Firewall Mechanism in a User Centric Smart Card Ownership Model

Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes

Information Security Group Smart card Centre, Royal Holloway, University of London
Egham, Surrey, United Kingdom
{R.N.Akram, K.Markantonakis, Keith.Mayes}@rhul.ac.uk

Abstract. Multi-application smart card technology facilitates applications to securely share their data and functionality. The security enforcement and assurance in application sharing is provided by the smart card firewall. The firewall mechanism is well defined and studied in the Issuer Centric Smart Card Ownership Model (ICOM), in which a smart card is under total control of its issuer. However, it is not analysed in the User Centric Smart Card Ownership Model (UCOM) that delegates the smart card control to their users. In this paper, we present UCOM's security requirements for the firewall mechanism and propose a generic framework that satisfies them.

1 Introduction

The multi-application smart card initiative [1] ensures a secure and flexible execution environment for multiple applications from same or different organisations [2,3]. It facilitates the co-existence of interrelated and cooperative applications that augment each other's functionality. This enables applications to share their data as well as functionality with other applications, introducing a major security concern of unauthorised inter-application communication. The solution to this problem has been the smart card firewall.

The firewall acts as a supervisory authority on a smart card, monitoring inter-application communications [4]. The main aim is to ensure security and reliability of application sharing mechanisms even in adverse conditions such as caused by a malicious application, a developer's mistake or design oversight [5]. The firewall deployed in the Issuer Centric Smart Card Ownership (ICOM) is well defined [5–9] and studied [10–13]. However, this is not the case for the firewall mechanism in the User Centric Smart Card Ownership Model (UCOM) [14], and it is the focus of this paper.

The widely adopted smart card based business model is the ICOM [2,14,15]. In this model, smart cards are under total control of the issuing organisation, referred to as the Card Issuer. Smart cards issued by a Card Issuer can host multiple applications and if required these can be from different organisations. Organisations that provide applications, but do not issue cards are referred to as Application Providers (or Service Providers) and they are reliant on establishing

a business and trust relationship with Card Issuers. Card Issuers and Application Providers also establish the necessary trust and assurance that the application will not harm the card platform and vice versa. Such an explicit business and trust relationship does not exist in the UCOM.

The UCOM gives the choice of applications to the users and they can request to have any application on their cards. The request is sent to the corresponding Service Provider (SP) in the UCOM. If the security assurance provided by the smart card along with its services and user credentials are valid then the SP leases its application(s) under certain terms and condition stipulated by the SP [14]. Leased application(s) are controlled only by their respective SPs and so this introduces unique issues regarding inter-application communications. In this paper, we will analyse the functional nature of the UCOM and its effects on the firewall mechanism and propose a framework that is suitable for secure operation.

In section two, we discuss the firewall mechanism within the multi-application smart card environment and how they are implemented in popular smart card platforms (e.g. Java Card [8] and Multos [9]). Section three describes unique issues presented to the firewall mechanism in the UCOM. In section four, a framework for a smart card firewall is presented that is suitable for the UCOM environment. In section five a case study briefly illustrates how the framework can be implemented, and finally section six provides the concluding remarks.

2 Multi-application Smart Card Platforms

In this section, we describe an application sharing mechanism in multi-application smart card platforms and how it is implement in Java Card and Multos.

2.1 An Application Sharing Mechanism

The most adopted business and operational scenario for the smart card based service model has been the ICOM [15]. For brevity, we will only discuss the application sharing (firewall) mechanism related to the ICOM in this section.

Multi-application smart cards facilitate co-operative schemes enabling optimised memory usage, with scope for data and service sharing between applications [15]. Therefore, a firewall mechanism should ensure application segregation while providing a secure and controlled way to allow applications to communicate data and share functionality. In the ICOM the issuer provides the platform security and reliability assurance, including the application segregation [7] that is necessary to avoid any on-card leakage of secret data. A firewall is basically an access control mechanism that does not protect against information propagation [7] (which is beyond the scope of this paper). In addition to protecting applications;

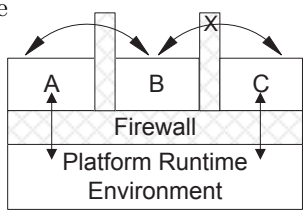


Fig. 1. A Generic Application Sharing Mechanism

the firewall mechanism should also protect the platform by ensuring that applications can only access platform services through a well formed interface that cannot be used to subvert any protection of the platform.

To explain the firewall mechanism refer to simple example illustrated in figure 1. Consider that there are three applications: A, B, and C. The Application Providers of A and B have a trust relationship but Application Provider of C is not fully trusted by them. Application A specifies data and functionality that it wants to share with B, these are termed as shareable resources. The firewall facilitates the sharing with the help of the runtime environment. When B requests access to the resource of A, the firewall verifies the access credentials and if successful it allows the access. However, in the case of a request from the application C, the request will be denied.

The firewall should also segregate the platform runtime environment from the application space. To execute privileged services the application(s) could only make requests to the runtime environment through well formed Application Programming Interfaces (APIs). The firewall should ensure that this communication channel should not become a means to subvert the firewall in order to gain unauthorised access to resources from other applications.

2.2 Firewall Mechanism in Java Card

Java Card [4] is a smart card platform that supports a scaled down version of the popular Java language. The architecture of a Java Card is shown in figure 2.

The Java Card Runtime Environment (JCRE) sits on top of the smart card hardware and manages the on-card resources, applet execution and applet security [8]. The JCRE consists of APIs (e.g. `javacard.framework.APDU`, `Util` and `Shareable`) that an application can use to access JCRE services. The JCRE also has system classes that are integral to its functions and these classes are not visible to applications. Applets reside on top of the JCRE, and they are grouped together into packages.

Each instance of an applet has a unique Application Identifier (AID) [8]. An instantiated representation of an applet is termed an object. Each object is associated with a context, including the JCRE objects (System Context). The Java Card Virtual Machine (JCVM) only allows an object to execute if the current "Active" context is the one from which it belongs. In figure 2, object of Applet$_{B1}$ will only executes if the "Active" context is context B. The firewall restricts all cross context communication except for object sharing mechanisms: JCRE Entry Point Objects and Shareable Interface Objects (SIO). All applets in a package have the same context so there is no firewall between them.

The JCRE Entry Point Objects are instances of the Java Card APIs that can be used by applications to access platform services. These objects are accessible to all applets, and they enable non privileged (applets) applications to execute privileged commands. The JCRE Entry Point Objects are implemented by the Java Card manufacturer who is responsible for their security and reliability.

The SIO enables an application to share its resources with other authorised application(s). To utilise the SIO functionality, an application should extend

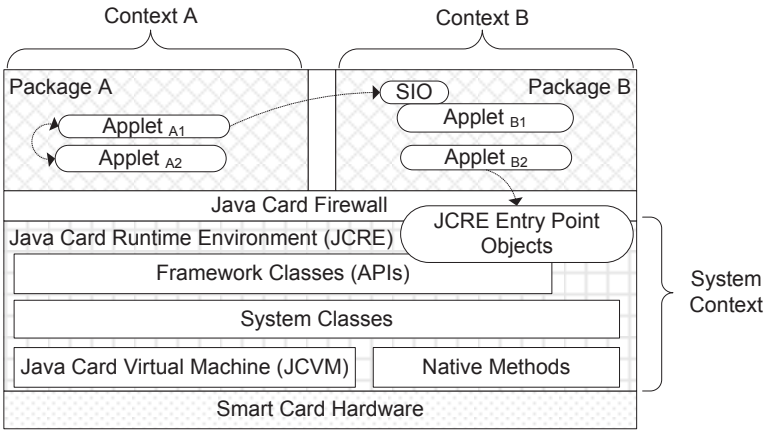


Fig. 2. Java Card Architecture

the shareable interface (`javacard.framework.Shareable`) and the functionality implemented in the extended class will be shareable with other applets.

When an object requests either an SIO or JCRE Entry Point Object, the JCVM saves the current "Active" context and invokes the requested object along with the associated context. Therefore, a shareable object always executes in its own context, enabling it to access any applet from the package it belongs. By taking into account figure 2 when Applet$_{A1}$ calls the SIO of Applet$_{B1}$, the JCVM saves context A and invokes context B along with initiating the execution of the SIO. The SIO object can then call any method in package B. Furthermore, it can also call any JCRE Entry Point Object. When the SIO completes its execution, the JCVM restores the previous context (context A).

2.3 Firewall Mechanism in Multos

Compared to Java Card, Multos [9] takes a different approach to the smart card firewall. The Multos Card Operating System (COS) resides over the smart card hardware as illustrated in figure 3a. The Multos COS administers communication, resource management, and the virtual machine [9]. Applications do not have direct access to the Multos COS services, instead they utilise the Application Abstract Machine that is a set of standard APIs consisting of instructions and built-in functions. These APIs are used by applications to communicate with the COS and request privileged services. The top layer is the application space, and similar to Java Card the application segregation is implemented by the firewall.

In Multos, application delegation is implemented to facilitate application resource sharing. The application that initiates the process is called the delegator and the application that is initiated is called the delegate. The process of delegation works as described below and shown in figure 3b:

1. Application A (delegator) creates an APDU in the public memory and invokes the delegate command. The APDU consists of application B's AID, requested data or function and delegator's AID.
2. The Multos COS initiates the execution of B that looks for the APDU in the public memory. It reads the APDU and processes it.
3. On completion, B creates a response APDU within the public memory.
4. The Multos COS switches back to A that then retrieves B's APDU.

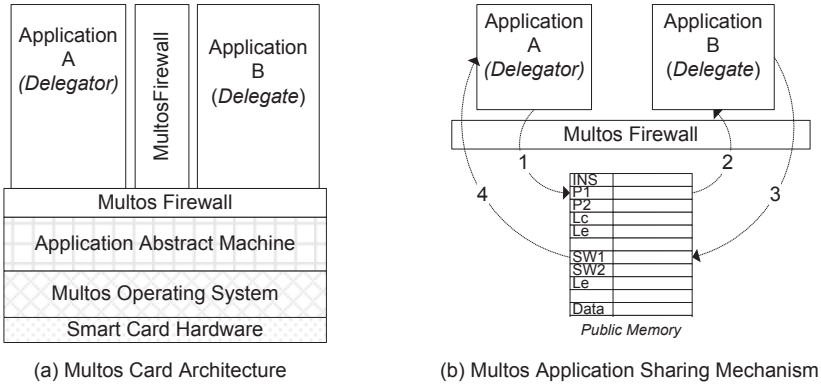


Fig. 3. Multos Card Architecture and Firewall Mechanism

In both Java Card and Multos, additional measures are implemented in conjunction with the firewall mechanism to protect the platform. These measures include byte-code verification (on-card and off-card) [16,17], strict mechanism to install applications [18] and virtual machine based security mechanisms [19,20].

3 User Centric Smart Card Ownership Model

In this section, we discuss the security and operational requirements for a firewall mechanism in the UCOM.

3.1 Application Sharing Requirements

The UCOM is expected to support a dynamic service environment with a wide range of application types. Therefore, the firewall mechanism should also reflect this dynamic nature [14].

Inter-application Communication. The UCOM firewall should facilitate a flexible mechanism that enables a server application[1] to implement a hierarchical

[1] Server application: An application that provides shareable data or functionality to authorised applications.

access level firewall. In such a firewall, a server application assigns shareable resources according to different access levels. A client application[2] is initially assigned an access level although the server application can also revoke, upgrade, or demote the existing privileges of a client application, illustrated by figure 4.

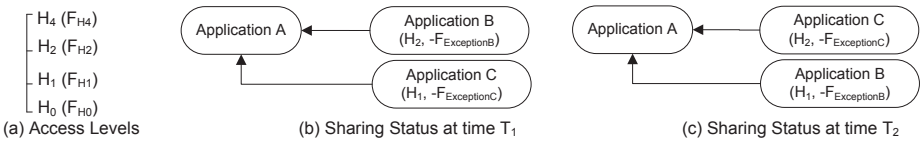


Fig. 4. Hierarchical Access Level Firewall

Consider an application A that offers shareable data and functionality divided into different hierarchical levels. Requesting applications are only authorised to access data or functionality matching assigned level. In figure 4a, there are four hierarchical levels with H0 the lowest and H3 the most privileged level. The data and functionality associated with each level is denoted by the "FLevel". The "-FException" is the negative permission, that lists the data or functionality that is not authorised to an application for the given access privileges. Application A keeps track of access levels along with -FException" associated with each applications. Application B's access privileges (H2, -FExceptionB) will be read as B has access to all data and function associated with level H2 (FH2)and below (FH0 and FH1) with an exception of data or functionality of "-FExceptionB'. B and C have access rights "H2, -FExceptionB" and "H1, -FExceptionC" respectively for A at time T1. At some later time (T2) A modifies the access privileges of B and C, demoting B to H1 and upgrading C to H2. In addition, the firewall mechanism will also allow the modification of the "-FException".

Unlike the present Java Card or Multos firewalls, in the UCOM the sharing permissions will have limited lifetime and on expiry the client application(s) have to renegotiate the access permissions with the server application.

Application Sharing Delegation. A client application can delegate access to a server application (after authorisation) to another application on its behalf.

Consider the following scenario with three applications A, B, and C. There is an application sharing relationships A→B, and B →C; but none between A and C. Let us assume by way of example that application B gives royalty points if the cardholder uses A and these points are redeemable from C. Therefore, usage of A can lead to redeemable points (benefits) from C. At some point in time, the cardholder requests the deletion of application B and it requests the permission from A to delegate its sharing privileges to C. It is at the sole discretion of A's

[2] Client application: An application that requests the shareable resources of a server application. The notation to present this relationship is Server → Client.

SP whether it would allow such an action or not. The SP of A may allow such action completely or impose conditions such as demoting the privileges to the lowest possible level for application C. Therefore from this point of time, C can access A on behalf of the B.

Application-Platform Communication. This requirement deals with bi-directional communication between an application and a smart card platform and it is sub-divided into two sections as listed below.

Application to Platform Communication. Platforms make their services available to applications either through Entry Point Objects [8] or standard APIs [9]. In both cases, applications may have access to more platform services than required that would not be desirable in the UCOM [14]. In the UCOM, applications are only given access to those platform services that are authorised by their SPs. The firewall ensures that an application cannot have access to any other services from the platform for which it is not authorised. This allows the SPs to control their applications' behaviours, especially in terms of on-card and off-card communication.

Platform to Application Communication. Java Card (like other multi-application smart cards) provides global access rights to the platform. The global access rights mean that an object of JCRE System Context can access any method (object) in any of the application contexts. However, the Java Card specification explicitly notes that the platform should only access certain methods (`select`, `process`, `deselect`, or `getShareableInterfaceObject`) from an applet context [8, see section 6.2.3]. In case of the UCOM, the firewall should ensure that a platform cannot have access to methods that are not sanctioned by the application SPs. Furthermore, it should enable an object or method to verify the requesting source. For example if the source is the platform, and it is trying to access an object or method not sanctioned by the corresponding SP, then it should throw a security exception.

Privacy Issues. In the UCOM, cardholders can have diverse applications on their smart cards, and each of these applications may represents their identity in some context. The firewall mechanism should not allow an application to discover the existence of other applications, because such a privilege could be abused to profile a user, perhaps for marketing or fraudulent purposes. In Java Card, `public static AID lookupAID` can be used to list the installed applications. It is not an issue in the ICOM as there is a central authority (card issuer) that has prior knowledge of installed applications and policed their functionality. However, it is a potential privacy threat in the UCOM.

4 Proposed Framework for the UCOM Firewall

In this section, the architecture of the proposed UCOM firewall is described along with explanation of its operations.

4.1 Overall Architecture

The UCOM is a smart card operating system and platform independent framework [14]. However, for brevity, clarity and intuitiveness we consider the Java Card firewall mechanism as the basis of our proposal. To illustrate the UCOM firewall, figure 5 shows a generic architectural view of the UCOM smart card that is principally similar to the Java Card (shown in figure 3).

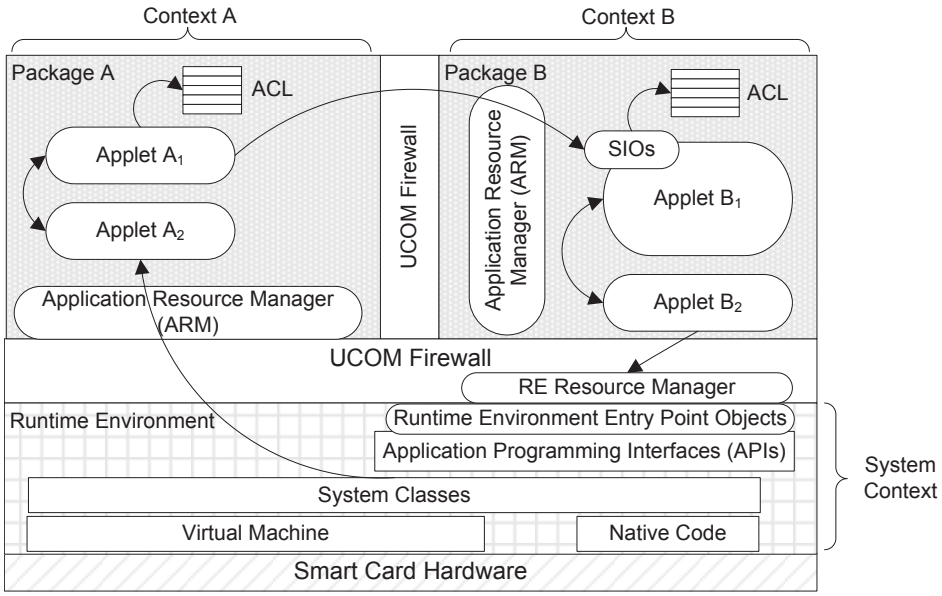


Fig. 5. Generic Architecture of User Centric Smart Card Firewall Mechanism

The Runtime Environment (RE) Resource Manager controls the access to the RE Entry Point Objects that are used to access platform services. The resource manager will enforce the security policy for applications as defined by the SPs, limiting the access to the platform resources as stipulated by the policy.

For each application (package), an Application Resource Manager (ARM) is introduced. This component will act as the authentication and resource allocation point. A client application will request a server application's ARM for shareable resources. The ARM will decide whether to grant the request based upon the client's credentials (associated privileges). At the time of application installation, the ARM also establishes a shareable interface connection with the platform, enabling it to access methods that are essential for the application execution. The platform can access any method in the application context only after authorisation from the application's SP. The ARM also receives information regarding the requesting application. If the request is from the system context for

a method that is not allowed to be accessed by the platform, then the ARM will throw a security exception.

An Access Control List (ACL) is a private list and it is used to facilitate the implementation of the hierarchical access mechanism. It can be update remotely by the corresponding SP via the ARM, enabling the SP to change the behaviour of its application's sharing mechanism. The ACL holds the lists of granted permissions, received permissions (permissions to access other application's resources) and a cryptographic certificate revocation list of client applications. The structure of an ACL is under the sole discretion of its SP.

The operations of the UCOM firewall can be sub-divided into two distinctive phases. In phase one, a binding is established between the client and server applications. This process includes authentication of the client's credentials and access privileges by the server's ARM. In the second phase, the client application requests resources in line with the privileges sanctioned by the ARM.

To have a consistent view of the sharing mechanism over diverse application scenarios, the description of the application binding and resource request process are deliberately defined in high-level representations. The fine details of these processes are left to the individual preferences of the SPs. The UCOM firewall mechanism supports these operations but does not define the minute details.

4.2 Application Binding

This process deals with the first request by a client application for shareable resource(s) of a server application (phase one). Upon receiving the request, the server application first ascertains that the requesting application is the authorised application as it claims. After authentication, both applications establish a cryptographic binding that is used in all future requests.

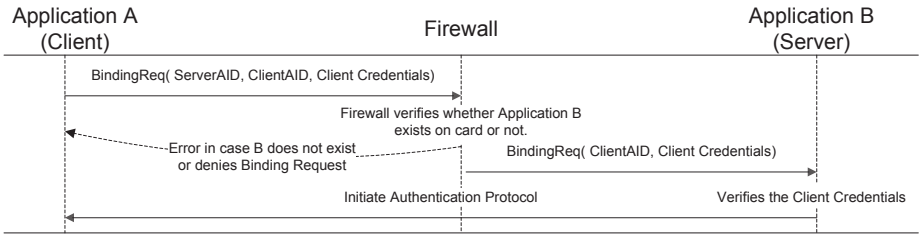


Fig. 6. Illustration of Application Binding Request Process

The process is illustrated in figure 6 and explained as below. Application A (client) sends a binding request message. This message consists of application B's (server) Application Identifier (AID), along with A's AID and credentials. The nature of the credentials can be at the sole discretion of the server application. However, to explain the process we use cryptographic certificates [21].

The SP of the server application, issues a cryptographic certificate to the client application's SP who in return issues individual (unique) certificates for its applications, certifying the unique public key pair of each client application. As the root authority (Certification Authority [21]) is the SP of a server application, any instance of the server application will be able to verify and accept it. On receiving the binding request the firewall mechanism looks up for the ServerAID to verify whether the application exists on the card or not. If it exists, the request would be forwarded to the corresponding ARM. Conversely, if the application does not exist, or server turns down the binding request, the firewall mechanism would throw an exception that would be same in both cases, to avoid a malicious application from potentially discovering the existence of an application on a card.

If the firewall forwards the request to the server application (Application B), it verifies the requesting application's credentials by initiating an authentication protocol. The outcome of the authentication protocol is generation and verification of a cryptographic (symmetric) binding key [22]. The client application will use this key in all future resource requests and in any related operation discussed in the subsequent sections. SPs should ensures that their authentication protocol is secure against application impersonation [22], and replay attacks [21].

4.3 Requesting Resource

A client application can request the server application's shareable resource as it required (subject to valid access permissions) as illustrated by figure 7.

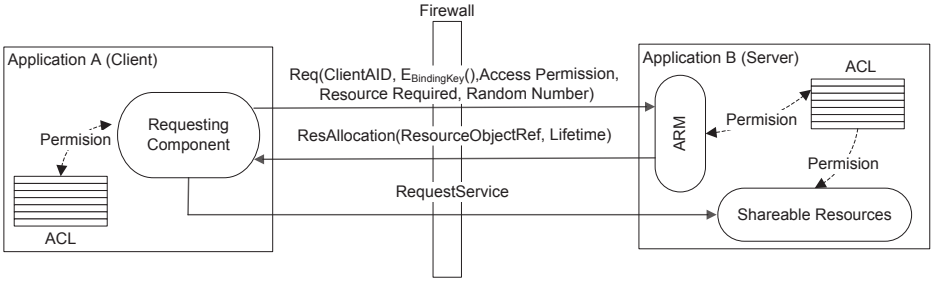


Fig. 7. Application Shareable Resource Access Request Process

The request message sent to the corresponding ARM consists of a ClientAID, an authenticator (message encrypted with binding key), access permission, required resource and a random number to provide freshness [21]. By verifying the authenticator, the ARM ascertains the origin of the message, i.e. the client application. Subsequently it checks the access permission for the client application (from the server application's ACL). If the client application is authorised

to access the requested resource, the ARM would return the resource's object reference along with the sharing lifetime.

As described in section 3.1, the client application may have negative permission. To implement negative permission control each of the data or methods of the shareable resource is tagged with a unique ID. When the client application accesses a method from a shareable resource object, the unique ID of the method is compared with the negative permissions. If there is a match the method returns with an exception.

4.4 Privilege Modification

The SP of a server application can modify the privileges of a client application by updating the ACLs. The ARM of the server application verifies the initiator's (SP's) identity and credentials, before it allowing the update of the ACL(s). The implementation of the privilege modification is at the sole discretion of the SP. However, such an update could be similar to application update mechanism already deployed, notably Over-The-Air updates in (U)SIM application [23].

4.5 Application Sharing Delegation

This functionality of the UCOM firewall is subject to the sharing terms and conditions between the relevant SPs, which will grant or deny requests as appropriate.

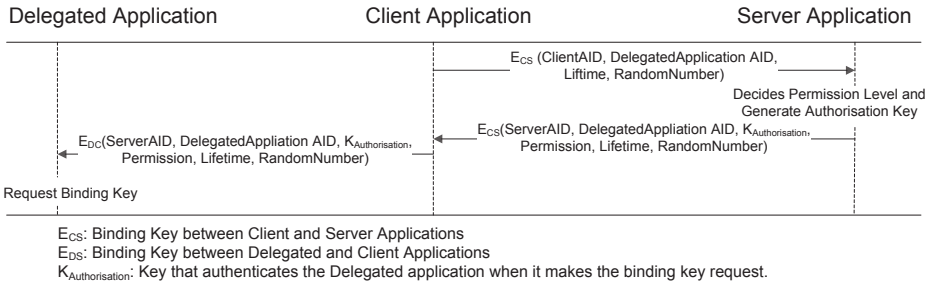


Fig. 8. Application Sharing Devolvement Dialogue

The privilege level of an application (delegated application) to which the client application delegates the resource-sharing does not have to be the same as itself. The privilege level of the delegated application is at the sole discretion of the server application's SP. The steps involved in the process of resource sharing delegation are listed below.

1. A client application requests a server application to delegate its resource-sharing privilege to another application.

2. According to the server application's policy, it can either keep the same level of privileges as the client application or demote the privileges for the delegated application. The server application generates a message encrypted by the binding key (Server→Client binding key) and sends it to the client application. The message contains Server AID, DelegatedApplication AID, Access permissions, Delegation lifetime and Delegation Request Key.
3. The client application decrypts the message and re-encrypts it with the Client→Delegated binding key and sends it to delegated application.
4. The delegated application uses it to authenticate itself to the server application and establishes a binding (section 4.2).

Once the delegation is completed, the client application cannot have access to the shareable resources, unless it requests the resource delegation to be terminated. The termination process is similar to the delegation process. Therefore, only one application (either client or delegated application) can access the shareable resources. The firewall mechanism ensures that once the resource delegation is terminated, the delegated application cannot have access to the resources.

4.6 Application-Platform Communication

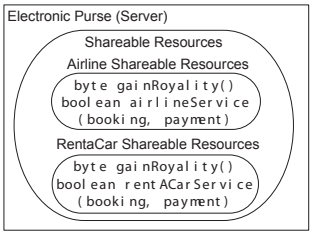
At the time of installation, an application establishes bidirectional resource sharing with the platform. The application can access those platform APIs that are stipulated in the SP's application lease policy [14] and the platform obtains the shared resources of the application that are necessary to initiate the application execution. The platform security context does not have global access in UCOM based smart cards. This is to avoid any possible exploitation of the platform that could lead to the information leakage (data or code) from an application. The resource-sharing delegation is disabled in the platform-application communication and the firewall would deny such requests to avoid any illegal access to the APIs by an application through resource sharing delegation.

5 Case Study

In this section, a UCOM case study is discussed of an electronic purse application with special functionality. The described implementation is simply to illustrate the firewall mechanism.

5.1 Overall Scenario

In this scenario there are three applications, Electronic Purse, ABC Airline and XYZ Rentacar. The electronic purse application has a trust relationship with the other two applications but with different privilege levels. Whenever the cardholder uses the Electronic Purse application, royalty points can be earned for the airline application that can either be redeemed from the airline or from rent a car service. For brevity, the details are brief focusing only on the firewall mechanism.



(a) Electronic Purse Shareable Resouces

Client Application	Hierarchical Level	Negative Permission	Delegated
ABC Airline	H_1	Non	false
XYZ Rentacar	H_0	Non	false
.....	...	....	....

Hierarchical Levels:
H1: Access allowed to Airline and RentaCar Shareable Resources
H2: Access allowed to only RentaCar Shareable Resources

(b) ACL Implementation

Fig. 9. Electronic Purse Application Implementation

The electronic purse application implements the shareable resources as illustrated by figure 9a. These resources have unique identifiers that are used to implement negative permission. The identifier is in the form of a byte value. For example, the `byte gainRoyality()` of the airline shareable resources, has the identifier "0x0001" represented by the `private static byte gainRoyalityID`. To enforce the negative permission, method identifiers are listed in the ACL that a method should check when it receives a request from the client applications.

5.2 Implementation Examples

In this section, we will describe the details of the SP's dependent components of the UCOM based firewall mechanism, which are listed below:

Authentication Protocol. The protocol [24] is based on two steps. In the first step the protocol initiates the mutual authentication, and at the second step a symmetric key is mutually generated and shared.

Authenticator. It is an encrypted message that verifies the identity of a client application. The authenticator for the airline application is $\mathsf{E}_{\mathsf{BindingKeyABC}}$(ABC-Identity | ResourceRequested | Random Number | Lifetime). The electronic purse application also calculates the authenticator, and if the results are the same then the ABC Airline request would be authenticated.

Application Sharing Delegation. The ABC airline application requests the resource sharing delegation. The electronic purse application only allows the delegated application to access the `gainRoyality()`. The resource sharing delegation process will upgrade the XYZ Rentacar application's privileges to H_1 with negative permission for `private static byte airlineServicesID`.

This case study shows a simplistic view of an implementation of those firewall components that are left to a SP's discretion. The proposed framework provides a supporting platform that enables individual SPs to either implement their

proprietary or well studied public algorithm to protect their shareable resources. This enables them to implement the crucial element of the firewall, and remove any possible ambiguity in different implementations (by card manufacturers).

6 Conclusion

In this paper, we discussed popular smart card based firewall mechanisms and how they work. Then we described the unique security requirements of the UCOM and presented an appropriate firewall mechanism extended from the Java Card firewall. During the research, the Multos based firewall mechanism was considered unsuitable for the open and dynamic environment that UCOM aims to support, because the security of the Multos firewall is reliant on the stringent application installation mechanism. In addition to implementing the traditional firewall controls, we also presented functionality that is lacking in the present popular firewall mechanism, but we consider them to be useful for the UCOM proposal. Future research directions will focus on implementation to test performance and practical feasibility of such proposals.

References

1. D. Deville, A. Galland, G. Grimaud, and S. Jean, "Smart card operating systems: Past, present and future," in *In Proceedings of the 5 th NORDU/USENIX Conference*, 2003.
2. D. Sauveron, "Multiapplication Smart Card: Towards an Open Smart Card?" *Inf. Secur. Tech. Rep.*, vol. 14, no. 2, pp. 70–78, 2009.
3. S. Chaumette and D. Sauveron, "New Security Problems Raised by Open Multi-application Smart Cards. ." *LaBRI, Université Bordeaux 1.*, pp. 1332–04, 2004.
4. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.
5. M. Montgomery and K. Krishna, "Secure Object Sharing in Java Card," in *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology*. Berkeley, CA, USA: USENIX Association, 1999, pp. 14–14.
6. M. Éluard, T. P. Jensen, and E. Denney, "An Operational Semantics of the Java Card Firewall," in *E-SMART '01: Proceedings of the International Conference on Research in Smart Cards*. London, UK: Springer, 2001, pp. 95–110.
7. C. Bernardeschi and L. Martini, "Enforcement of Applet Boundaries in Java Card Systems," in *IASTED Conf. on Software Engineering and Applications*, 2004, pp. 96–101.
8. *Java Card Platform Specification; Application Programming Interface, Runtime Environment Specification, Virtual Machine Specification*, Sun Microsystem Inc Std. Version 2.2.2, March 2006. [Online]. Available: http://java.sun.com/javacard/specs.html
9. *Multos: The Multos Specification, http://www.multos.com/*, Online, Std.
10. M. Huisman, D. Gurov, C. Sprenger, and G. Chugunov, "Checking Absence of Illicit Applet Interactions: A Case Study," in *Fundamental Approaches to Software Engineering, FASE 2004*. Springer, 2004.

11. W. Mostowski and E. Poll, "Malicious Code on Java Card Smartcards: Attacks and Countermeasures," in *CARDIS '08: Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications*. Berlin, Heidelberg: Springer, 2008, pp. 1–16.
12. M. Éluard and T. Jensen, "Secure Object Flow Analysis for Java Card," in *CARDIS'02: Proceedings of the 5th conference on Smart Card Research and Advanced Application Conference*. Berkeley, CA, USA: USENIX Association, 2002, pp. 11–11.
13. P. Bieber, J. Cazin, A. E. Marouani, P. Girard, J. L. Lanet, V. Wiels, and G. Zanon, "The PACAP Prototype: A Tool for Detecting Java Card Illegal Flow," in *JavaCard '00: Revised Papers from the First International Workshop on Java on Smart Cards: Programming and Security*. London, UK: Springer, 2001, pp. 25–37.
14. R. N. Akram, K. Markantonakis, and K. Mayes, "Application Management Framework in User Centric Smart Card Ownership Model," in *The 10th International Workshop on Information Security Applications (WISA09)*, H. Y. YOUM and M. Yung, Eds., vol. 5932/2009. Springer, August 2009, pp. 20–35.
15. P. Girard, "Which Security Policy for Multiplication Smart Cards?" in *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology*. Berkeley, CA, USA: USENIX Association, 1999, pp. 3–3.
16. D. A. Basin, S. Friedrich, J. Posegga, and H. Vogt, "Java Bytecode Verification by Model Checking," in *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*. London, UK: Springer, 1999, pp. 491–494.
17. D. A. Basin, S. Friedrich, and M. Gawkowski, "Verified Bytecode Model Checkers," in *TPHOLs '02: Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics*. London, UK: Springer, 2002, pp. 47–66.
18. C. Colby, P. Lee, G. C. Necula, F. Blau, M. Plesko, and K. Cline, "A Certifying Compiler for Java," in *PLDI '00: Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*. New York, NY, USA: ACM, 2000, pp. 95–107.
19. G. Barthe, G. Dufay, L. Jakubiec, and S. Melo de Sousa, "A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines," in *VMCAI '02: Revised Papers from the Third International Workshop on Verification, Model Checking, and Abstract Interpretation*. London, UK: Springer, 2002, pp. 32–45.
20. E. Börger and W. Schulte, "Defining the Java Virtual Machine as Platform for Provably Correct Java Compilation," in *MFCS '98: Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*. London, UK: Springer, 1998, pp. 17–35.
21. B. Schneier, *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995.
22. D. Deville and G. Grimaud, "Building an "impossible" verifier on a java card," in *WIESS'02: Proceedings of the 2nd conference on Industrial Experiences with Systems Software*. Berkeley, CA, USA: USENIX Association, 2002, pp. 2–2.
23. K. Mayes and K. Markantonakis, Eds., *Smart Cards, Tokens, Security and Applications*. Springer, 2008.
24. K. Markantonakis and K. Mayes, "A Secure Channel protocol for multi-application smart cards based on public key cryptography," in *CMS 2004 - Eight IFIP TC-6-11 Conference on Communications and Multimedia Security*, D. Chadwick and B. Prennel, Eds. Springer, September 2004, pp. 79–96.