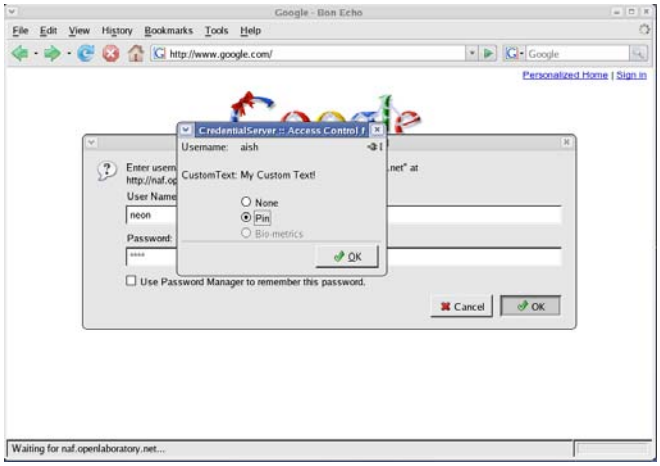

Figure 19: User asked to select access level

On subsequent access, browser invokes the Credential Server with the challenge as an input. Before giving the challenge and the stored credential to the Interpreter, the Credential Server asks the user to supply the PIN for local access. The user does not have to input the password again.

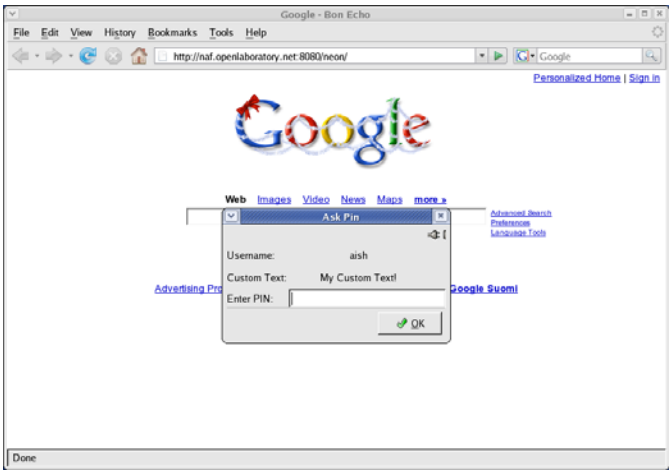

Figure 20: On subsequent visits, only PIN is asked while displaying the custom text.

The response is calculated in the Lua Interpreter and given back to the browser. This response is sent back to the server for authentication. Firefox's nsHttpChannel and nsHttpDigest Modules were modified for providing support for Credential Server.

Figure 21: Login is successful

5.3 Symbian Implementation

The Symbian version of the Credential Server was jointly implemented by Kari Kostiainen and Aarne Rantala at Nokia Research Centre, Helsinki. Support for SIP client and for provisioning was developed while working for this thesis jointly by the author and Aarne Rantala at Nokia Research Centre, Helsinki.

5.3.1 Symbian Architecture

The Symbian Platform Security [27] provided the secure environment in this implementation. For further details please refer to Section 2.1.2. The Credential Server was designed and implemented following the Symbian Server-Client architecture. The Server executable must be loaded in order for it to receive and process requests.

The Symbian architecture differed from the generic architecture in some aspects and is shown in Figure 22. The OPK is stored unencrypted in a private directory that is accessible only by the Credential Server process. This secure storage is provided by Symbian Platform Security. The relational database offered by the Symbian OS was used using the class `RDbStoreDatabase`. The database file is also stored in the private directory of the Credential Server process.

Most of the modules of the Onboard architecture were linked together to form one monolithic binary, including the Key Initializer, Credential Server and Lua Interpreter (discussed in later sub-sections).

As Symbian also does not offer any trusted UI concepts, the same mechanism using custom text and custom pictures were used as described in Section 4.3.3. The pictures were already stored in

the private directory of the Credential Server. Instead of storing the pictures, the corresponding IDs of the picture were stored in the Credential Server database. Native Symbian OS UI APIs were used to build and invoke the UI.

The Credential Client module is used by applications to access the Credential Server. The client-server authentication is done using capabilities such that only clients with the required capability may connect to the server (In our implementation, we used `Network Services` capability). A special server name is assigned (beginning with '!') to the Credential Server, which can be used only if the server also has this certain capability (`ProtServ` capability). A server using this capability requires that the server has been signed by a trusted party and thus the likelihood of malicious Credential Server starting before the correct one is small.

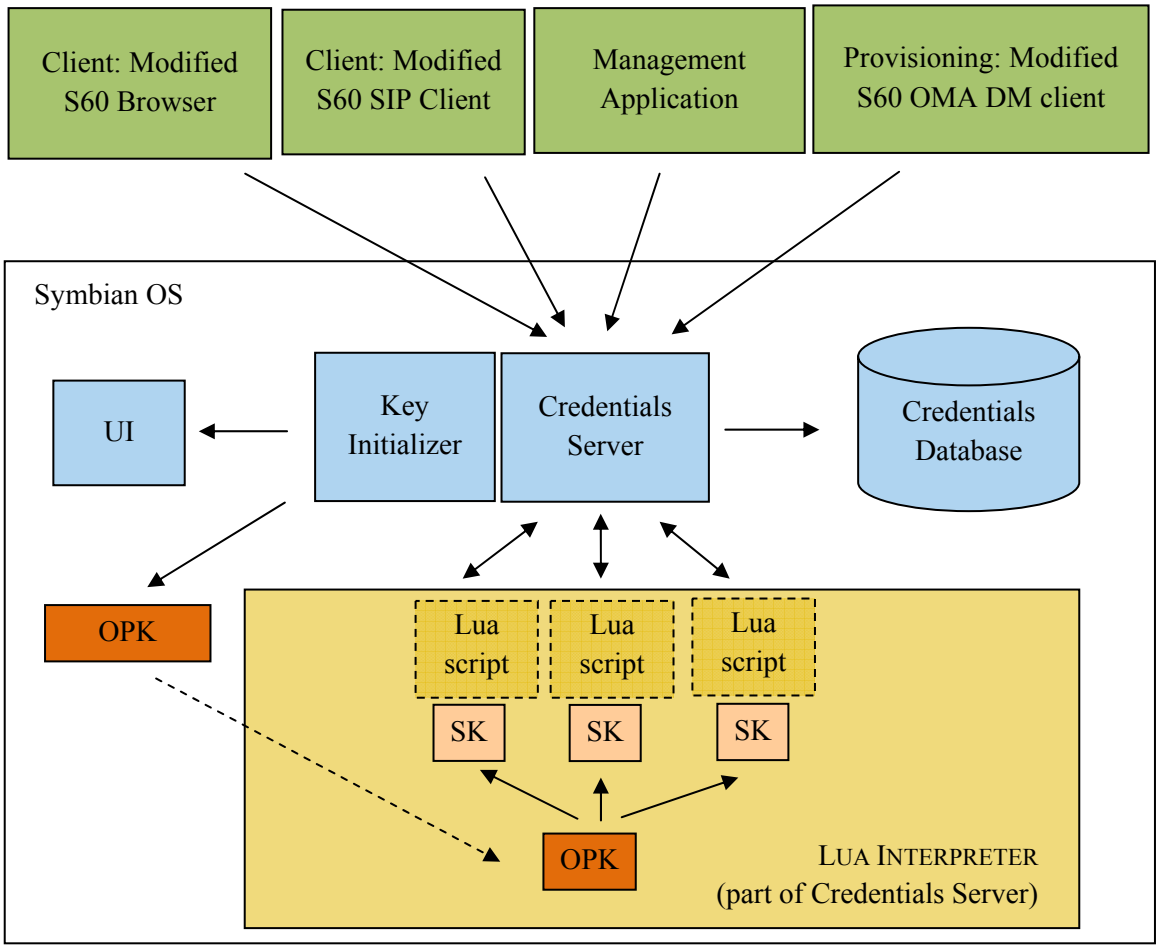

Figure 22: Symbian Architecture

5.3.2 Lua Interpreter in Symbian

A similar implementation of the Lua Interpreter was used with support for provisioning as described in Section 5.1. Lua Interpreter was linked as part of the Credential Server and hence establishing a COMKey for authentication was not required as previously prescribed in Section 4.2.2. This combining was done as it was most convenient and no gain was perceived in keeping Lua Interpreter separate from the rest of the binary.

5.3.3 Protecting the Credentials

In Symbian, the Credential Server is not started when the device boots. It is only started the first time the application attempts to connect to the Credential Server via the Credential Client using normal Symbian IPC server startup mechanism.

Symbian provides private directory to each application. Hence, protection by means of encryption is not necessary for storing the OPK and it (and well as PPK) can be kept in the clear in the private directory of the Credential Server. The confidentiality and integrity of this is maintained by Symbian Platform Security.

Key Intializer's functionality in Symbian is implemented in a separate module called "Key Manager" which is integrated with the Credential Server. Unlike Linux, it does not reside as an independent application. The approach of putting the keyinit functionality in the Credential Server was taken, for easier implementation.

Storage Key Initialization

When the Credential Server is loaded, the Key Manager module checks if there is a file – `opk.key` residing in the private directory of the Credential Server. On the first use, this check fails. The Key Manager module then creates a 16 byte random key using the library functions provided by Symbian. The generated key file is stored in the file `C:/private/<directory#>/opk.key` and is given to the Lua Interpreter by calling the ***`add_secret`*** primitive. The `directory#` is the Secure Identifier (SID) of the Credential Server.

On the first use after a subsequent boot, the Key Manager module checks if `opk.key` file exists in the Credential Server directory. The key file is read and given to the Lua Interpreter using the ***`add_secret`*** primitive.

The Credential Server is shutdown and unloaded from the memory if it has not been used for a preconfigured time period as per normal Symbian IPC convention. On every subsequent restarts, this cycle is repeated.

Protecting the provisioned key

Provisioning in Symbian follows a similar procedure as in Linux. The Provisioned Platform Key (PPK) is sent from a trusted server such as the Device Management (DM) server. The provisioning client on the mobile phone (DM client) gives this PPK to the Credential Server. Credential Server is authenticated with the DM client through platform security, example SID checks.

The Credential Server gives this PPK key to the Lua Interpreter using the primitive ***switch_to_ppk***. The provisioning remains the same as described in the generic architecture for Provisioning in Section 4.3.2.

When the Credential Server is used for the first time after a phone restart, the Key Manager initializes the Lua Interpreter with the OPK key. The Credential Server then checks for the `ppk.sealed` file stored in the private directory and gives it to the Lua Interpreter using the ***insert_ppk*** primitive. Further details are the same as in Section 4.3.2.

5.4 Contribution in this thesis

At the start of the thesis, the Onboard Board Credentials system had been designed already by Asokan, et al. The Lua interpreter was designed and implemented in ANSI C by Jan-Erik Ekberg. A proof of concept on Symbian platform was jointly implemented by Kari Kostiainen and Aarne Rantaala.

The author was responsible for implementing the Onboard Credentials system on Linux. Although the architecture of the software remained similar across Symbian and Linux, there were many implementation specific differences. The various modules of the Onboard Credentials system were rewritten in C++ for Linux. The implementation required installation of Trusted Grub [62] for trusted boot mechanisms. TrouSerS drivers required modification for support in initrd. The Lua interpreter was ported to Linux as a kernel module that functioned as a device driver to a character device created on the platform. Lua interpreter was modified for adding support for storage key initialization and provisioning in both Linux and Symbian platforms. Additionally, COMKey support was added to the Lua interpreter in Linux for authentication of Credential Server.

The Credential Server was configured to start in `/etc/inittab` and the initrd was sealed to the measurements as described in Section 5.2.4 LIDS was installed to further protect the integrity of the system from an adversary with root privileges.

To illustrate the usage of the Onboard Credentials system, Firefox browser was modified. Additionally, the SIP client on Symbian was also modified jointly by the author and Aarna Rantaala, to support Onboard Credentials for HTTP digest authentication.

6 Analysis

This thesis presents a novel system called Onboard Credentials for storing and using credentials in which the credentials are expressed as scripts and run in an interpreter situated in a secure environment offering increased security compared to software credentials by using secure hardware modules such as TCG/TPM [7] [7]. To provide a proof-of-concept, a prototype of the system was developed on two platforms, Linux with TCG/TPM and with Symbian OS with Platform Security. The preceding sections have discussed the design and subsequent implementations of the Onboard Credentials architecture. In this section, we analyze the Onboard Credentials system and evaluate it in comparison to other solutions proposed in this direction.

6.1 Security Evaluation

The Onboard Credentials system utilizes secure environment to store the credentials. The secure environment in the Linux implementation consists of the TPM hardware module and the secure execution environment provided by Linux operating system. The keys for encrypting the credential secrets are derived from storage key (OPK) which is sealed to the current platform configuration by using the TPM. The unseal operation in the TPM will fail as the platform measurements will change due to the presence of the malware. This provides as a protection against malware attempting to extract the key. TPM provides a mechanism to report the integrity of the system, which can be presented as a proof that no malware is present on the system. Symbian, we utilize the secure storage and per-process memory isolation provided by Symbian Platform Security as the secure environment.

We discussed that the system allows credentials originating from multiple sources to be stored in our system. The multiple sources consist of trusted sources and as well as untrusted third parties. The hash of the logic part of the credentials and the storage key is used to derive the key with which the credential's secret is encrypted. This makes sure that the script of one credential type cannot maliciously access and exract the secrets of another credential type. As a consequence of this, we also prevent direct access of the storage key by the scripts implying that scripts, especially originating from third parties, are never able to steal the storage key. Secure hardware such as ARM TrustZone [15] and Java Cards [17] natively support multiple credentials and provide isolation between them in the secure environment.

The Credentials Interpreter executes the script keeping a tight control over the scope of the credential's logic inside the secure environment. This ensures that the scripts can never interfere with other applications in the secure environment. Lua Interpreter that ensures that it does not take all of the system's resources. This makes sure that the interpreter does not harm other

applications in the secure environment such as consuming all of the CPU time and causing denial of service.

The design of the Lua Interpreter is such that it utilizes the memory space of the caller-application of the interpreter for all computation. This means that should the script of the credential try to trigger a buffer overflow or other similar memory buffer attacks, the memory of Lua Interpreter will not be affected. As no copy from internal memory to memory of the caller-application done, it rules out the chances of an accidental copy of internal memory of the Lua Interpreter to an external memory space. Such a feature protects accidental leakage of keys (e.g. OPK, PPK, COMKey) stored inside the Lua Interpreter.

Password managers, such as PwdHash, Password Multiplier and PassPet all aim at enhancing the security by protecting the passwords against dictionary attacks. In all three projects, the user remembers only a simple password. The passwords for individual services are derived from information from the website and the user password with the assumption that even if an attacker is able to retrieve the password for one website, passwords for other websites will not be compromised. Such a scheme still suffers from the vulnerability of guessing attacks as a human password is involved. In the Onboard Credentials architecture, such attacks are not successful as the storage keys are not human passwords.

To establish the trust of our system with the user, a trusted UI similar to one proposed in *PassPet* was used. This somewhat prevents phishing attacks by applications that may try to spoof the Credential Server to extract the local authentication information.

The Onboard Credentials system works with a characteristic that once a key is provisioned and stored in the system, it will be never is presented outside the secure environment in the clear. This characteristic of never presenting the key outside the secure environment also poses a problem when migrating credentials from one device to another. We address this challenge by using a Provisioned Platform Key (PPK) as discussed in Section 4.3.2. This allows the credentials to be migrated to other devices while still being protected with a key that is provisioned to the device preferably out of band. Thus, we devise a mechanism in which the credential secrets are protected even when they exist outside a device.

We have already discussed that we protect the credentials using secure hardware that makes extraction of credential secrets very difficult. To protect the unauthorized usage of the credentials, local access control checks were implemented as described in Section 4.3.3. To prevent any spoof attacks at attempts to phish the local access secrets by malware, we use a trusted UI as used in *PassPet* [49]. This establishes a trust between the user and the Credential Server. Trust between the Credential Server and Lua Interpreter is established by using COMKey where the operating system cannot provide process authentication.

Our system provides some level of protection against phishing if the web client is non-malicious. For example, in normal use, a user may visit a fake website. If the user does not check the realm, he may give the username and password to the fake website. However, in our system, the browser

gives the realm to the Credential Server. As the credentials are indexed with the Server ID, wrong credentials are not used.

The Onboard Credentials system does not require that any existing protocols be modified. This also ensures that no exhaustive testing is required for ensuring the security of the modified protocol. The password managers in [3], [4], [49] as such do not break any existing protocol but they do require extra efforts in recreating passwords for the services.

We summarize the security features of the Onboard Credentials system:-

- The system provides better security than software credentials by using secure hardware modules.
- The system ensures security between multiple credential types.
- The system ensures security between credentials from difference sources.
- The system prevents direct access to the storage key by the credentials.
- The system incorporates access control checks to the credentials.
- The system prevents some level of protection against phishing and spoofing attacks by using trusted UI concepts.
- The system has a property that once stored, it never gives the secret in unencrypted form to the application.

6.2 Usability Evaluation

In this section, we analyze the usability aspect of the Onboard Credentials system.

Existing works in this field, such as *PwdHash, Password Multiplier and PassPet* all aim at enhancing the security by protecting the passwords against dictionary attacks, while keeping the usability aspect in perspective. Usability is enhanced as the user is required to remember only one password. Security is not compromised as this password is used as a salt to derive passwords that are more complex. In the Onboard Credentials system, the credential secrets are encrypted with a cryptographic key. Therefore, the user does not need to remember any passwords.

Generally, secure environment can be utilized by only trusted applications. Third parties may want their information to be stored securely using secure environment. Certifying an application as trusted is a time consuming process that may involve many intermediaries. The script-interpreter mechanism combined with script specific keys allows even third parties to utilize the secure environment, without affecting the sanctity and security of the secure environment. The interpreter can also run in normal operating system so that extensive testing of the credential script can be performed before actual deployment.

The user is not required to input the credential secret for accessing services as it is supplied by the Credential Server, making it much more convenient to the user. To prevent frequent input of PIN

or fingerprint for local access control checks, the local authentication information is cached for a predefined time.

The credentials may be used on client machines that do not support Onboard Credentials, as the Onboard Credentials system does not modify the credentials. In such a case, the user must present the credentials to the concerned application directly. This is different from as *PwdHash, Password Multiplier and PassPet* as they require that the plug-in must be installed on the client machine. Only PwdHash provides an online service for retrieving the passwords if the plug-in cannot be installed. Further, the password managers assume that the user will remember the secrets.

We summarize the salient usability features of the Onboard Credentials system:

- The system facilitates even third parties to develop new credential types
- The user is not required to present the credentials frequently thus, making the login process convenient for the user.
- Local access control checks such as PIN, fingerprints are cached to avoid asking the user frequently.
- Credentials can also be used on systems that do not have Onboard Credentials system.
- The system allows copying credentials to another device or backup repository conveniently.
- The system provides a mechanism to possibly migrate the credentials while they are still encrypted.

6.3 Criteria Evaluation

In this section, the proposed system is analyzed against the criteria presented earlier in Section 3. In addition, the related work presented in Section 2.5 is compared and evaluated against.

1. The system should provide protection from malware.

Our architecture is robust and does not leak any credentials in spite of a compromised system as long as the secure environment remains safe. The secure environment protecting the storage keys does not allow any other object or process to access the stored keys. On platforms such as Linux with TPM, the storage keys are sealed to a platform configuration. Installation of malware changes the platform measurements thus disabling the decryption of these keys. TPM further prevents any attempts to bypass the normal boot sequence in order to extract the storage key. Symbian Platform Security prevents access to the storage keys by any other process other than the Credential Server process. Further, it only allows applications whose behavior has been

verified and thus signed to have access to the protected storage area of the operating system. Attempts at providing protection against malware by integrity checks have been previously discussed and incorporated in [46], [6], [45] and [32].

The system also prevents phishing attacks by malware attempting to spoof the application. The Onboard Credentials incorporates a trusted path between the user and the Credential Server. It presents the user with text and picture that is known only to the user and the Credential Server application, thus establishing a mutual trust between them. The PIN cache feature prevents the user from inputting the local PIN frequently. A similar concept is used in PassPet [49] where the user is presented with a Trusted UI.

Denial of Service (DoS) attack is possible on the platforms such as Linux. By changing the platform state, the OPK will fail to be decrypted. Other secure hardware modules such as ARM TrustZone (Section 2.1.2) may be used to provide secure storage of the OPK without this vulnerability.

2. The system should permit multiple credential types.

Secure environment is tightly controlled and only trusted applications are allowed to run inside it. Therefore, adding a new application in the secure environment is a time consuming process. Using the scripts mechanism to express logic part of the credentials facilitates easier creation of new credential types as new credential types do not imply addition of a new application in the secure environment. In our architecture, only the Credentials Interpreter that runs inside the secure environment needs to be approved. As the scripts execution is tightly controlled, untrusted third parties may be allowed to utilize the secure environment. Any bugs or harmful behavior of the script, intentional or otherwise is contained inside the interpreter itself. The script approach is also better than having separate applications for each credential type in the secure environment as it prevents extensive testing of each such application for a credential type before it can be included in the secure environment.

The credential types are protected against each other as the key for encrypting the secret part of the credential is derived from the logic part of the credential (which identifies the credential type). Thus, a key derived from one credential type is not applicable to another credential type. This protects the secrecy between different credential types.

The scripting environment facilitates development of tools and emulators with debugging facilities to be run as software on normal computers. Thus, it is possible to test the scripts prior to deployment to the secure environment. None of the existing software based credentials storage systems discussed in (Section 2.5.2) allow addition of new credential types. The only credentials types that the password managers [3] [4] [49] can store are only passwords. Hardware based modules such as Java Cards (Section 2.1.3) allow storing of different credential types for different applications.

3. The system should permit multiple credential instances.

We had described the need to store multiple credentials, which may be each of different credential types. The Onboard Credentials system supports multiple credentials of differing credential types. Thus, our system can support a multitude of applications and services. As mentioned earlier, different credential types are protected against each other using the script specific keys. Secure hardware such as ARM TrustZone [15] and Java Cards [17] provide secure storage and isolation between secrets.

4. The system should allow credentials from multiple sources such as users, service providers, corporate IT departments, etc.

Our architecture allows new instances of credentials to be created externally and provisioned to the devices. These credentials may be created by device manufactures, service providers and untrusted parties as well. The user is allowed to create credentials and utilize the Onboard Credentials system to securely store her credentials.

Other similar projects - *pwdHash, PassPet, Password Multiplier* do not have any such feature to allow third parties to create passwords and store them in its database. The user must create each password himself by browsing through each service's website.

5. The system should provide a mechanism of a backup of the credentials.

Our system allows the user to copy the encrypted credentials database and store it on another device. The storage key – Provisioned Platform Key (PPK) may be provisioned externally to the device from a trusted third party server. The PPK is protected by the OPK present already (or created) on the device. Although protection of PPK was not necessary on the Symbian device (as Symbian Platform Security guarantees that none other than the Credential Server can read the private directory where the PPK is stored), the implementation on Linux and Symbian was kept the same to keep the Lua Interpreter the same on both the platforms.

On a device change, there is an external infrastructure required to provision the PPK, but this is required as it would be difficult for the user to remember the long cryptographic key. A similar provisioning concept in case of a device change is described in *PassPet*. In *pwdHash,* if a plug-in does not exist on the computer, the user may go to the service provider's website to retrieve the passwords. In Password Manager, the credentials are not stored locally as they are calculated at runtime from site name and the master username, password previously entered. Transferring to a new computer requires installation of the software plug-in [63]. There is no web service available to retrieve passwords; this means that the services are unavailable on computers where the plug-in cannot be installed.

6. The system should be compatible with existing services such as servers, protocols and clients.

It is not feasible for services to modify their existing service architecture to accommodate a new scheme to store credentials. The Onboard Credentials architecture does not interfere with existing service infrastructures. It does not break any protocol and continues to function with existing services. Such a quality is also satisfied by the password managers discussed in [3], [4], [49].

Single Sign-On schemes such as Liberty Alliance [52] and Microsoft Passport [51] have this shortcoming, as they require additional network infrastructure as well as SSO aware clients. It additionally requires changes in the authentication protocol. Thus, deployment is expensive in Single Sign-On mechanism.

In our system, the client application needs to be modified to make it aware of the Onboard Credentials architecture. However our experience with the implementation in Linux and Symbian in Web browsers and SIP clients using HTTP digest have shown they very little change is required in the code (less than 50 lines of C++ code). One shortcoming of such a solution is that it is much more feasible to make a change at a central location as a server than change individual clients. SHEMP [32] requires modifications at client to provide attestation of the software running on the clients.

Server modifications are required in [49] to provide backup services. [31] and [32] also require modifications at server and protocols for using the secure credentials storage.

7. The system should allow services to be accessible from multiple devices

Users should be able to access services from more than one device; storing the credential on one device should not disable service access from another device unless mandated by the service provider. At the same time user should be able to access the service from a device that does not implement the proposed system.

Onboard Credentials system allows the credentials database to be copied onto multiple devices. To use the credentials, the Onboard Credentials system must be present on the other devices. The storage key protecting the credentials may be provisioned by a trusted third party as already discussed in Section 4.3.2. One limitation of our architecture is that we do not currently provide any method for remote attestation of the device before the trusted server sends the PPK. This is a risk as a PPK could be released to a compromised system. How the trust is established between the provisioning server and the client is beyond the scope of this thesis.

In password storage schemes such as [3], [4], [49], it is possible to use the passwords from multiple devices. In *PassPet*, the copy of the site-labels (from which the site-specific password

can be derived) is stored on a remote server. In all three, the corresponding plug-in should be installed on the device to use the protected passwords. Only *pwdHash* can work without the plug-in installed on the computer but at the cost of usability. [31] and [32] allow the credentials to be accessed from different devices as the credentials are stored in a central repository.

8. The system should allow policy based restriction of the usage of credentials.

Service providers may wish to enforce a policy restricting the usage and copying of the credentials and the system should enforce such access control checks. Onboard Credentials architecture provides an access control mechanism to ensure a secure usage of credentials. The access control method prevents unauthorized usage of credentials such that the service provider may direct the level of authentication required for using sensitive credentials. The authentication mechanisms may be a local PIN or in the future, biometric information. Our system currently does not provide any support for restricting or copying individual credentials from one device to another.

Table 1 compares various related work in the field of providing secure storage of credentials. Some criteria are not applicable and are mentioned as n/a. For example, the criterion "allows multiple credentials" is not applicable to Liberty Alliance SSO. Partially fulfilled criteria and are mentioned as "maybe". For example, in Software based password managers category, PassPet supports credential backup by storing the tokens needed for generating the passwords, but Password Multiplier and PwdHash do not.

Index	Criteria	Software based password managers	Liberty Alliance SSO	Java Cards	Hardware secured credentials repositories	Onboard Credentials
1.	**Provides protection against malware**	*maybe*	*no*	*yes*	*maybe*	*yes*
	Protects against dictionary attacks	*maybe*	*maybe*	*yes*	*yes*	*yes*
2.	**Allows multiple credential types**	*no*	*no*	*yes*	*maybe*	*yes*
	Allow third party development of different credential types	*no*	*no*	*maybe*	*no*	*yes*
3.	**Allows multiple credential instances**	*yes*	*n/a*	*yes*	*yes*	*yes*
4.	**Allows creation of credentials by users**	*yes*	*n/a*	*yes*	*maybe*	*yes*
	Allows creation by service providers	*no*	*n/a*	*yes*	*yes*	*yes*
	Allows creation by untrusted third parties	*no*	*n/a*	*no*	*no*	*yes*

	Allow provisioning of credentials	*maybe*	*n/a*	*no*	*no*	*yes*
5.	**Allows backup of credentials**	*maybe*	*n/a*	*no*	*no*	*yes*
	Copy credentials from one device to another	*no*	*n/a*	*n/a*	*no*	*yes*
6.	**Breaks existing services at server**	*no*	*yes*	*no*	*yes*	*no*
	Breaks existing services at client	*maybe*	*yes*	*n/a*	*maybe*	*yes*
	Breaks existing protocols	*no*	*yes*	*no*	*yes*	*no*
7.	**Usable on multiple devices**	*maybe*	*yes*	*yes*	*yes*	*yes*
8.	**Allow policy based restriction of usage of credentials**	*no*	*no*	*maybe*	*maybe*	*yes*
9.						
10.						

Table 1: Comparison of different credential management systems

6.4 Future Work

There is a need to have a policy framework that restricts the usage of credentials. It would be interesting to research and see who benefits from such restrictions and which party defines these policies – the service provider, the creator of the credential or the user himself. We also not yet clear as how the policies should be expressed, whether as scripts to be executed in the interpreter or as XML files. The future work could also constitute integration of the policy framework with the provisioning architecture. How exactly these policies are enforced, would an interesting research topic. Currently, only local PIN is supported. For access control policy, in the future the PIN may be replaced by biometric information that would be faster and more convenient.

Presently, only HTTP Digest credential type is supported. Other credential types could be implemented and the shortcomings and benefits of our system for those new types could be analyzed. Currently, our system does not implement the copying of the credentials from one device to another. Such a feature could be implemented in the future.

ARM TrustZone provides secure storage and secure execution environment. This could be used to store a device dependent OPK inside the secure hardware. TrustZone also provides a Trusted Interpreter that can execute generic trusted applications. This can be one alternative to the Lua Interpreter used in our architecture. Currently the applications at client need to be modified to make use of the Onboard Credentials system. In the future, PKCS #11 interface could be supported by Credential Server so that is possible for PKCS#11 aware applications to interact

with Credential Server without needing any modifications. Java Cards can be used to store multiple credentials securely. A possible implementation can be to use Java cards for providing secure storage and execution of the credentials.

The provisioning server can send data other than PPK to the device. The data could consist of a set of instructions that the provisioning server wishes to give to the Credential Interpreter. Such a feature could be useful for communicating with the script inside the Credential Interpreter. Confidentiality can be maintained between the provisioning server and the script inside the Credential Interpreter as both are aware of a common shared secret – the PPK. Applicability and potential uses of such a feature could be an interesting research topic for the future.

7 Conclusion

In this thesis, the problem of increasing number of credentials and issues with securely storing and using credentials is discussed. We conclude that hardware based storage schemes offer most security but at the cost of flexibility and infrastructure and deployment costs. The other extreme – software based storage offers flexibility and low costs but suffers from inherent weaker security. The trust in software based storage schemes is limited to the sanctity of the underlying software which can be easily compromised when used without any integrity protection mechanisms. Further, software based encryption and storage schemes use a PIN for cryptographic purposes and storage of this key poses a problem. The PIN itself is vulnerable to dictionary attack.

The Onboard Credentials system proposed in this thesis addresses some, if not all of the issues with which the alternative schemes suffer. We illustrate a novel way of expressing credential logic as scripts and use the secure environment offered by the platform to execute and store the credentials. Such a system allows third party development of different credential types and allows untrusted applications to utilize the secure environment, which is typically restricted otherwise. It provides protection against malware and dictionary attacks. It also, almost, seamlessly integrates into the existing infrastructure. We then present a proof-of-concept of this architecture as implementation across two platforms - Linux and Symbian and demonstrate that the proposed architecture can be implemented on platforms that are fast becoming ubiquitous.

With the proposed system, the user receives a Single Sign-On experience for most of the services although occasionally he may be asked for access control information such as PIN. In the future, access control may be augmented by faster and more convenient mechanisms such as biometric information. Thus, the results of the implementation reveal that it is indeed possible to have a system that provides secure storage, flexibility, ease of use and that addresses most of the shortcomings of the existing alternate solutions.

8 Bibliography

[1] **RSA.** SecurID Authentication. [Online] http://www.rsasecurity.com/node.asp?id=1156.

[2] **Mozilla.** Firefox Web Browser. [Online] www.firefox.com.

[3] **B.Ross, C. Jackson, N. Miyake, D. Boneh and J. Mitchell.** *Stronger password authentication using browser extensions.* Baltimore, USA, 2005. Proceedings of the 14th USENIX Security Symposium. pp. 17–32 .

[4] **J. Alex Halderman, Brent Waters; Edward W. Felten.** *A convenient method for securely managing passwords.* Chiba, Japan : ACM Press, 2005. Proceedings of the 14th International World Wide Web Conference. pp. 471-479. 1-59593-046-9.

[5] **Arvind Narayanan and Vitaly Shmatikov.** *Fast dictionary attacks on passwords using time-space tradeoff.* Alexandria, VA, USA : ACM Press, 2005. CCS '05: Proceedings of the 12th ACM conference on Computer and communications security. pp. 364-372. 1-59593-226-7.

[6] **Microsoft Corporation.** Trusted Platform Module Services in Windows Vista. [Online] April 25, 2005. [Cited: 12 22, 2006.] http://www.microsoft.com/whdc/system/platform/pcdesign/TPM_secure.mspx.

[7] **Trusted Computing Group.** Trusted Computing Group: TPM. [Online] https://www.trustedcomputinggroup.org/groups/TCG_1_0_Architecture_Overview.pdf.

[8] **Trusted Computing Group.** Trusted Computing Group: About Us. [Online] October 2006. [Cited: 12 22, 2006.] https://www.trustedcomputinggroup.org/about/TCGBackgrounder_revised_amp_oct_17_06.pdf.

[9] **Rick Wash.** The Security of Trusted Computing. [Online] April 13, 2004. [Cited: January 9, 2007.] http://www.citi.umich.edu/u/rwash/pubs/trusted-cwru-notes.pdf.

[10] **Ernie Brickell, Jan Camenisch, Liqun Chen .** *Direct anonymous attestation.* Washington DC : ACM Press, New York, 2004. CCS '04: Proceedings of the 11th ACM conference on Computer and communications security. pp. 132-145. 1-58113-961-6.

[11] **W.A. Arbaugh, D.J. Farber and J.M. Smith.** *A Secure and reliable bootstrap architecture.* s.l. : IEEE Computer Society, 1997. Proceedings of the 1997 IEEE Symposium on Security and Privacy. pp. 65 -71.

[12] **R Sailer, X Zhang, T Jaeger, L van Doorn.** *Design and Implementation of a TCG-based Integrity Measurement Architecture.* August 2004. Proceedings of the 13th USENIX Security Symposium. pp. 223-238.

[13] **LinuxBIOS.** [Online] http://www.linuxbios.org/.

[14] **ARM.** Designing with TrustZone - Hardware Requirements. [Online] [Cited: January 9, 2007.] http://www.arm.com/pdfs/TrustZone_Hardware_Requirements.pdf.

[15] **T. Alves and Don Felton, ARM.** TrustZone: Integrated Hardware and Software Security. [Online] July 2004. www.arm.com/pdfs/TZ%20Whitepaper.pdf.

[16] **S Ravi, A Raghunathan, S Chakradhar.** *Tamper resistance mechanisms for secure embedded systems.* s.l. : IEEE, 2004. Proceedings of 17th International Conference on VLSI Design. pp. 605-611.

[17] **Sun Microsystems Inc.** Java Card(TM) Platform Security. *Java Card Security White Paper.* [Online] 2001. http://java.sun.com/products/javacard/JavaCardSecurityWhitePaper.pdf.

[18] **RSA Security.** www.rsasecurity.com.
[19] **RSA Laboratories.** PKCS #11 v2.20: Cryptographic Token Interface Standard. [Online] June 28, 2004. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf.
[20] **P. Loscocco and S. Smalley.** *Meeting critical security objectives with security-enhanced Linux.* Ottawa : s.n., 2001. Proceedings of the 2001 Ottawa Linux Symposium.
[21] **National Security Agency.** Security-Enhanced Linux. [Online] http://www.nsa.gov/selinux/.
[22] **Spencer, Ray, et al.** *The Flask Security Architecture: System Support for Diverse Security Policies.* 1998. UUCS-98-014.
[23] **National Security Agency.** [Online] www.nsa.gov.
[24] **Wright, C., et al.** *Linux security modules: general security support for the linux kernel.* 2003. Foundations of Intrusion Tolerant Systems. pp. 213-226.
[25] Linux sysmask. [Online] http://wims.unice.fr/sysmask/doc/compare.txt.
[26] *LIDS Project- LIDS Secure Linux System.* http://www.lids.org.
[27] **Craig Heath.** *Symbian OS Platform Security: Software Development Using the Symbian OS Security Architecture.* s.l. : Symbian Press, 2006. p. 250. ISBN 0-470-01882-8.
[28] **Symbian.** Symbian Signed Whitepaper. *Symbian Signed.* [Online] https://www.symbiansigned.com/Symbian_Signed_White_Paper.pdf.
[29] **Wheeler, David A.** Secure programming for Linux and Unix HOWTO. [Online] http://www.dwheeler.com/secure-programs/.
[30] **Jeff Nelson, David Jeske, Google Inc.** Limits to Anti-Phishing. *W3C Security & Usability Workshop.* January 2006. Draft status as of 25th January 2006.
[31] **M. Lorch, J. Basney, D. Kafura.** *A Hardware-secured Credential Repository for Grid PKIs.* April 2004. 4th IEEE/ACM International Symposium on Cluster Computing and the Grid.
[32] **J. Marchesini, S. Smith.** *SHEMP: Secure Hardware Enhanced MyProxy.* New Brunswick, Canada : s.n., Oct 2005. Proceedings of the 3rd Annual Conference on Privacy, Security and Trust (PST).
[33] **R. MacDonald, S.W. Smith, J. Marchesini, O. Wild.** *Bear: An Open-Source Virtual Secure Coprocessor based on TCPA.* Department of Computer Science, Darthmouth College. August 2003. Technical Report. TR2003-471.
[34] **The Trusted Computing Group.** Trusted Computing Group Specifications. [Online] 2003. Available from: https://www.trustedcomputinggroup.org/specs.
[35] **D. Gustafson, M. Just and M. Nystrom.** *RFC 3760: Securely Available Credentials (SACRED) - Credential Server Framework.* April 2004.
[36] **A. Arsenault and S. Farrell.** *RFC 3157: Securely Available Credentials - Requirements.* August 2001.
[37] **T. Wu.** *RFC 2945: The SRP Authentication and Key Exchange System.* September 2000.
[38] **S. Bellovin and M. Merritt.** *Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise.* s.l. : ATT Labs Technnical Report, 1994. pp. 72-84.
[39] **S. Bellovin and M. Merritt.** *Encrypted Key Exchange: Password-based Protocols secure against dictionary attacks.* May 1992. Proceedings of the IEEE Symposium on Research in Security and Privacy. pp. 72-84.
[40] **D. Jablon.** *Strong Password-Only Authenticated Key.* September 1996.

[41] **RSA Laboratories.** *PKCS 12 v1.0: Personal Information Exchange Syntax.* s.l. : RSA Laboratories, June 24, 1999.
[42] **RSA Laboratories.** *PKCS #15 v1.1: Cryptographic Token Information Syntax Standard.* June 2000.
[43] **Microsoft Corp.** BitLocker Drive Encryption: Technical Overview. [Online] May 16, 2006. http://www.microsoft.com/whdc/system/platform/hwsecurity/BitLockerTechOver.mspx.
[44] **Vijay Auluck, Shelagh Callahan and Abhay Dharmadhikari, Intel Corp.** Manageable Identities. *Library Paper: Wireless LAN.* [Online] http://www.bizforum.org/whitepapers/intel-4.htm.
[45] **Nicolai Kuntze and Andreas U. Schmidt.** *Trusted Computing in Mobile Action.* [ed.] H.S. Venter, et al. s.l. : Information Security South Africa (ISSA), 2006. Peer-reviewed Proceedings of the ISSA 2006 from Insight to Foresight Conference. 1-86854-636-5.
[46] **Balfe, Shane, Lakhani, Amit D. and Paterson, Kenneth G.** *Trusted Computing: Providing Security for Peer-to-Peer Networks.* Konstanz, Germany : IEEE Computer Society, 2005. Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), 31st August - 2 September. pp. 117-124. 0-7695-2376-5.
[47] **Camenisch, Jan.** *Protecting (Anonymous) Credentials with the Trusted Computing Group's TPM V1.2.* 2006, SEC.
[48] **L. Detweiler.** *The snakes of medusa - internet identity subversion. Cypherpunks mailing lists.* 1993.
[49] **Ka-Ping Yee and Kragen Sitaker.** *Passpet: convenient password management and phishing protection.* Pittsburgh, Pennsylvania : {ACM Press, 2006. SOUPS '06: Proceedings of the second symposium on Usable privacy and security. pp. 32-43. 1-59593-448-0.
[50] **Wu, T.** *The Secure Remote Password Protocol.* San Diego, CA : s.n., March 1998. Proceedings of 1998 Internet Society Network and Distributed System Security Symposium. pp. 97-111.
[51] *Addressing Online Identity: Understanding the Microsoft Passport Service.* 3, September 2002, Information Security Technical Report, Vol. VII, pp. 65-80.
[52] **Liberty Alliance.** Liberty Alliance Project. [Online] www.libertyalliance.org.
[53] **OASIS .** Security Assertion Markup Language (SAML). [Online] [Cited: 01 01, 2007.] http://oasis-open.org/specs/index.php.
[54] **Chappel, D.** Introducing Windows Cardspace. Redmond, WA : Microsoft, 2006.
[55] **David Chappel.** Introducing Windows CardSpace. [Online] Microsoft Corp., April 2006. http://msdn2.microsoft.com/en-us/library/aa480189.aspx.
[56] **J.D. Tygar, A. Whitten.** *WWW Electronic Commerce and Java Trojan Horses.* Oakland, CA : s.n., November 1996. 2nd USENIX Workshop on Electronic Commerce. pp. 243-250.
[57] The Programming Language Lua. [Online] www.lua.org.
[58] TrouSerS - The open-source TCG Software Stack. [Online] http://trousers.sourceforge.net/.
[59] **Almesberger, Werner and Lermen, Hans.** Unix man pages: initrd(4). [Online] http://www.rt.com/man/initrd.4.html.
[60] GTK+ - The GIMP Toolkit. [Online] www.gtk.org.
[61] SQLite. [Online] www.sqlite.org.
[62] **Applied Data Security Group.** Applied Data Security Group - Projects - Trusted GRUB. [Online] [Cited: January 11, 2007.] http://www.prosec.rub.de/trusted_grub_details.html.

[63] **S. Chiasson, P.C. van Oorschot, R. Biddle.** *A Usability Study and Critique of Two Password Managers.* Vancouver : USENIX, 2006. Proceedings of the 15th USENIX Security Symposium.
[64] **B. Ramsdell.** *RFC 3851: Secure/multipurpose Internet mail extensions (S/MIME) version 3.1 message specification.* July 2004.
[65] SourceForge.net: TrouSerS. [Online] http://sourceforge.net/projects/trousers.
[66] **A. Pashalidis and C. Mitchell.** *Single sign-on using trusted platforms.* [ed.] C. Boyd and W. Mao. Bristol, UK : Springer - Verlag, 2003. 6th International Conference, ISC. Vol. 2851, pp. 54-68.