# The Workshop on Web Applications and Secure Hardware

John Lyle[1], Shamal Faily[1] and Marcel Winandy[2]

[1] Department of Computer Science, University of Oxford, UK
first.last@cs.ox.ac.uk
[2] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
marcel.winandy@trust.rub.de

## Workshop Overview

Web browsers are becoming the platform of choice for applications that need to work across a wide range of different devices, including mobile phones, tablets, PCs, TVs and in-car systems. However, for web applications which require a higher level of assurance, such as online banking, mobile payment, and media distribution (DRM), there are significant security and privacy challenges. A potential solution to some of these problems can be found in the use of secure hardware – such as TPMs, ARM TrustZone, virtualisation and secure elements – but these are rarely accessible to web applications or used by web browsers.

The First Workshop on Web Applications and Secure Hardware (WASH'13) focused on how secure hardware could be used to enhance web applications and web browsers to provide functionality such as credential storage, attestation and secure execution. This included challenges in compatibility (supporting the same security features despite different user hardware) as well as multi-device scenarios where a device with hardware mechanisms can help provide assurance for systems without. Also of interest were proposals to enhance existing security mechanisms and protocols, security models where the browser is not trusted by the web application, and enhancements to the browser itself.

## Committees

We are grateful to the following people for making this workshop possible.

### Organising Committee

- John Lyle (University of Oxford)
- Marcel Winandy (Ruhr-University Bochum)
- Shamal Faily (University of Oxford)

**Programme Committee**

- Andrea Atzeni (Politecnico di Torino)
- Andrew Martin (University of Oxford )
- Chris Mitchell (Royal Holloway)
- John Lyle (University of Oxford)
- Jun Ho Huh (University of Illinois at Urbana-Champaign)
- Kevin Jones (EADS Innovation Works, UK)
- Marcel Winandy (Ruhr-University Bochum)
- Marcos Cáceres (Data.Driven)
- Michael LeMay (Intel, USA)
- Nick Allott (Nquiringminds)
- Ronald Tögl (TU Graz)
- Shamal Faily (University of Oxford)
- Virginie Galindo (Gemalto)

**Additional Reviewers**

- Andrew Paverd (University of Oxford)
- Atanas Filyanov (Ruhr-University Bochum)
- Justin King-Lacroix (University of Oxford)

**Local Organisation**

- Michael Huth, Imperial College London

## Review Process

Every submitted paper received at least three reviews from qualified academic reviewers. The acceptance rate was 43%.

# Managing Access to Security Hardware in PC Browsers

Darmawan Suwirya, H. Karen Lu, and Laurent Castillo

Gemalto, Technology and Innovation, Austin, TX, USA
{darmawan.suwirya, karen.lu, laurent.castillo}@gemalto.com

**Abstract.** Web applications that require higher levels of security have various security options that can be deployed on the server. However providing security on the client consuming the service remains a challenge especially when the application runs in a web browser. While various types of security hardware can work with the user's computer, there is no convenient way for web applications to access such hardware from the web browser. To fill this gap, we develop a software tool called SEAM, which is a SEcure Add-on Management framework built on top of the web browser native extension framework. Through SEAM, web applications can access secure hardware in a controlled manner. SEAM was devised with usability and security at the core. It is flexible and convenient to use, easy to deploy, and works across major PC platforms and web browsers. This paper describes the rationale and design of SEAM, and illustrates its applications.

**Keywords:** SEAM, web application, secure hardware, web browser, browser extension, SConnect

## 1 Introduction

Web applications' ubiquity, ease of use, ease of deployment, and cross platform availability make them a preferred way for providers to deliver services, and for users to consume these services. Many such applications, especially those involving personal identifiable information and/or high value transactions, require higher levels of security and privacy protections. Service providers can deploy various security measures, such as firewall and intrusion detection, on the server side, but they still have difficulties to deploy any on the client side because their clients mostly run the application in web browsers, in which they have little control. Adversaries take this opportunity and utilize vulnerabilities of web applications to attack the client side for their own financial gains.

One way to enhance the security is to adopt multi-factor authentication that protect both web applications and users. This requires at least two factors among "what you know", "what you have", "who you are", "where you are", and so on. Beside the what-you-know factor (e.g. username/password), other factors typically require special hardware in acquiring the information. Although web browsers have been continuously improving in term of usability and security, there is still no convenient way for web applications to access such hardware from the web browser. To fill this

gap, we develop a software tool called SEAM, which is a SEcure Add-ons Management framework built on top of the web browser native extension framework. Through SEAM, web applications can access security hardware in a controlled manner.

SEAM is based on our previous work SConnect [1], which addresses the need of web applications to access smartcards in PC browsers. Most of the SConnect concepts and values are still valid and remain as the state-of-the art, e.g. architecture, design, implementation choices, and security, application and deployment models. As the use cases grew and learning from years of usage and deployment experiences, it became necessary to extend SConnect to support new usage and deployment scenarios.

The remaining content of this paper will explain about the new requirements in section 2. We'll describe an application example in section 3 and present the high level SEAM architecture in section 4. We'll explain the security mechanisms in section 5 and finally outline the SEAM implementation in section 6. Ensuring the integrity of web applications, web browsers, and user computers is out of the scope of this paper.

## 2 Requirements

SEAM intends to address the following new requirements while still satisfying the original SConnect requirements in term of *security*, *ease of use* (for both software developers and end users), *simplicity of deployment*, and *availability* (across major PC platforms and web browsers):

**Multi Hardware Support**. Some use cases need hardware connectivity beyond the smartcard. Some may even require more than one type of hardware connectivity concurrently. For example, a Match-on-Card [3] application would require connectivity to both biometric and smartcard reader devices in order to perform the user authentication.

**Hardware Driver Deployment.** The smartcard reader API has been well standardized through PCSC [2] and its driver usually comes by default with most of OS installations. Unfortunately, that is rarely the case for other types of security hardware. They usually come with their own set of drivers that users have to install manually. For a web application to get the best end user experience, it is critical for it to be able to easily deploy and install device drivers.

**Multi Versioning Support.** Earlier SConnect's design and implementation revealed issues with sharing the use of an add-on among multiple web applications. In practice, multiple versions of the add-on need to be installed and co-exist at the same time in the user's browser. This means that we need to have a way to allow sharing usage of SEAM among multiple web applications in a safe manner, without the risk of breaking web applications when upgrading another.

**Easy Customization.** Web applications often require customization of the add-ons, for instance for branding, consistency of the UI, or addition of new features. SEAM should provide a way to support and manage this kind of customization efficiently.

**Easy Deployment.** As we anticipate that there will be a large number of SEAM package variants as new features are added, automating add-ons deployment becomes a very crucial point to address, for both the web applications developers and end user experience.

## 3 Application

Let's take a real use case example of a web application offering a strong multi factors authentication service that utilizes both smartcard and biometric devices. In this application, the protocol is based on a simple PKI based challenge response authentication scheme: the server issues a challenge, sends it to the client for signature computation, and finally verifies the signature received.

On the client side, signature computation is performed by the PKI application on the smartcard. For added security and convenience, it's also utilizing the Match-on-Card application on the smartcard, replacing the traditional PIN verification. In this implementation, the application requires two connectivity add-ons and one functional add-on: PCSC smartcard reader connectivity, Futronic FS80 [4] fingerprint reader connectivity and FingerjetFX [5] fingerprint extractor function. Futronic FS80 also requires installation of its own driver.

To use the application, the user will be first requested to insert her smartcard. After detecting the insertion of a valid smartcard, containing PKI and Match-on-Card applications, and a valid fingerprint reader device, the application asks the user to scan her fingerprint. In the case where any fingerprint related component is missing, the application will provide a fallback choice on traditional PIN verification.


**Figure 1.** Match-on-Card application example

## 4 Architecture

Figure 2 below describes the high level, end-to-end architecture of the SEAM solution. SEAM consists of client and server side components. The client side component, called SEAM Extension, is in the form of a browser extension that

enables JavaScript code running in the browser to access native resources, e.g. security hardware. The server side component, called SEAM Library, is a JavaScript library that web applications download on-demand. It encapsulates the SEAM Extension functionalities in an easy-to-use application interface.

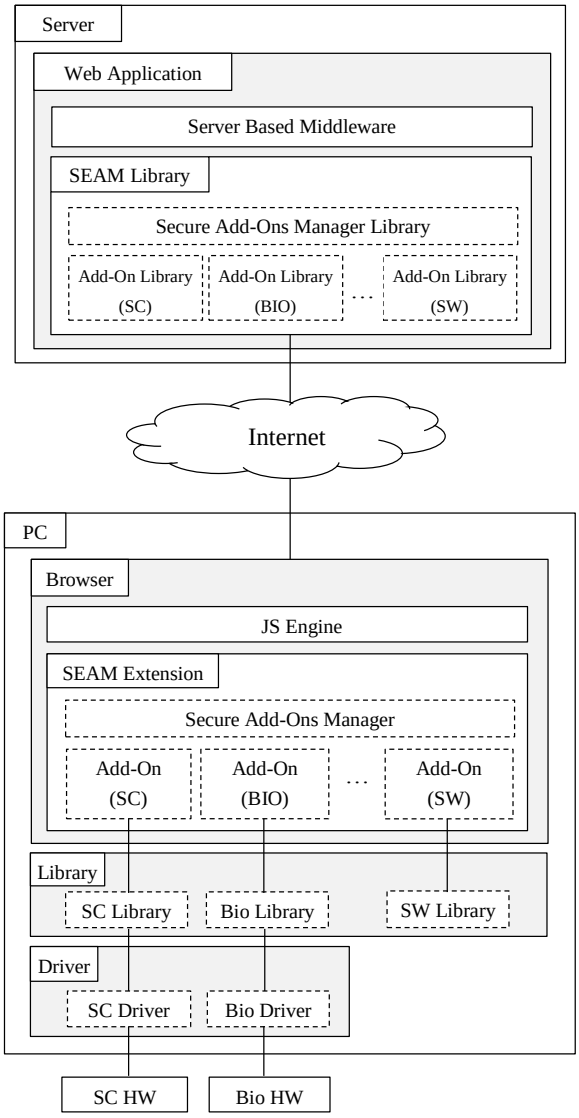


**Figure 2.** SEAM architecture

The SEAM Extension itself is made of two parts: the core and the add-ons. The core works as a secure add-ons manager on top of the browser's native extension framework: for instance, it offers services to install, list, and remove add-ons. The add-ons provide the actual implementation for accessing native resources. Delving a bit deeper, the core consists of the following components:

**Security Gateway**. This module is needed to prevent malicious web applications from accessing the security hardware through the add-ons. Without passing all the security checks and requirements imposed by this layer, the web application cannot use the SEAM Extension. The security mechanisms will be further detailed in the next chapter.

**Add-Ons Package Manager.** This module is responsible for managing the lifecycle of add-on packages, e.g. installation and un-installation, and provides the add-ons management user interface.

**Add-ons Runtime Manager.** This module is responsible for hot plugging add-ons on install and removal. It makes sure each add-on is correctly instantiated, and corresponding API calls are correctly mapped and routed at runtime.

## 5 Security Mechanism

Enabling web applications to access security hardware will enhance security. At the same time, this will also enlarge the attack surface. To protect the user, the security hardware, and the web application, SEAM employs several security mechanisms described below. When any of these security checks fails, SEAM prevents the web application from accessing the security hardware.

**Digital Signature.** SEAM Extension is digitally signed by using a code signing certificate issued by a trusted and well-known certificate authority. A signed extension instills confidence in users by validating the source of the extension.

**Enforce HTTPS.** In order to ensure secure communication with the remote web server and to prevent man-in-the-middle attacks (MITM), SEAM Extension mandates HTTPS between browser and remote web server.

**Server Validation.** During SSL handshaking, the browser receives and examines the website's SSL certificate. When the browser determines that certificate is invalid, it presents a warning to the user. Many users ignore the warning and continue [6], which may land them at a malicious website or make them fall victim to MITM attacks. To mitigate this risk, the SEAM Extension does an additional SSL certificate validation.

**Connection Key.** The Connection Key uniquely binds the SEAM-based web application with the SEAM Extension. The SEAM Extension issuer (e.g. Gemalto) signs the Connection Key with an asymmetric key whose public part is embedded in the Extension. The Connection Key contains additional information about the web

application server, like its add-ons usage permissions and SSL fingerprint. The owner of a web application must go through a strict vetting process defined by the issuer in order to obtain the Connection Key, which ensures controlled distribution.

**Revocation List.** SEAM Extension also implements CRL (Certificate Revocation List) mechanism to check the revocation status of the Connection Key.

**User Consent.** As the last step, after passing all the transparent security checks mentioned earlier, the SEAM Extension will ask for user permission at the beginning of each use. For convenience, it also provides a way to remember a user's decision for that particular domain name.

**Signed Add-Ons Package.** This mechanism is added in SEAM to ensure the integrity and security of the add-on package installation. The mechanism is similar to the one used for the Connection Key, utilizing a different set of key-pairs specific for this purpose. Signature is applied to a cryptographic hash of the add-on package payload, and embedded inside the add-on's manifest file.

## 6 Implementation

On the server side, SEAM and each of the add-ons come with their associated JavaScript libraries that will allow an easier integration in web applications. These JavaScript libraries abstract all the direct calls to the SEAM Extension and add-ons functionalities, and hide differences that might exist from one browser specific implementation to another.

On the client side, SEAM is implemented as a browser extension, native to each browser extensibility technology, targeting three major OS, i.e. Windows, Linux and OSX, and four major browsers, i.e. Internet Explorer, Firefox, Chrome and Safari. For Internet Explorer, it uses ActiveX and Browser Helper Object, and is packaged as an .exe installer. For Firefox, it uses XPCOM, and is packaged as .xpi bundle. For Chrome, it uses a combination of Chrome Extensions and NPAPI, and is packaged as .crx bundle. For Safari, it uses a combination of Safari Extensions and NPAPI, and is packaged as .pkg installer.

The resulting SEAM Extension installer packages are kept around a few hundreds KB each. It ensures fast and smooth download times. Install experience and access to add-ons management page are maintained as close as possible to each browser's native experience. This familiarity ensures a better installation and use experience.

SEAM Add-ons are packaged in a proprietary format, which is basically a zip file containing the following components:

- **addon.zip.** This file contains all the necessary components for the add-on implementation itself, e.g. security hardware connectivity.
- **driver.zip.** This file contains all the necessary components for driver installation. This file is optional and only present if a hardware driver deployment and installation is required.

- **icon.png.** This file is the icon file that will be used and displayed in the install dialog and add-ons management page.
- **manifest.json.** This file contains the declaration and description of the add-on package itself, encoded in JSON format. Most importantly, it also declares signature values for checking the integrity of driver.zip and addon.zip files contained inside the package.

## 7 Conclusion

SEAM is a secure add-ons management framework built on top of native browser extension mechanisms. It allows web applications to access native resources in a secure and flexible manner. When the web application needs to interface with a new hardware, SEAM also enables it to easily deploy the associated device driver. SEAM employs several security mechanisms in the implementation that will protect its use from various malicious attack scenarios. At the same time, the SEAM design puts the main focus on user experience, overall usability, and maintainability of the solution. SEAM is a work in progress. We continue improving its functionality and performance.

## 8 Acknowledgement

The authors gratefully acknowledge the feedback and support from Dr. Ksheerabdhi Krishna, Dr. Olivier Potonniee and Michael Hutchinson.

## References

1. Kapil Sachdeva, H. Karen Lu, Ksheerabdhi Krishna, *"A Browser-Based Approach to Smart Card Connectivity"*, IEEE Workshop on Web 2.0 Security & Privacy, Oakland, CA, 2009.
2. PC/SC Workgroup, *Specifications*, http://www.pcscworkgroup.com, V 2.01.11, June 2012.
3. Match-on-Card, http://www.matchoncard.com/what-is-moc/
4. Futronic, FS80 USB 2.0 Fingerprint Scanner, http://www.futronic-tech.com/product_fs80.html
5. DigitalPersona, FingerJetFXose, http://www.digitalpersona.com/fingerjetfx/
6. Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, Lorrie Faith Cranor, *"Crying Wolf: An Empirical Study of SSL Warning Effectiveness"*, The 18th Usenix Security Symposium, Montreal, Canada, Aug. 10-14, 2009.

# Using the W3C WebCrypto API for Document Signing

Nick Hofstede and Nick Van den Bleeken

Inventive Designers, Sint Bernardsesteenweg 552, 2660 Antwerp, Belgium,
https://www.inventivedesigners.com/

**Abstract.** This paper focuses on digitally signing documents as a specific use case for making secure hardware available to a web application. We explore the current options available to implementers and the drawbacks associated with each option. Then we look at the emerging Web Cryptography API developed by the W3C and discover missing functionality needed to implement this use case. Finally, we suggest a way to extend the API in order to support digitally signing documents using secure hardware.

**Keywords:** web applications, secure hardware, Web Cryptography API, W3C

## 1 Use Case

Converting paper processes to digital ones is an obvious trend within organisations. Documents are stored electronically and processes involving paperwork are transitioned to web applications. Not only is this more ecological, it enables a lot of opportunities to increase productivity and reduce costs. The past decade the paperless office has been inching ever closer to reality.

Not all paper documents are so easily replaced by their digital equivalent however. After digitizing the low-hanging fruit, attention now turns to processes that are more difficult to digitize. One of the common hurdles involves signatures. Often they can be replaced by auditing the web applications and using your corporate account to identify yourself, but some documents really do require a legally binding signature. Whereas on paper this involves taking a pen and making a scribble, the digital equivalent is often more complex. Web applications capable of generating documents digitally signed by the user are needed.

### 1.1 Signing Documents

The European Union recognizes three levels of digital signatures [1]: electronic signatures like scanned in handwritten signatures or email footers, advanced electronic signatures which are created using modern cryptography standards and qualified signatures which are advanced electronic signatures satisfying additional requirements. This last level is an advanced electronic signature made

using a key which has a qualified certificate associated with it. Qualified certificates can only be issued by certified certificate authorities and are only issued using the most stringent processes and only to the most secure type of keys. In return, qualified signatures must be considered equivalent to handwritten signatures by a judge whereas other electronic signatures are open to interpretation based on the actual details. The peace of mind this obligation provides makes them a requirement often encountered.

The requirement that a key should be kept under the sole control of the owner for qualified certification is interpreted to mean the key should be embedded in a hardware device. This to make sure undetected copying is impossible. In order to create a qualified signature, a European citizen will therefore always need a smart card or key dongle. A web application creating digital signatures should therefore be able to access the secure hardware of the user.

### 1.2 Electronic Identity Cards

The requirement for secure hardware prompted several European countries to issue electronic identity cards to their citizens. Currently countries like Belgium, Germany and many others provide electronic identity cards capable of creating electronic signatures [2–10] and more countries are planning on issuing them.

These cards, many of them issued mandatory, create a large group of users with access to secure hardware and a certificate ready to create legally binding electronic signatures. Governments have begun using these cards in web applications for the retrieval of official documents like birth certificates (after authentication) or submitting online tax forms (requiring a signature). Many more use cases for government and corporate applications can be thought of, but in this paper we will be focussing on signing documents.

## 2 Current Situation

Using these identity cards in web applications today is possible, but serious drawbacks exist.

### 2.1 Client-side TLS

The first applications that popped up were using client-side TLS for authentication. The user navigates to a website and is asked to authenticate herself. Most if not all of the identity cards come with middleware that either adds the certificate to the operating system's key ring allowing it to be accessed by browsers, or installs specific browser plug-ins registering the certificate for client-side use.

After the user has authenticated herself, consent with the content to be submitted to the application (like a tax form for example) is typically given by pushing a button. Proof of this consent is then carried by the auditing information logged by the web application. No real electronic signature is created and consent can only be inferred from the audit logs of the web application.

Even this simple scheme can lead to problems when one is not careful. In particular, revocation checks often need to be explicitly enabled at the SSL termination point as it is typically disabled by default. Additionally, client-side authentication suffers from a few other problems like not being able to log out of a session and somewhat bad usability due to it being needed before a connection to the server is established and a web page providing guidance can be shown. While this can be worked around using different domains, it would be preferable if the authentication process is initiated by the web page itself to allow for a richer user experience.

In any case, no actual qualified electronic signature is created, only an audit trail.

### 2.2 Java Plugin

If a qualified electronic signature needs to be created to sign a PDF document, or to create an archive containing a XAdES signature for example, the browser needs to access the smart card. This can be done using a custom plug-in, but a better option is to use a more widely available general purpose plug-in for this task. Chances are the user already has this plug-in installed which avoids going through an additional installation procedure. Using javascript and a Java applet, data can be shipped to and signed by the secure hardware. Typically, one of two methods is used. Either the applet probes for a library installed as part of the card's middleware using JNI, or the smart card is accessed directly using APDU commands available in standard java runtimes starting from version 6 [11].

The Java plug-in needs to be installed and the applet needs to be granted additional privileges. This is cumbersome, but until recently it was a reasonable approach. After a series of vulnerabilities in Java however, browsers either disable the plug-in, or only allow it to run when it is the latest version and has been given explicit permission. This degrades the user experience so much that using an applet is no longer a viable option.

## 3 Evaluation of the Web Cryptography API

Mainly driven by mobile applications being implemented as web applications, more and more functionality, including access to hardware like GPS sensors [12] and cameras [13], has been made available through javascript API's. When we first heard about a Web Cryptography API [14] being under development we looked at it to remove the dependency on a Java applet to access the secure hardware device and do the signing. With a use case like "The ability to select credentials and sign statements can be necessary to perform high-value transactions such as those involved in finance, corporate security, and identity-related claims about personal data." in the working group's charter [15] as a goal this did not seem far-fetched at all.

### 3.1 Design

In order to keep the scope of the API limited, the Working group has defined a very narrow scope for its main document. In order to stay away from concepts that are not portable between operating systems, cryptographic libraries or user agent implementations, provisioning operations or the discovery of cryptographic modules is considered out of scope.

While this might seem like a severe restriction, by supporting key generation functions it still allows for many cryptographic use cases. Indeed, as long as the keys are generated by the api, the provided operations (`encrypt`, `decrypt`, `sign`, `verify`, `digest`, `deriveKey`, `importKey` and `exportKey`) allow for a wide array of applications like secure messaging, data integrity protection of cached data and cloud storage.

On the more practical side of designing the API, the working group follows JavaScript best practices and tries to make every call that might take some time, or that might conceivably require user interaction to be asynchronous. This allows for the program flow of the javascript application to continue while waiting for the user the grant permissions, enter a pin number or calculations to be completed.

### 3.2 Signing

The key types and sign operations supported by the API are suited for the use case we have in mind. The issue has been raised whether "broken" cryptography algorithms like SHA1 and PKCS#1 v1.15 should be included, but for the sake of backwards compatibility and integration with server-side software it has been decided they remain. While we hope the world quickly moves on to the more modern alternatives which are supported as well, given the number of deployed smartcards implementing only these older algorithms, excluding them would seriously limit the applicability of the API today.

### 3.3 Key Discovery

Recognizing that not all use cases work with keys generated by the application itself, the group started work on key discovery in a separate working draft [16]. In order to limit the scope and driven by a use case focussing on trusted platform modules (TPMs) and digital rights management (DRM), the specification currently limits itself to "discovering named, origin-specific pre-provisioned cryptographic keys for use with the Web Cryptography API". While this allows for user agents to make keys stored on secure elements like TPMs available to web applications originating from a given domain under a known name, none of those adjectives are a good match for the signing use case.

While it might be possible to assign names to the keys provided by the secure hardware, in all naming schemes we could come up with it would be impossible for the web application to guess what that name might be. Additionally, the keys contained in the secure hardware wouldn't be origin-specific either. A user shall

want to use her smart card to authenticate herself to different web applications. Finally, the keys are not pre-provisioned. A way to prompt the user to (re)insert her smart card or USB dongle would be needed.

Key discovery as currently conceived by the WebCrypto Key Discovery draft isn't useful for discovering keys that reside on users' smart cards.

### 3.4 Certificate Based Discovery

Instead of name-based key discovery we believe attribute-based discovery of keys based on the key's algorithm or the accompanying certificate is necessary. By limiting the keys known to the browser by algorithm, issuing certificate authority, intended usage and other relevant properties, a short list could be presented to the user where she can pick the key needed to complete the action she started.

While this operation closely resembles selecting the appropriate key to use when setting up client-side TLS, it should be noted that this operation can be made asynchronous and can be initiated after a page has been presented to the user. It is conceivable that the user prompt takes the same form as other requests for access to specific resources like your location or camera. The main difference might be that this wouldn't have an "Allow" and "Disallow" button, but a "Select..." one popping up a dialog requiring further interaction.

## 4 Proposed Extension

Despite its current shortcomings, we believe the Web Cryptography API forms a solid basis. We joined the working group to help shape the API and make the use case outlined in this paper possible. We're currently working on a proposal that will extend the current API to enable the creation of web applications that use secure hardware for the creation of digital signatures.

### 4.1 Additional API

We propose a new `X509Certificate` class, and two new asynchronous methods. A first one to search for certificates and related keys, and a second one to enable exporting certificates to a byte array.

The `X509CertificateSelector` will take a dictionary containing filters. Filters include things like issuing certificate authority, usage flags, key algorithm, validity dates and others.

Invoking the `X509CertificateSelector` will create a subset of all known certificates known to the browser and initiate a selection procedure by the user agent. This procedure can start with a subtle banner to request access like the location or media capture API's. When clicked, the subset can be presented as a list to the user. Confronted with the list, the user will pick the certificate appropriate for the action she started and grant the web application access to the associated keys. This grant is origin-specific. When the keys are used an additional dialog window prompting for a pin code may be shown.

By keeping the user in the loop and requiring her to explicitly allow access to the certificate and keys stored in the operating system's store, this API can't be used to fingerprint users or glance information from unrelated certificates stored in the store.

The export functionality is necessary because the certificate (or certificates as you probably need the entire chain) used will likely have to be embedded in or associated with the digital signature that is being created.

### 4.2 Prototype

We are implementing enough of the WebCrypto API and the proposed extensions as a browser plug-in to validate this proposal [20]. The code is available under an Apache license on github [21].

*Note* This is an initial proposal and not all issues have been discovered or indeed resolved. We welcome comments, insights and code contributions you may have or want to share.

## References


1. European Parliament and Council. Directive 1999/93/EC on a Community framework for electronic signatures. 13 December 1999. Retrieved from http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:NOT
2. Austria. The Austrian electronic ID Card "Bürgerkarte". Retrieved from http://www.buergerkarte.at/
3. Belgium. The Belgian electronic ID Card. Retrieved from http://eid.belgium.be/
4. Estonia. The Estonian eID Card, "EstEID". Retrieved from http://www.id.ee/
5. Finland. The Finnish eID Card, "'FINEID'. Retrieved from http://www.fineid.fi/
6. Germany. The German electronic ID Card, "Personalausweis". Retrieved from http://www.personalausweisportal.de/
7. Italy. Carta di Identit Elettronica (C.I.E), the Italian electronic ID Card. Retrieved from http://www.halnet.it/cie/
8. Portugal. The Portuguese electronic ID Card, "Carto de Cidado". Retrieved from http://www.cartaodecidadao.pt/
9. Spain. The Spanish eID Card. Retrieved from http://www.dnielectronico.es/
10. Sweden. Fakta om nationellt id-kort (Facts about the national ID card). Retrieved from http://www.polisen.se/inter/nodeid=33378&pageversion=1.jsp
11. Oracle. Java Smart Card I/O API. Javadoc, package specification. Retrieved from http://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/
12. Geolocation API Specification W3C Proposed Recommendation 10 May 2012 Retrieved from http://www.w3.org/TR/geolocation-API/
13. HTML Media Capture W3C Last Call Working Draft 26 March 2013 Retrieved from http://dev.w3.org/2009/dap/camera/
14. World Wide Web Consortium. Web Cryptography API. W3C Working Draft 8 January 2013. Retrieved from http://www.w3.org/TR/2013/WD-WebCryptoAPI-20130108/
15. World Wide Web Consortium. Web Cryptography Working Group Charter. 3 April 2013. Retrieved from http://www.w3.org/2011/11/webcryptography-charter.html

16. World Wide Web Consortium. WebCrypto Key Discovery. W3C Working Draft 08 January 2013. Retrieved from http://www.w3.org/TR/2013/WD-webcrypto-key-discovery-20130108/
17. Cryptographic Token Interface Standard. Version 2.30, Editor Simon McMahon with Robert Griffin of RSA as project coordinator. RSA PSS mechanism parameters. Retrieved from http://www.cryptsoft.com/pkcs11doc/v230/
18. Mac Developer Library. kSecInputIsDigest constant definition. Retrieved from https://developer.apple.com/library/mac/search/?q=kSecInputIsDigest
19. Windows Dev Center. CryptSignHash function documentation. Retrieved from http://msdn.microsoft.com/en-us/library/windows/desktop/aa380280
20. Inventive Designers API proposal, version 79f54d9 Retrieved from https://github.com/InventiveDesigners/webcrypto-key-certificate-discovery-js/wiki/API
21. GitHub, Inc. Inventive Designers' webcrypto-key-certificate-discovery-js repository. Retrieved from https://github.com/InventiveDesigners/webcrypto-key-certificate-discovery-js

# Position Paper: Can the Web Really Use Secure Hardware?

Justin King-Lacroix[1]

Department of Computer Science, University of Oxford
justin.king-lacroix@cs.ox.ac.uk

**Abstract.** The Web has become the platform of choice for providing services to users in a range of different environments. These services can have drastically differing security requirements; for more sensitive services, it is attractive to take advantage of the wide deployment of secure hardware in the consumer space. However, the applicability of that hardware to any given security problem is not always obvious. This paper explores which use cases are appropriate, and which are not.

## 1 Introduction

The Web has transformed from a medium of information exchange to a platform for providing services. As desktop computers have become more powerful, Internet connections wider, and Web browsers more capable, Web applications have evolved to the point where they can now compete with their desktop counterparts on feature-set, performance, and price. Additionally, Web applications have several advantages: for one, the Web architecture requires application code to be downloaded from the remote server on every launch, placing the burden of application installation and maintenance on server operators (who are required to be IT-savvy) rather than end-users (who are not); for another, user data is frequently also stored on the server side, making it automatically accessible from anywhere on the Internet.

However, different applications can have very different security needs: an Internet banking application might need much stronger evidence of identity to authorise a financial transaction than a social networking website to authorise a post. This specific problem has traditionally been solved with two-factor authentication: in addition to their password (*something you know*), the user also possesses a secure element (*something you have*) which can generate a one-time code [15]. These elements can also be used to show liveness – certain banks ask users to re-authenticate before allowing certain types of transactions.

Much work has been done on user authentication in general, and two-factor authentication in particular [6]. These topics will therefore not be explored further in this paper. Instead, we ask the following two questions:

1. What other security properties might Web applications require?
2. Can secure hardware practically provide those properties?

## 2 Web application security

Concerning Web applications, we divide security properties into two broad classes: those internal to the application itself, guaranteed by programmers at design time, and those regarding – and provided by – its execution environment. Naturally, secure hardware can only help with the latter class.

To aid in discussion, we further divide the application's execution environment into three sections: its hardware environment, including the physical environment of that hardware; its software environment, which includes that both server- and client-side; and its network environment, since Web applications are inherently network-based.

Finally, we remark that any hardware security device can only be used to make meaningful assertions about components of the system that explicitly use it, or are implemented in terms of other components that themselves use it. This fact is well-understood, but is restated here because it is critical to the discussion that follows.

*Hardware* The application may require knowledge of hardware present or absent at the client, or may be required to present such information about the server. More strongly, it may be required to enforce restrictions on the nature of that hardware, its location, or other properties of its physical environment (such as the availability and quality of electrical power, or level of electromagnetic interference). Finally, the application may require knowledge or restriction of the identity of the devices on which it runs. The identity of the person currently using the device (or, indeed, whether there is even a user currently present) can also be considered part of its physical environment.

*Software* The application may wish to know or restrict the operating system (OS), OS patch level, Web browser or browser version, or installed software on any given device. The configuration of any of these components is also a concern; particular examples include the presence (or absence) of browser plugins, rights of the user account under which the browser is running, or modifiability of elements of the system's configuration. System state information, such as network location and address(es), CPU and memory load, and currently-running tasks, could also be required.

Equally, the application may be required to present such information about servers to its clients, so that the clients can make security decisions regarding the transmission of sensitive information.

*Network* Traffic between client and server may have confidentiality, integrity, or authenticity constraints. It may be that all traffic must be confidential to only the communicating parties, or that this only applies to some traffic. Equally, all traffic may need to be tamperproof, or only some, or may require different levels or types of integrity protection at different times. Finally, certain data may require proof of additional verification, such as obtaining explicit user consent, before it can be considered authentic.

## 3 Secure hardware in the consumer space

Consumers already have access to a variety of secure hardware. They are accustomed to authorising financial transactions using chip-and-PIN smart credit cards, inserting SIM cards into their mobile phones, and authenticating to banking (and other) Web applications using hardware tokens [14]. However, deploying a secure element may not always be cost-effective; in any case, all of these examples use secure elements purely for user authentication.

The Trusted Platform Module [1] is a near-ubiquitous example of a hardware security module (HSM). It provides services for encryption, signing, and key storage. Moreover, the availability of cryptographic keys and decrypted plaintext can be made to depend on the software state of the system, which can equally be asserted to a remote party using *remote attestation*. [8,9] Assertions about the composition and construction of that system are made by its manufacturer(s), via the *endorsement* and *platform credentials*. Finally, the TPM also provides a cryptographically-strong device identity.

Trusted Execution Environments (TEEs) are also widely deployed, with ARM TrustZone [3] and Intel Trusted Execution Technology (TXT) [7] – both CPU architecture extensions – the most common examples. These provide for isolated code execution, which can allow for more flexible key usage policies than those available with an HSM, although TEEs based on these technologies often use the TPM for its measurement and attestation services [2,11]. In particular, the ability to execute arbitrary code could be leveraged to guarantee explicit user consent.

Notably absent from consumer devices is a secure sensor access mechanism: sensor data may be used by both Web and native applications (insofar as browsers and OSes provide useful APIs to that end), but there is currently no way of assuring the accuracy of the data therby obtained.

## 4 Secure hardware in Web applications: the difficulties

The Web is a highly heterogeneous environment: Web browsers can be found for most operating systems, on several hardware architectures, across multiple device form factors, with different interface paradigms. Ideally, the browser should abstract over these differences, presenting a uniform rendering surface for Web applications. In reality, that abstraction must leak information about its underlying layers to the application; for example, for usability reasons, the application may change its interface depending on the local form factor.

This goes double for security-related services: in order for a security service to be useful, both the service itself *and its implementation* must be trusted. Web security APIs must therefore expose low-level information to the application, in order that a meaningful trust decision can be made. Further, this information must be verifiable, which, for a service backed by secure hardware, requires the exposure of hardware-level APIs directly to the application.

This presents two problems. First, for secure hardware, or indeed security services in general, to be exposed to Web applications, either the entire software

stack above them must be trusted – which presents a verification and maintenance nightmare to application developers – or their APIs must be carefully designed not to trust said stack – which limits how they can be abstracted over, inhibiting the paradigm of generic service-provision that has contributed to the success of the Web. Second, in order to support any such security services, all browsers on all platforms must be modified to support them, an issue showcased most recently by HTML5's video codec fragmentation.

In short, either application developers must be able to make trust decisions about entire software stacks of arbitrary complexity, or browsers and operating systems must be modified to support specific security technologies, with little room for abstraction.

## 5 Secure hardware in Web applications: the possibilities

All is not lost: while secure hardware is challenging to apply to the Web space in general, it has several specific, but significant uses.

*Hardware* Secure hardware can provide very few meaningful guarantees at the hardware level, in no small part because much security hardware operates independently of other hardware in the machine.

CPU instruction set extensions are the obvious exception to this rule, and indeed they are excellent tools for isolating executing code. System devices like the IOMMU perform a similar function for hardware devices. However, neither of these tools can assist with issues related to network security: network abstractions are orthogonal to the memory isolation that they both provide.

In essence, secure hardware can make very few statements about the hardware environment in which an application is executing, other than its own presence and state. To provide any stronger guarantees, it must (like the TPM) be very closely integrated into that environment.

*Software* The TPM already provides a cryptographically-strong device-specific identity. Applications needing to limit their distribution to a small set of trusted machines thus already have a mechanism for doing so; this is supported by existing PKI infrastructure in browsers and OSes already, and so requires no modification to either. (This type of limitation is most useful for applications with designated administration nodes, whose state can be carefully maintained.)

Credential storage is another deployed use of secure hardware – the TPM and smart cards being well-known examples – that can be easily integrated into nearly any application. However, the fact that those credentials are resident in a hardware security module, and the properties of that module, *must* be known to the application in order for an informed security decision to be made. Additionally, depending on the nature of both the stored credentials and the hardware, credential revocation may be problematic.

Perhaps the most interesting application is a combination of TPM-based runtime state attestation, TXT's virtualisation-based software security, and recent

work on TPM virtualisation [4,10,13]: applications with high security demands can distribute read-only virtual machine images containing a restricted and well-understood software stack. The hardware TPM can identify the hypervisor running on the bare hardware, and the virtual TPM can then identify the software state of the virtual machine [5].

*Network* Paradoxically, by virtue of being implemented at a layer of abstraction below both the Web application and, indeed, the browser, the network environment is the easiest to augment with secure hardware. Combining TPM-based keys with TLS is already a well-understood technique; more recent work involves using a TEE to provide more sophisticated key policy options than are possible with only the TPM [12].

## 6 Conclusion

Deployed consumer security hardware is an attractive choice for fulfilling the security needs of Web applications. However, its applicability is not always clear. In general, the need to modify the Web browser and the underlying OS limits the number and scope of technologies that can be usefully employed.

However, there are two major cases in which secure hardware can be of use. The first is in securing services at a lower layer of abstraction than the Web application, and whose security is thus negotiated and provided transparently. TLS, and developments thereon, are an excellent example of this kind of security. In a similar vein are TPM-based keys and cryptographic transactions, with their integration into OS cryptographic infrastructure in a manner transparent to the browser. However, some trust decisions must be able to be made about the implementations of each layer in use; this is an open problem.

The second is in a 'pass-through' mode, where security services are explicitly exposed by all layers in between the hardware and the application. This mode is more difficult to achieve, since every intermediate layer must be partially or fully rewritten to support the change. However, it has seen limited success with two-factor authentication – where the abstraction layers between the hardware interface and the Web application are few.

In either case, however, secure hardware is limited in the guarantees that it can provide. It cannot be used to make assertions about other devices in the machine unless those devices are designed (or forced) to interact with it. Equally, it cannot be used to make assertions about high-level constructs – such as OS user account or application identity – unless the software that implements those constructs directly interacts with it.

## References


1. Trusted Platform Module main specification 1.2 part 1: Design principles. Tech. rep., Trusted Computing Group
2. GlobalPlatform TEE System Architecture. Tech. rep., GlobalPlatform Inc. (2011)

3. ARM Holdings: ARM Architecture Reference Manual
4. Cooper, A.: Towards a trusted grid architecture. Ph.D. thesis, University of Oxford (2010)
5. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. ACM SIGOPS Operating Systems Review 37(5), 193–206 (2003)
6. Grosse, E., Upadhyay, M.: Authentication at scale. In: Proceedings of the IEEE Symposium on Security and Privacy. SP'13, vol. 11, pp. 15–22 (2013)
7. Intel Corporation: Trusted eXecution Technology (TXT) – Measured Launched Environment Developer's Guide
8. King-Lacroix, J., Martin, A.: BottleCap: a credential manager for capability systems. In: Proceedings of the 7th ACM Workshop on Scalable Trusted Computing (2012)
9. Martin, A.: The ten page introduction to trusted computing. Research Report CS-RR-08-11 (2008)
10. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: Efficient TCB reduction and attestation. In: Proceedings of the IEEE Symposium on Security and Privacy. pp. 143–158. SP'10 (2010)
11. McCune, J.M., Parno, B.J., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: an execution infrastructure for TCB minimization. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems. pp. 315–328. Eurosys'08 (2008)
12. Paverd, A., Martin, A.: Hardware security for device authentication in the smart grid. In: First Open EIT ICT Labs Workshop on Smart Grid Security. SmartGridSec12, Berlin (2012)
13. Perez, R., Sailer, R., van Doorn, L.: vTPM: virtualizing the trusted platform module. In: Proceedings of the 15th Conference on USENIX Security (2006)
14. Vasudevan, A., Owusu, E., Zhou, Z., Newsome, J., McCune, J.: Trustworthy execution on mobile devices: What security properties can my mobile platform give me? In: Trust and Trustworthy Computing, Lecture Notes in Computer Science, vol. 7344, pp. 159–178 (2012)
15. Weir, C.S., Douglas, G., Carruthers, M., Jack, M.: User perceptions of security, convenience and usability for ebanking authentication tokens. Computers & Security 28(12), 47–62 (2009)

# Towards Enhancing Web Application Security Using Trusted Execution

Cornelius Namiluko, Andrew J. Paverd, and Tulio De Souza

Department of Computer Science, Oxford University
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
Email: [firstname.lastname]@cs.ox.ac.uk

**Abstract.** The web continues to serve as a powerful medium through which various services and resources can be exposed or consumed through web applications. Web application platforms such as *webinos* facilitate communication between the various smart devices in a personal network. Although modern web applications use various cryptographic techniques for authentication and encryption, the security of these techniques is directly linked to the security of the private (secret) keys. Although various techniques exist to protect these keys, we argue that the use of secure hardware can provide stronger security guarantees. In particular, we describe our work-in-progress experiments towards using functionality provided by a Trusted Execution Environment (TEE) in web applications. These experiments include an implementation of the *webinos* platform integrated with ARM TrustZone technology. Our preliminary results are promising in terms of both the feasibility of implementing this architecture and the performance of the system.

**Keywords:** ARM TrustZone, GlobalPlatform, Trusted Execution Environment, webinos,

## 1 Introduction

Wide-spread use of smart mobile devices has opened up a range of new possibilities for web applications. Feature-rich web applications can be designed to use resources such as web cameras, GPS receivers and Near Field Communication (NFC) transceivers to provide interactive services. As a result, personal information such as location, messages and contacts is increasingly being exposed to the web leading to various security concerns about the confidentiality, integrity and availability of this sensitive information. Rather than relying on software alone to manage access to resources on these devices, it has been proposed that protection should be included as part of the hardware platform [3]. Grawrock [3] argues that a viable approach towards device security is through trusted execution — a paradigm in which non-security sensitive operations cannot influence sensitive operations even though both take place on the same platform. This provides the capability to control access to sensitive information and resources. However,

trusted execution functionality is normally provided at a low level of abstraction and in order for applications running at a higher level of abstraction to utilize this functionality, there must be mechanisms to expose the functionality in a flexible manner without compromising security.

In this paper, we propose that the security of web applications can be enhanced through a device-independent framework that enables web applications to utilize the functionality provided by trusted execution. We focus on *webinos*[1], a state-of-the-art platform for running web applications across multiple devices. Details of the webinos architecture are presented in Section 2. We propose that webinos can be enhanced to take advantage of trusted execution functionality provided by ARM TrustZone technology described in Section 3. In Section 4, we present our enhanced webinos architecture in which various cryptographic operations can be performed in the trusted environment in order to protect sensitive information such as cryptographic keys. We describe our ongoing experimental work in Section 5 and present a preliminary evaluation in Section 6. Our preliminary results are promising both in terms of the feasibility of implementing this architecture and the performance of the system.

## 2 webinos Architecture

Devices such as mobile phones, smart TVs, home appliances, energy meters and even cars are now capable of connecting to the Internet, leading to the so-called *Internet of things*. It is often the case that an individual will own multiple Internet-connected devices, each providing specific functionality or services. In order to take advantage of the composite set of services, there must be a mechanism for interconnecting these devices and facilitating resource sharing.

The *webinos* platform is an example of a system that achieves this objective. Based on node.js technology, the webinos platform provides an infrastructure for securely executing web applications across multiple devices. Through a set of APIs [9] such as geolocation, NFC and contacts, the webinos platform facilitates access to these services and resources by web applications. Using webinos, a device can access services and resources provided by a different device within the user's personal network (called a *Personal Zone)*. To enable this, each device runs a webinos component called a Personal Zone Proxy (PZP) and all devices in a particular zone are interconnected either through peer-to-peer communication or using a central component called the Personal Zone Hub (PZH).

There are several use cases for webinos [8] but for the purpose of this paper, we consider a specific scenario in which a user wishes to use a smart TV to watch a video stored on his or her smartphone. Figure 1 shows the overall webinos architecture relevant to this scenario.

Since the smart TV is webinos-enabled, a web application running on the TV can make the appropriate API calls to request the video from the smartphone.

[1] webinos is an EU-funded project and affiliate program aiming to define and deliver an Open Source Platform and software components for the Future Internet http://www.webinos.org

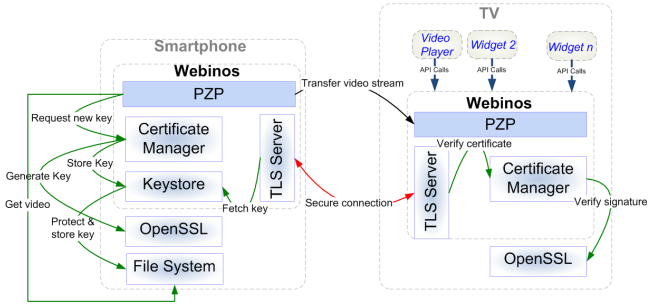


**Fig. 1.** Architecture of webinos — showing only the components discussed here

As part of this process, the TV must establish a secure communication link to the smartphone as shown in Figure 1. The key webinos components shown in the figure are explained below.

**Certificate Manager** — The certificate manager component provides functions for generating cryptographic keys and certificates for use in Transport Layer Security (TLS) connections. The implementation of this component relies on OpenSSL and runs on all webinos enabled devices. The component exposes a `genRSAKey` function, which returns either a 1024 or 2048 bit RSA keypair.

**Key Store** — The keys generated by the certificate manager and any passwords used in webinos are stored in a keystore. This allows for secure storage even across platform reset events. In the webinos architecture, the keystore functionality is provided using native platform mechanisms such as *gnome keyring*.

**TLS Server** — A TLS server is instantiated as part of webinos to support both client-server and peer-to-peer device connections. The server uses a secret key from the keystore and the cryptographic primitives provided by the underlying node.js platform which in turn uses OpenSSL to establish TLS connections.

## 3 Trusted Execution Environments

Various hardware-based mechanisms have been developed to provide enhanced security guarantees by building on well-established security principles such as defence in depth, least privilege and isolation. An architectural pattern that has emerged from these mechanisms is the *Trusted Execution Environment* (TEE). The fundamental concept of a TEE is that it allows specific software operations to be executed in isolation from the rest of the system. At present, the most common use case for a TEE is to provide a root of trust for other aspects of the system by performing certain security critical operations such as the management and storage of cryptographic keys in the TEE. Due to the hardware-enforced isolation from the rest of the system, it is possible to trust the software executed in the TEE without having to trust any other software on the system. In some cases, it is also possible to prove the degree of isolation to an

external entity. Various implementations of the TEE architectural pattern have been developed for different platforms by both industry and academia including the *Flicker* research project [4] and ARM TrustZone technology [1].

ARM Trustzone is a security technology that provides a hardware-enforced TEE on ARM platforms. Trustzone is implemented as a set of security extensions on certain processor cores based on the ARM version 6 or version 7 architecture [1]. As shown in Figure 2, Trustzone partitions the platform hardware into two distinct *worlds*, namely the *normal world* and the *secure world*. With the support of other TrustZone-enabled platform hardware elements, this ensures that components running in the normal world do not have access to resources belonging to the secure world. The general approach is to run a minimal secure kernel in the secure world in parallel with a feature-rich OS in the normal world.

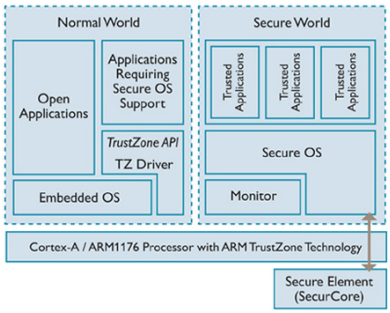


**Fig. 2.** ARM TrustZone Software Architecture [1]

By implementing TrustZone as a processor extension, it is possible for a single core to execute both the normal world and the secure world in a time-sliced manner thus eliminating the need for a separate security co-processor whilst still ensuring full isolation. Transitions between the normal and the secure worlds are managed by a secure software component running in a new processor mode called secure monitor mode, shown as the *Monitor* in Figure 2. Whilst TrustZone hardware is already available, the software required to use this functionality is still in a state of flux. Recently, the GlobalPlatform consortium released the GlobalPlatform TEE API Specification [2], in an effort to standardize access to TEE functionality across different devices.

## 4 TEE-enhanced webinos Architecture

The philosophy of enabling uniform and secure resource access across multiple devices makes webinos a suitable platform for exposing secure hardware functionality to web applications. The webinos platform already facilitates discovery and access control for cross-device resource sharing. This platform uses the device OS and native applications to expose several APIs to webinos widgets (web

applications that run on devices). These APIs are an abstraction of the resources provided by each device. We view the functionality provided by a TEE as another type of resource that could be made accessible in a similar manner. We argue that the security of web applications can be enhanced through the use of a TEE and that the webinos architecture can be extended to provide this functionality to web applications with minimal modifications. Figure 3 illustrates our proposed webinos architecture enhanced with trusted execution in which the platform is divided into an isolated *secure domain* for security-sensitive operations and a *feature-rich domain* for all other operations.

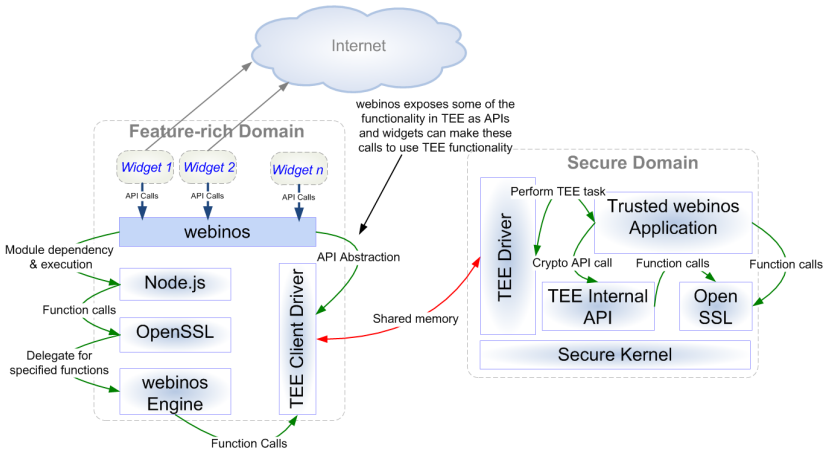


**Fig. 3.** An architecture of webinos enhanced with TEE

### 4.1 Feature-rich Domain

The feature-rich domain includes the main device OS and the majority of webinos functionality as well as any installed webinos widgets. The following components from this domain are important in our enhanced webinos architecture:

**OpenSSL with Modular Engine Functionality** — Since version 0.96, the OpenSSL library has been designed to support ENGINES [5] — modules that can be dynamically or statically loaded to provide alternative implementations of cryptographic functions. OpenSSL ENGINES are sometimes used for interfacing with secure hardware such as a smartcard or Trusted Platform Module (TPM)[2].

**The webinos TEE OpenSSL Engine** — As shown in Figure 4, the webinos TEE engine is an OpenSSL ENGINE that runs in the normal world and communicates with the secure world through the TEE client driver. This allows

[2] https://github.com/ThomasHabets/openssl-tpm-engine

specific cryptographic operations to be performed in the secure world. In particular, the webinos TEE engine provides encryption, decryption and signature operations using keys that are only accessible in the secure world. These keys are referenced using *key_id* fields as shown in Listing 1.1.

**Listing 1.1.** Sample of cryptographic operations provided by webinos TEE engine

```
int webinos_private_encrypt(char* key_id, ...);
int webinos_private_decrypt(char* key_id, ...);
int webinos_sign(char* key_id, ...);
```

### 4.2 Secure Domain

The enhanced architecture uses the principle of least privilege to isolate a secure domain from the feature-rich domain. The following aspects of the secure domain are important in our enhanced architecture:

**Secure kernel** — The secure kernel provides common functionality for the secure domain and ensures that all operations in this domain are completely transparent to the feature-rich domain. The secure kernel also enforces strict isolation between different trusted applications in the secure domain.

**Trusted webinos application** — The trusted webinos application is responsible for authenticating the source of the requests from the feature-rich domain. This could be achieved by inspecting the requests or using technology similar to the integrity measurement architecture (IMA) as described in [6]. This trusted application also provides key life-cycle management functionality. This application is based on the GlobalPlatform TEE Internal Specification [2], but can also utilize the cryptographic library included in the secure domain.

**Secure domain cryptographic library** — The kernel in the secure domain is built with support for cryptographic functions. This is provided by libraries such the OpenSSL or PolarSSL that have been installed in the secure domain.

## 5 Case Study: Securing TLS Sessions

It is informative to consider the scenario introduced in Section 2 because it involves a complete life-cycle of a cryptographic operation, which is a common scenario in most Internet-connected systems. We consider how TLS sessions are established using the enhanced webinos architecture. This process involves securely generating, storing and using cryptographic keys in TLS connections as shown in Figure 4.

In order to demonstrate the feasibility of realizing this new architecture, we have undertaken an investigation to understand how webinos makes use of cryptographic operations and we have performed a preliminary experiment in which an OpenSSL engine is modified to take advantage of functionality provided by a TEE. In this experiment we do not explicitly consider secure persistent storage of cryptographic keys. This could be achieved using functionality from the GlobalPlatform specification (e.g. `TEE_CreatePersistentObject`) [2] and supported by technology such as a TPM MOBILE also running in the TEE [7].

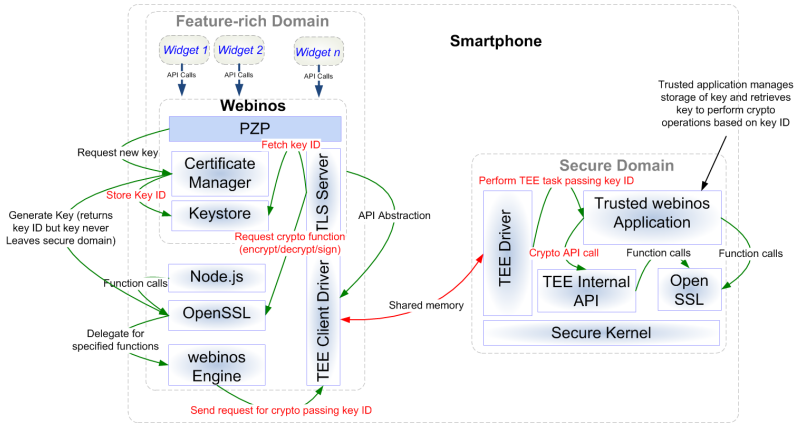


**Fig. 4.** The workflow of webinos when enhanced with TEE

### 5.1 Development Environment

The OpenVirtualization project[3] aims to create an open source software stack from TrustZone that implements the GlobalPlatform TEE API [2]. In order to accelerate the development of TrustZone software, Winter et al. [11] have created a software development and emulation framework for ARM TrustZone as part of the SEPIA research project. This framework includes a modified port of the *qemu-system-arm* processor emulator that supports TrustZone extensions for certain ARM processors and is available as an open source project called *qemu-trustzone*[4]. In order to develop a proof-of-concept implementation of our enhanced architecture, we use a combination of the OpenVirtualization software and the qemu-trustzone emulator. The OpenVirtualization project provides a software development kit (SDK) for building the normal world and secure world kernels. At present, the SDK only supports Linux 2.6.38-rc7 as the normal world kernel and so we are using a Debian "squeeze" root filesystem image for the ARM architecture. We have successfully built the OpenVirtualization images, as well as the normal world and secure world kernels and have run the system using qemu-trustzone. We have used this emulated system for our initial experiments and obtained some preliminary results as described in the following section.

## 6 Evaluation

In this section, we evaluate the extent to which our enhanced webinos architecture makes TEE functionality available to mobile web applications. webinos provides an abstraction layer, in the form of APIs, that enables web applications

[3] http://www.openvirtualization.org/
[4] https://github.com/jowinter/qemu-trustzone

to make use of resources and services provided by other devices. Building on this approach, there are two primary ways in which webinos can be used to bring the security benefits of a TEE to web applications: The first is to provide direct access to TEE functionality through dedicated APIs and the second is to enhance the security of current webinos APIs using TEE. An example of the first approach would involve extending the Secure Elements API in webinos [10] to allow access to TEE functionality. An example of the second approach would be to redesign an existing component such as a policy manager so security-sensitive parts of the component are executed in the secure domain. By leveraging open standards upon which webinos is built, the enhanced architecture, therefore, enables TEE functionality to be exposed to web applications in a platform neutral manner. This allows web applications to use the same interfaces to TEE functionality across different devices, leading towards cross-platform security.

Although the implementation of the architecture is ongoing, preliminary results are promising. We have successfully run webinos in the normal world OS and have proxied all cryptographic functions through the engine with minor modifications. We are able to make calls to the secure world using the TEE client library provided by the OpenVirtualization SDK and have successfully built part of the OpenSSL library into the secure world. We are currently implementing the communication interfaces between the engine the secure domain. One of our objectives is to demonstrate the feasibility of implementing this proposed architecture and, as discussed in the previous sections, this appears to be feasible and will only require minimal changes to webinos or the underlying node.js platform. Another important consideration for this kind of architecture is system performance. Using the current implementation, a preliminary performance analysis has been performed to determine how this compares to other implementations such as [6]. The overall performance of the system in [6] is limited by the performance of a separate cryptographic co-processor, the TPM, which has not been designed as a high-performance device. In [6], the overall time taken for the primary cryptographic operation (an RSA signature operation) was in the order of 1000 milliseconds but in our current experiment, preliminary tests have shown that the equivalent operation can be performed in the order of 10 milliseconds because all computations take place on the main CPU. This level of performance latency would be essentially unnoticeable in web applications because it is lower than average communication latencies over the Internet. Although this result has been obtained using the TrustZone emulation framework [11] on a different hardware architecture, we expect to demonstrate similar results on real-world hardware in the near future.

## 7 Conclusion

Web applications make use of resources on mobile devices such as cameras or navigation systems to create an interactive experience for users. As a result, they are becoming an attractive channel for attacks against mobile devices. Web applications can take advantage of platform features such as Trusted Execution

Environments to protect sensitive information. However, TEE features are normally provided at a low level of abstraction and so in order for these features to be useful for web applications, a platform independent and standards-based approach is essential. We have proposed, demonstrated and tested an enhanced webinos architecture augmented with ARM TrustZone technology. Although this work is ongoing, preliminary results from our experiments are promising in terms of feasibility of implementation and system performance.

## Acknowledgements

The work described here is funded by the webinos project with collaboration from the SEPIA project. The authors thank Johannes Winter, Ronald Toegl and Martin Pirker for their support and insights on TrustZone technology and *qemu*. We also thank the OpenVirtualization community and support team for their assistance. Andrew Paverd is funded by the *Future Home Networks and Services* project sponsored by British Telecom.

## References

1. ARM. TrustZone. Last accessed: April 2013 http://www.arm.com/products/processors/technologies/trustzone.php.
2. GlobalPlatform. TEE System Architecture v1.0; TEE Internal API Specification v1.0; TEE Client API Specification v1.0;. Technical report, 2011.
3. D. Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 2009.
4. Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for TCB minimization. In *Eurosys '08 Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, volume 42, page 315, April 2008.
5. OpenSSL Project. OpenSSL Engine Documentation. http://www.openssl.org/docs/crypto/engine.html Last accessed: April 2013.
6. Andrew J. Paverd and Andrew P. Martin. Hardware Security for Device Authentication in the Smart Grid. In *First Open EIT ICT Labs Workshop on Smart Grid Security - SmartGridSec12*, Berlin, Germany, 2012.
7. Trusted Computing Group. TPM MOBILE with Trusted Execution Environment for Comprehensive Mobile Device Security. Technical Report June, 2012.
8. webinos Consortium. webinos Use Cases and Scenarios - v1.0. Technical report, 2009. Last accessed: April 2013 http://www.webinos.org/content/webinos-Scenarios_and_Use_Cases_v1.pdf.
9. webinos Consortium. webinos Phase II API Specifications. Technical report, 2012. Last accessed: April 2013 http://www.webinos.org/content/webinos-phase_II_device,network,and_server-side_API_specifications.pdf.
10. webinos Consortium. webinos Secure Elements API, 2012. http://dev.webinos.org/deliverables/wp3/Deliverable34/secureelements.html Last accessed: April 2013.
11. Johannes Winter, Paul Wiegele, Martin Pirker, and Ronald Toegl. A flexible software development and emulation framework for ARM TrustZone. In *INTRUST 2011*, 2011.

# A Path Towards Ubiquitous Protection of Media
## (Position Paper)

Ronald Toegl, Johannes Winter, and Martin Pirker

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{rtoegl,jwinter,mpirker}@iaik.tugraz.at

**Abstract.** Mobile devices have become powerful and user-friendly. At the same time they have become a hosting platform for a wide variety of services. Naturally, the interests of the various stakeholders on a mobile platform are not the same. Thus, there is demand for a strict separation model of services on mobile devices. In this paper, we outline a possible approach to enable a Secure Media Path on mobile devices. Our approach aims to support the needs of the different stakeholders, with respect to openness, content protection and client privacy. The architecture takes into account the resource constraints of mobile devices.
**Keywords**: Trusted Computing, TrustZone, Multi Screen, Digital Rights Management

## 1 Introduction

Enabling a Secure Media Path (SMP) on mobile devices is a non-trivial problem, as such small platforms are restricted by multiple constraints. For example, such a device needs to be open enough to enable modifications or replacement of parts of the software stack, while at the same time the mobility aspect of the platform requires efficient adaption to changes in environment and connectivity.

A SMP aims to guarantee the protection of content, while at the same time needs to operate within the contraints of the mobile platform. On a high level, a SMP consists of software and hardware components that allow to protect content (and protection keys) and which enforce associated usage rules. The use of dedicated hardware elements and personalization capabilities make the bypassing of the content protection a significantly hard problem.

The basic platform requirement is isolation. The isolation (or virtualization) features of state-of-the-art CPUs and chipsets enable the hardware assisted separation of small, secure code and data proceesing from common rich OS code and data processing. A strictly reduced Trusted Computing Base (TCB) can be more easily checked for integrity and thus then tasked with the sensitive operations.

*Outline* The remainder of this paper is organized as follows. In Section 2 we motivate an example scenario for a SMP on modern mobile platforms. Then, Section 3 gives a short summary on the state-of-the-art of security enhanced hardware platforms. Section 4 presents our proposal for a SMP architecture. The paper concludes in Section 5.

## 2 A Future Scenario of Media Consumption

We imagine the capabilities of media consumption in the near-future and describe it in the following usage scenario: Amadeus has just bought a new smartphone. On the way home, he explores his new device. Among the pre-installed apps, Amadeus finds an app for a large cloud provider's payment service and its movie store. In the App Store, he finds additional applications to access his bank account at Sparkasse, a bitcoin client, a wallet app, and free apps from various broadcasters including free-tv and pay-tv networks.

He launches the Sparkasse App, which uses a specific, secure part of his device's screen to display the message provided by Sparkasse. He enters his account information to access his account. Next he checks the movies available in the pay-tv video store and picks one. Upon touch of the "Play" button, he is presented with a menu of payment options, including a service backed by a federated cloud ID, his account at Sparkasse, and bitcoins. Each account shows the available amount of money. Amadeus taps the Sparkasse account. When the screen changes, he immediately recognizes the visual brand of the Sparkasse app asking him to confirm the payment. The mobile phone indicates that the Sparkasse app is indeed the authentic origin of the payment dialogue. After confirmation, the movie starts to stream and play immediately. The bus arrives. Amadeus sits at the back of the bus and enjoys the first thirty minutes.

At home, he puts his new phone on the coffee table and switches on his smart TV set. As soon as switched on, he sees a menu where he can choose to continue to watch the current movie scene on the TV set as a 3D movie in 4k quality with surround sound, without a need to purchase the movie again. While the movie plays on the smart TV, the phone goes to sleep to conserve battery. His wife joins in to watch the second half of the movie, when Amadeus's phone rings. He picks the phone and goes to the other room to have the call. The smart TV keeps showing the movie so that Amadeus's wife keeps watching it.

This short story illustrates a number of elements not possible today: First, the consumer has full control over the selection of content and payment methods. For privacy-sensitive apps, security is made tangible to the user using a secure portion of the screen. Use credentials are protected from the rich OS installed on the mobile device. It is also important to the consumer that media delivery is seamless across different devices, and the mobile device can act as a media gateway for the home. Different devices that display the same media may offer device-specific enhanced experiences, and cooperate closely and seamlessly. Finally, the content that is consumed is well-protected. There is a strong separation between the protected media content on the device and any apps running on the rich OS. Without it, a pay-tv provider would not agree to stream their media to the device. Yet, content protection is transparent to the user. Equally important to the content providers: they can either provide their content to standard apps and service providers that handle payment in a transparent way, or provide their own apps that link into a secure media interface that is the same across devices. This set of features is not yet possible with current day devices.

## 3 Security Enhanced Platforms

Modern state-of-the-art platforms provide distinct security support features. They enable enhanced cryptographic primitives, strictly isolated processing and (remote) attestation of the platform state. We now give a short overview on these technologies.

The term *Trusted Computing* has been mostly established by specifications of the Trusted Computing Group (TCG), an industry consortium. The core component, the Trusted Platform Module (TPM) [14], is a low-cost hardware security module that is physically bound to its host device. A tamper-resilient integrated circuit contains implementations for public-key cryptography, key generation, cryptographic hashing, and random-number generation. The TPM provides high-level functionality such as collecting and reporting the current system state, and providing evidence of the integrity and authenticity of this measurement, known as *Remote Attestation*. Consequently, a sucessful TPM-enabled remote attestion of a platform can provide the confidence that the platform is in the correct state to be host for a secure media path environment.

### 3.1 ARM TrustZone

One of the dominant processor architectures employed in current mobile and embedded devices is the ARM architecture. Current ARM-based processor designs span a wide range of application fields, ranging from tiny embedded devices (e.g. ARM Cortex-M3) to powerful multi-core systems (e.g. ARM Cortex-A9 MPCore). Also, ARM introduced a set of hardware-based security extensions called TrustZone [2] to ARM processor cores and on-chip components.

The key foundation of ARM TrustZone is the introduction of a *secure world* and a *non-secure world* operating mode. This secure world and non-secure world mode split is an orthogonal concept to the privileged/unprivileged modes already found on earlier ARM cores. On a typical ARM TrustZone core, secure world and non-secure world versions of all privileged and unprivileged processor modes co-exist. For the purpose of interfacing between secure and non-secure world a special Secure Monitor Mode together with a Secure Monitor Call instruction exists. The AMBA AXI bus in a TrustZone enabled system carries extra signals to indicate the originating world for any bus cycles. Thus, TrustZone aware System-On-Chip (SoC) peripherals can interpret those extra signals to restrict access to secure world only; a secure world executive can closely monitor any non-secure world attempts to access secure world peripherals. To summarise, an ARM TrustZone CPU core can be seen as two virtual CPU cores with different privileges and a strictly controlled communication interface.

### 3.2 Trusted Execution Environments

Previously, ARM had published its own TrustZone software API specification [1]. Together with Trusted Logic, ARM has developed a closed-source TrustZone software stack, complementing the TrustZone hardware extensions. ARM has

since donated its TrustZone API to the GlobalPlatform industry association and this has developed into the Trusted Execution Environment (TEE) Client API [5]. It allows an application in the "non-secure world", which typically runs a rich-OS such as Google Android or Microsoft Windows Mobile 8, to communicate with the "secure world". ARM has also been working with other companies to develop the TEE Internal API [6] that interfaces between a Trusted OS, running in the secure world and a Trusted Application.

Today, all modern ARM-based Smartphones (Cortex-A CPU based) include a TEE based on SoCs by manufacturers like Qualcomm, Samsung, Nvidia, and Texas Instruments. Accordingly, TEEs are already deployed on the field since for several years, featuring Trusted OSes currently made by Trusted-Logic/Gemalto (Trusted Foundation) or Giesecke & Devrient (Mobicore). Moreover, ARM, Gemalto and Giesecke & Devrient and others have recently created the "Trustonic" Joint Venture on TEE Trusted OS and its ecosystem of services.

### 3.3 Research on TEEs and TEE Applications

Several scientific publications deal with proposals for secure mobile and embedded system designs based on the ARM TrustZone security extensions. Use of ARM TrustZone hardware to securely manage and execute small programs ("credentials") were described in [9] and [3]. A similar runtime infrastructure was used by the authors of [4] to implement a mobile trusted platform module. Similarly [12] proposes a trusted runtime environment utilizing Microsoft's .NET Framework inside the TrustZone secure world. With the use of a managed runtime environment the authors try to benefit from the advantages of a high-level language combined with hardware security and isolation mechanisms provided by the underlaying platform.

A large number of publications deal with possible applications of ARM TrustZone to implement, for example, digital rights management [8], cryptographic protocols [15], mobile ticketing [7] and [10], wireless sensor networks [17], or anonymous payment for remote cloud service resource consumption [11].

An approach of using a modified Linux kernel acting as secure world operating system for a mobile virtualization scenario has been discussed in [16]. This work showcases an experimental open-source software environment for experiments with ARM TrustZone in combination with Trusted Computing primitives. The software framework offers a prototype kernel running within a trusted environment and features a software based Trusted Platform Module hosted in a TrustZone protected runtime environment and an Android operating system accessing it through a high-level API.

## 4 Proposed Architecture

Media processing is generally a resource intensive task with high demands of processing power memory and bandwidth, especially with high definition material. Traditional, stationary set-top boxes employ various types of smart cards in

combination with specialized system-on-chip and board-level designs to provide adequate performance as well as protection of content data, which is delivered and processed on the device. Commonly, these traditional set-top boxes are closed special-purpose embedded systems with well-defined restrictions on the software and configuration changes an end-user of the device is able to perform. However, on smart phones and tablet computers, users expect to be able to customize their devices to a great degree, for example by installing all kinds of third-party applications.

Typical transformations on the stream include signal processing tasks like decompression, color-space conversions, equalization of audio signals, and scaling or rotation of video signals. Current mobile computing platforms often implement at least parts of these computationally intensive tasks directly in hardware to reduce the computational requirements and power-consumption of the platform. To support secure media paths, it is necessary to securely integrate additional transformation steps in the basic architecture. Such steps include content decryption and surrounding frameworks like policy engines and key management. To avoid unintended and unwanted interference between arbitrary applications running on the platform and the SMP core services, it is necessary to introduce two separate security domains on the platform. Due to the bandwidth requirements of high-quality video content, encryption algorithms may be moved into dedicated hardware blocks.

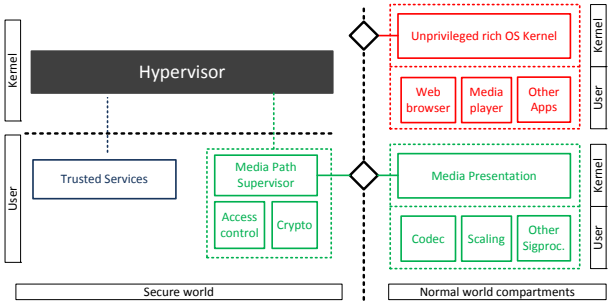


**Fig. 1.** Layout of the proposed SMP architecture.

We propose to leverage the TrustZone hardware-extensions to establish a software-based SMP. In our proposed architecture, the components constituting the core implementation of the SMP itself are protected against direct interference from malicious applications through software and hardware isolation, and virtualization techniques. By executing the content processing in domains isolated from the rest of the system, the use of media-processing software components provided by the content provider along with the content become possible. Because media processing is isolated from the rich OS, the interests of the content provider to protect their content from piracy are preserved. At the same

time, such software is not able to subvert the security and privacy of the user because it can access the relevant parts of the media pipeline only. Secure handover between devices can be supported by remote attestation, which can also be easily done over Bluetooth or Near-Field-Communications [13].

ARM TrustZone divides the platform into multiple worlds. The so-called secure world is controlled by the highly secure and low-complexity trusted OS. Besides the trusted OS, the platform executes one or multiple instances of a rich OS such as Android in the so-called normal world. There, playback is controlled and presented to the user either by specialized apps, or simply in the HTML5 compliant web browser. Thereby, our approach retains compatibility with current mobile operating systems. Because the secure world is hidden from any software executed in the normal world, information that is critical for security and privacy can be protected by processing it in the secure world only. Furthermore, hardware components that are critical for the SMP can be explicitly assigned to the secure world, eliminating attack vectors for sniffing high-value content from the normal world.

Because the rich operating system cannot be assumed to be free of security-critical bugs, it is necessary to address the challenge of a secure channel to protect the integrity of user input passed to trusted apps.

## 5 Conclusions

We presented our vision and proposal for protecting the presentation of media in highly mobile and interactive systems. Our approach is motivated through a future usage scenario which illustrates the interaction of users with several platforms that seamlessly distribute high-fidelity media. We have reviewed the state-of-the art of TrustZone-enabled systems and proposed to leverage it to establish secure media paths.

For the future we would like to encourage the community to work together to reach the manifestation of this vision.

**Acknowledgments.** This paper presents an idea and approach that was contemplated together with Roderick Bloem and Christian Schwarz. This work was supported by the EC, through project FP7-ICT-STANCE, grant agreement number 317753, and project DALIA of the AAL joint programme.

## References

1. ARM Limited: TrustZone API Specification v2.0 (June 2006), pRD29-USGC-000089
2. ARM Limited: ARM Security Technology Building a Secure System using TrustZone Technology. `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf` (2009), pRD29-GENC-009492C

3. Ekberg, J.E., Asokan, N., Kostiainen, K., Rantala, A.: *Scheduling execution of credentials in constrained secure environments*. In: Proceedings of the 3rd ACM workshop on Scalable trusted computing. pp. 61–70. STC '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1456455.1456465
4. Ekberg, J.E., Bugiel, S.: *Trust in a small package: minimized MRTM software implementation for mobile secure environments*. In: Proceedings of the 2009 ACM workshop on Scalable trusted computing. pp. 9–18. STC '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1655108.1655111
5. GlobalPlatform: TEE Client API Specification v1.0. http://www.globalplatform.org/specificationsdevice.asp (July 2011)
6. GlobalPlatform: TEE Internal API Specification v1.0. http://www.globalplatform.org/specificationsdevice.asp (December 2011)
7. Hussin, W.H.W., Coulton, P., Edwards, R.: *Mobile Ticketing System Employing TrustZone Technology*. In: Proceedings of the International Conference on Mobile Business. pp. 651–654. IEEE Computer Society, Washington, DC, USA (2005), http://dl.acm.org/citation.cfm?id=1084013.1084282
8. Hussin, W.H.W., Edwards, R., Coulton, P.: *E-Pass Using DRM in Symbian v8 OS and TrustZone: Securing Vital Data on Mobile Devices*. Mobile Business, International Conference on 0, 14 (2006)
9. Kostiainen, K., Ekberg, J.E., Asokan, N., Rantala, A.: *On-board credentials with open provisioning*. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. pp. 104–115. ASIACCS '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1533057.1533074
10. Pirker, M., Slamanig, D.: A Framework for Privacy-Preserving Mobile Payment on Security Enhanced ARM TrustZone Platforms. In: Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 1155–1160. TRUSTCOM '12, IEEE Computer Society, Washington, DC, USA (2012)
11. Pirker, M., Slamanig, D., Winter, J.: Practical Privacy Preserving Cloud Resource-Payment for Constrained Clients. In: PETS 2012. LNCS, vol. 7384, pp. 201–220. Springer Verlag (2012)
12. Santos, N., Raj, H., Saroiu, S., Wolman, A.: *Trusted Language Runtime (TLR): Enabling Trusted Applications on Smartphones* (2011)
13. Toegl, R., Hutter, M.: An approach to introducing locality in remote attestation using near field communications. The Journal of Supercomputing 55(2), 207–227 (2011), http://dx.doi.org/10.1007/s11227-010-0407-1
14. Trusted Computing Group: *TCG TPM Specification Version 1.2 rev 113* (2011), https://www.trustedcomputinggroup.org/developers/
15. Wachsmann, C., Chen, L., Dietrich, K., Löhr, H., Sadeghi, A.R., Winter, J.: *Lightweight Anonymous Authentication with TLS and DAA for Embedded Mobile Devices*. In: Burmester, M., Tsudik, G., Magliveras, S., Ilic, I. (eds.) Information Security, Lecture Notes in Computer Science, vol. 6531, pp. 84–98. Springer Berlin / Heidelberg (2011), 10.1007/978-3-642-18178-8_8
16. Winter, J.: Trusted computing building blocks for embedded linux-based arm trustzone platforms. In: Proceedings of the 3rd ACM workshop on Scalable trusted computing. pp. 21–30. ACM, Alexandria, Virginia, USA (2008)
17. Yussoff, Y.M., Hashim, H.: *Trusted Wireless Sensor Node Platform*. In: Ao, S.I., Gelman, L., Hukins, D.W., Hunter, A., Korsunsky, A.M. (eds.) Proceedings of the World Congress on Engineering 2010 Vol I, WCE '10, June 30 - July 2, 2010, London, U.K. pp. 774–779. Lecture Notes in Engineering and Computer Science, International Association of Engineers, Newswood Limited (2010)

# Threat Model of a Scenario Based on Trusted Platform Module 2.0 Specification

Jiun Yi Yap and Allan Tomlinson

Information Security Group
Royal Holloway, University of London
Egham, Surrey
TW20 0EX, United Kingdom
Jiun.Yap.2012@live.rhul.ac.uk, Allan.Tomlinson@rhul.ac.uk

**Abstract.** The Trusted Platform Module (TPM) is a device that can be used to enhance the security of web applications. However, the TPM has to be used in a proper manner in order to benefit from its security properties. A threat model will contribute towards developing a better understanding of how to use the TPM and serve as a reference for future work. In this paper, a web application scenario based on the TPM 2.0 specification is developed and the threat model is constructed using Microsoft's security development lifecycle threat modelling tool. The threats to each element in the model are analysed and the appropriate mitigations are worked out.

**Keywords.** Trusted Platform Module 2.0, Threat Modeling, Web Application, Secure Hardware.

## 1 Introduction

Protection offered by hardware security mechanisms, such as the TPM, can significantly strengthen the security of a web application. This is because the TPM provides assurance of the trustworthiness of the computing platform and offers security functions that build upon the established trust.

Several papers have been presented in the past discussing attacks on TPM 1.2 specification [1-4]. These works focused on examining TPM protocols, identifying weakness, and suggesting solutions to the problems. However, there is a need to provide easier to understand information to people who wish to use TPM technology. Threat modelling can be conducted on use scenarios based on TPM as the process helps to develop a better understanding of this technology. In addition, the results from threat

analysis and mitigations highlight potential security issues to be considered when conducting further research into the applications of TPM.

In this paper, a scenario based on the TPM 2.0 specification is crafted. Microsoft's security development life cycle threat modelling tool is then used to develop the threat model for this scenario [5]. The threats identified are analysed and the appropriate mitigations are worked out.

**Paper Overview** Section 2 gives a brief overview of the TPM and Section 3 explains the threat modelling methodology. In Section 4, we describe the scenario for the threat model and it is followed by threat identification and mitigations in Section 5. Section 6 concludes the paper.

## 2 Brief Overview of TPM

The TPM specification is developed by the Trusted Computing Group (TCG). Some software such as Microsoft's BitLocker uses the TPM to enhance its protection against cyber threats. On the other hand, there are Intel and AMD CPU architecture enhancements that leverage on the TPM to provide security functions for trusted computing. TPM 2.0 is the latest specification from TCG and it replaces the previous TPM 1.2 specification. The most recent revision to TPM 2.0 was published in March 2013 [6].

The three roots of trust, roots of trust for measurement, storage and reporting, provide the minimum functionality required to describe the attributes that contribute towards a platform's trustworthiness. The TPM aims to provide these three roots of trust. In most TPM implementation for the personal computer, the device is attached to the computer motherboard and exchanges data with the rest of the computer components through the Low Pin Count (LPC) data bus.

The key components of TPM 2.0 are shielded storage location, protected program instructions, cryptographic engines and random number generator. A Trusted Computing Base (TCB) can be a BIOS or OS that has proved to be secure and hence can be trusted. When a TCB works together with a TPM 2.0 device, they can offer the capabilities of integrity measurement and reporting, protected data storage location, certification and attestation and authentication.

The changes and enhancements to TPM 2.0 compared to the existing TPM 1.2 include: support for additional cryptographic algorithms, enhancements to the availability of the TPM to applications, enhanced authorisation mechanisms, simplified TPM management and additional capabilities to enhance the security of platform services.

## 3 Threat Modelling

Besides Microsoft's secure development life cycle threat modelling tool, there are an array of threat modelling frameworks and tools, such as OCTAVE from Carnegie Mellon University's Software Engineering Institute [7] and the open source TRIKE [8]. The Open Web Application Security Project (OSWAP) recommends Microsoft threat modelling process [9] and hence the Microsoft tool is chosen to be used in this work. At the beginning of the threat modelling process, the tool resolves the target scenario using a Data Flow Diagram (DFD). A DFD will show all the elements involved in that scenario. An element can be an external entity, a process, a data store or a data flow. A boundary that represents the separation between system components or privilege level will then be defined. This is followed by applying the STRIDE model to identify threat categories for every element in the DFD. STRIDE stands for ***S***poofing, ***T***ampering, ***R***epudiation, ***I***nformation disclosure, ***D***enial of service and ***E***levation of privilege. Only certain threat categories can apply to certain elements [10], see table 1.

| Element Type | Threat Types | | | | | |
|---|---|---|---|---|---|---|
| | ***S*** | ***T*** | ***R*** | ***I*** | ***D*** | ***E*** |
| External Entity | X | | X | | | |
| Process | X | X | X | X | X | X |
| Data Storage | | X | X | X | X | |
| Data Flow | | X | X | | X | |

**Table. 1. STRIDE-per-element matrix from [10]**

The tool will automatically generate the threat categories for each element based on table 1 but each threat category has to be analysed manually. The tool guides the identification of specific threats by providing a set of questions. For every identified threat, an appropriate mitigation should be worked out. Before the threat model report can be generated, additional information on assumptions, external dependencies and security notes can be entered into the tool. It is important to note that the threat model report is a live document and it should be constantly updated whenever a new threat is detected or there is a configuration change to the target scenario.

## 4 Description of Scenario

In this simplified scenario, TPM 2.0 is used to encrypt the cryptographic key used for encrypting data for sharing with a group. This allows the key to be securely exchanged. This scenario is selected because it uses TPM's shielded storage feature and is applicable to a web application situation where certain sensitive web data has to be securely shared with other user over a computer network. The scenario illustrated in figure 1 describes how a symmetric key used for encrypting data is shared using TPM's key duplication function. References to TPM commands from chapter 3 of

TPM 2.0 specification are made at key points of this process. It is noted that TPM 2.0 commands are different from TPM 1.2.

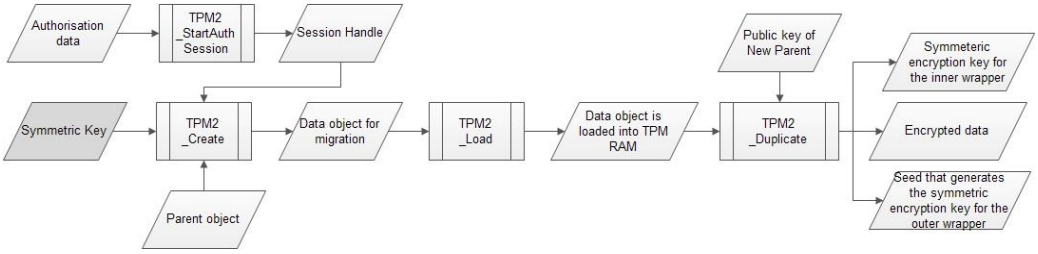


**Fig. 1. To encrypt symmetric key for group share**

In figure 1, TPM2_Create is used to package the key into a TPM object. But before the command can be executed, an authorisation session for the use of a parent object to create the child TPM object has to be started. Upon successful authorisation, TPM2_Create command will execute and produce a data object that contains the key. This data object will have a flag setting indicating that it can be duplicated. In addition, a user can specify an authorisation policy to control access to this data object. The next step is to load this data object into the TPM RAM using the command TPM2_Load. This command will return a handle to the key object. The final command to run is TPM2_Duplicate whereby this data object is repackaged and encrypted. The output from TPM2_Duplicate is the encrypted duplicated object, the symmetric encryption key used to encrypt the inner wrapper and a seed that generates the symmetric encryption key for the outer wrapper. The confidentiality of the seed value is protected by a public key provided by the destination TPM. These outputs can be transferred to the destination TPM using mechanism that protects the confidentiality and integrity of the duplicated object and check the authenticity and authorisation of the destination TPM.

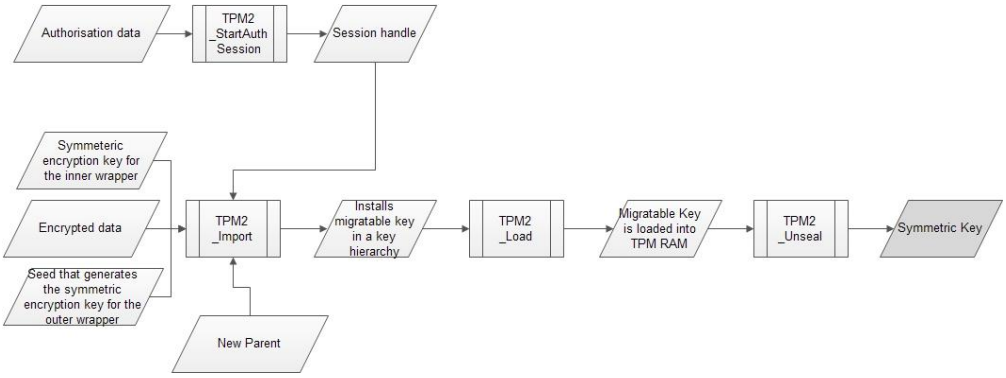


**Fig. 2. To recover symmetric key for group share**

At the destination TPM, the reverse is carried out. Referring to figure 2, TPM2_Import is used to transfer the duplicated object into the destination TPM. An authorisation session for the use of the new parent object is started. Upon successful authorisation, the command will execute and the duplicated object is decrypted. To protect the confidentiality of the key object, it is encrypted with an encryption key derived from the new parent. This key object is then loaded into the TPM RAM using the command TPM2_Load. A handle to the loaded key object is returned to the user. To obtain the symmetric key, the authorisation data and key object handle are provided to the command TPM2_Unseal. When this command executes successfully, the symmetric key is presented to the user.

## 5 Threats Identification and Mitigation

Using Microsoft‘s secure development lifecycle threat modelling tool, two DFDs were drawn to represent the scenario of encrypting and decrypting the symmetric key for group share. The DFDs are shown in figure 3 :

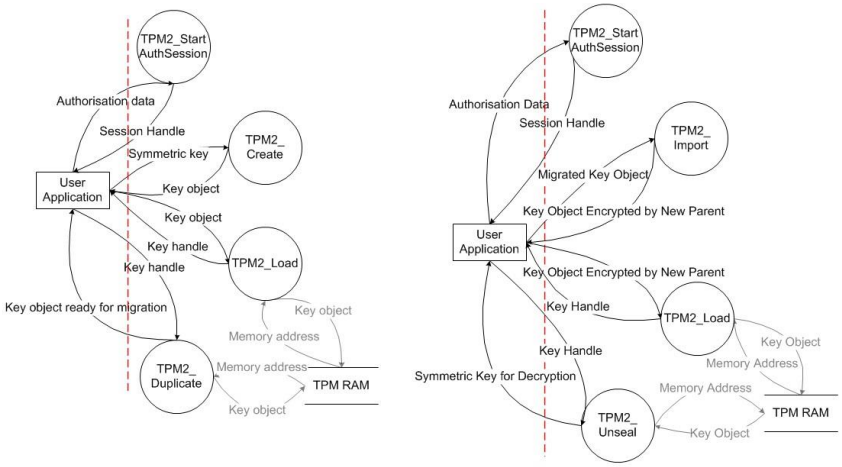


**Fig. 3.** DFD for encrypting symmetric key (left) and for recovering symmetric key (right).

The tool analyzed the two DFDs individually and threat categories for every element were generated. A total of 101 potential threats were identified for the process of encrypting the symmetric key for group share while a total of 96 potential threats were identified for the decrypting process. The data flow between the processes and TPM RAM are not accessible externally and hence they were not analyzed (grey coloured lines). Appropriate mitigations for all the identified threats were worked out. TPM 1.2 attacks [1-4] identified in earlier studies could not be applied directly to this threat model as the protocols and commands for TPM 2.0 have been changed.

In this paper, it is impossible to present all the threats and mitigations for this scenario but some of the more critical ones will be discussed in table 2.

| S/N | Element | Type | Description | Mitigation |
|---|---|---|---|---|
| 1 | TPM2_Import | ***S*** | Attacker attempts to load a duplicated key object that is not generated by a TPM. | The source TPM can insert an unique identifying value into the key object when when using TPM2_Create. The destination TPM will verify the authencity of the key object by inspecting this identifying value. |
| 2 | TPM2_StartAuthSession | ***I*** | The cryptographic protection for the authorized sessions can be weakened if the nonce and salt value used in the generation of the session key have low entropy. | The method used by the software application to generate the nonce and salt value has to meet security requirements, for example NIST SP 800-90A. An alternate method is to use TPM's random number generator (RNG) to provide these values. However, TPM's RNG has to meet security requirements as well. |
| 3 | Key object (TPM2_Create to User Application) | ***I*** | The sensitive part of the key object is symmetrically encrypted using a key derived from the parent object. A random value is included in the process as an initialization vector (IV). When an object is created for duplication, the IV is set to zero. The key objects can be susceptible to cryptographic analysis if the parent object is reused multiple times. | The user application has to avoid reusing the parent object multiple times when creating an object for duplication. |
| 4 | TPM2_Create | ***R*** | User denies executing this command. | TPM will have to rely on the TCB to keep a log of the commands performed on TPM. The availability of a log is crucial to forensic investigation in the event of a security incident. An example of a guideline for the security management of the log will be NIST SP 800-92. |

**Table 2. Threat descriptions and mitigations**

Since TPM's design objectives do not include protection from physical attacks, this paper will not dwell on this threat but a user should be aware of the types of physical attack [11,12] and take appropriate mitigations.

## 6 Conclusion

In this paper, the threat modelling process is used to develop a better grasp of TPM technology and its application. A scenario on using TPM to share a symmetric cryptographic key is crafted and the threat model is produced. Although the scenario is simple, the amount of threats and the required mitigations are substantial. Hence, it is beneficial that TPM users conduct threat modelling on their use scenarios. Meanwhile, this work highlights some potential pitfalls that should be considered when conducting further research into the applications of TPM.

**Acknowledgements.** We would like to thank Graeme Proudler and Liqun Chen from HP Labs, UK for their advice on TPM 2.0 specification.

**References.**

1. Liqun, C and Mark, R.: Offline Dictionary Attack on TCG TPM Weak Authorisation Data, and Solution. In: David, G., Helmut, R., Ahmad-Reza, S., and Claire, V. (eds.) Future of Trust in Computing. Vieweg & Teubner (2008)
2. Liqun, C. and Mark, R.: Attack, Solution and Verification for Shared Authorisation Data in TCG TPM. In: Pierpaolo, D. and Joshua D, G. (eds.) FAST 2009. LNCS, vol. 5983, pp. 201-216. Springer, Heidelberg (2010)
3. Danilo, B., Lorenzo, C., Andrea, L. and Mattia, M.: Replay Attack in TCG Specification and Solution. In: ACSAC 2005, pp. 127-137. IEEE Computer Society (2005)
4. Sigrid, G., Carsten, R., Dirk, S., Marion, A. and Rainer, P.: Security Evaluation of Scenarios Based on the TCG's TPM Specification. In: Joachim, B. and Javier, L. (eds) ESORICS 2007. LNCS, vol. 4734, pp. 438-453. Springer, Heidelberg (2007)
5. Microsoft Secure Development Life Threat Modelling Tool, http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx
6. Trusted Computing Group.: Trusted Platform Module Library Family "2.0" Level 00 Revision 00.96. 15 March 2013
7. OCTAVE Threat Modelling Tool, http://www.cert.org/octave/
8. TRIKE Threat Modelling Tool, http://www.octotrike.org/
9. The Open Web Application Security Project Threat Risk Modelling, https://www.owasp.org/index.php/Threat_Risk_Modeling
10. Shawn, H., Scott, L., Tomasz, O. and Adam, S.: Uncover Security Design Flaws Using the STRIDE Approach. MSDN Magazine, November 2006
11. Christopher, T.: Semiconductor Security Awareness, Today & Yesterday. Black Hat DC 2010
12. Bryan, P.: Bootstrapping Trust in a "Trusted" Platform. In: HOTSEC 2009, Art. 9. USENIX Association (2008)