Development of an E-commerce Site with Smartcard Payment Mechanism

Christopher S. Lacey
MEng Electronic Systems Engineering with Management Studies

Supervisor: Mr. P J Miller

Electronic Engineering
School of Engineering and Applied Science
Aston University

Submitted: May 2001

Acknowledgements

The author wishes to express his gratitude to the following:

Mr. P J Miller and Dr. J A R Williams of the School of Engineering and Applied Science, Aston University, for providing ongoing advice and assistance for the duration of the project.

Mr. J Ward and Mr. P Trevis also of the School of Engineering and Applied Science, Aston University, for providing technical support.

Hitachi Smart Commerce division for the donation of smartcard equipment and development software; specifically, Mr. J Griffiths for providing technical support and to Mr. R Evans for arranging sponsorship.

Mr. M Meyerstein of BT Cellnet for providing information and source code with respect to Mondex value transfer.

Contents

Table of Figures

1 Synopsis

In an attempt to provide a solution to the problem of using credit cards for payment over the Internet, the objective of this project was to implement a fully functioning E-commerce site which utilised a smartcard mechanism for payment..

Due to the fact that the creators of existing smartcard wallets appear reluctant to divulge their full specifications, a smartcard-based electronic cash system has been developed from scratch, providing means for instantaneous, anonymous transfer of value across an insecure network, such as the Internet. Analysis and test of the system have suggested the implementation to be secure.

Additionally, smartcard technology has been employed to solve another perceived problem with business-to-consumer E-commerce sites: that of the need for repetitive personal data entry. A profile system has been created which permits storage of personal data in one location (the smartcard), and rapid completion of HTML forms by automatic retrieval of this information.

An E-commerce site has been created with which these two systems have been successfully integrated, indicating that smartcard technology does provide a feasible means for addressing the problems identified. However, complications identified at the client side suggest that widespread adoption of the technology will not occur until suitable standards are developed and adhered to.

2 Introduction

2.1 Context

The explosive growth of the Internet has caused a revolution in the manner in which businesses and consumers conduct commercial exchanges. *E-Commerce* is currently a major growth industry, and the number of transactions carried out online is escalating exponentially.

The advantages provided by Internet commerce are self-evident, and explain the enthusiasm shared by companies and customers for trading in this manner. For the supplier, there is greater potential to compete on a global scale, and cost savings can be attained in terms of staff and real estate by removing the need for public-facing premises. For the consumer, a means is provided to browse and search for products, and compare the prices of different suppliers, more quickly and easily than was previously possible.

However, some problems have arisen with respect to business-to-consumer ("B2C") systems, which to this day have prevented them from realising their full potential.

Firstly, there is a general reluctance amongst the public to transfer their credit or debit card number across the Internet, for fear of it being intercepted and unlawfully misused. The use of encrypted communication (via SSL[a]) has gone a long way to alleviate this fear, but it does still remain an issue: a significant number of potential purchases are lost for this reason.

Secondly, credit and debit cards are not ideally suited to purchasing many of the products or services that are available, or could be made so. In many situations, a system more resembling cash would be preferential - avoiding delays inherent within credit card clearing systems, permitting *micropayments* (*e.g.* of a few pence) to be made for online services, retaining customer anonymity and providing a means for the user to be aware of his current balance at all times.

Finally, the tedious activity of repetitively entering personal information, such as shipping address, for every transaction or site registration is disconcerting to many users. A research study conducted by Jupiter Communications (NY) in 1999 indicated that more

[a] Secure Socket Layer

than a quarter of users surveyed had abandoned a transaction solely due to the length or complexity of the form which had to be completed.

Various solutions to these problems have been proposed, such as SET[a] and Web 'Beanz' for payments; and 'Autocomplete' and Microsoft Profile Assistant for completing forms. However, each of these systems solves only one of the problems mentioned: SET, for example, avoids the need for transmission of credit card information, but still uses such a card as the ultimate means for payment. In addition, most solutions tend to be tied to a user's own computer, preventing them from being used effectively in Internet cafés or on other machines.

2.1.1 Applicability of Smartcard Technology

Smartcards have two fundamental capabilities - that of data storage and processing power. In terms of the former, they provide advantages in terms of their portability and - more uniquely - the fact that the data stored upon them can be made tamper-proof. Physical security of the cards provides good protection against attempts to read or modify the contents of memory by external means. Data can therefore only be accessed via the interface defined by the program resident on the card, meaning that a system could be created whereby information is not released from the card unless a correct PIN[b] is entered beforehand, for example.

The processing power of the card is of particular use for cryptographic and other sensitive operations, where - for example - digital signatures can be generated and validated without a user's private key ever leaving the card.

Smartcards' tamper-proof data storage, and their capability to perform cryptographic operations, therefore appeared to provide a feasible means for addressing the problems previously described: firstly, by allowing user profile information to be stored and quickly transferred by insertion of the card into a smartcard reader; secondly, by providing a secure means for value to be stored and transferred by means of a trusted applet resident on the card.

[a] Secure Electronic Transactions standard
[b] Personal Identification Number

2.2 Requirements

The aim of this project was to design and create a fully operational E-commerce site which would make use of smartcard technology in order to attempt to provide a solution to the problems previously discussed. Users would therefore be required to have a suitable smartcard reader attached to their computer in order to make use of this.

2.2.1 Electronic Cash

In order to represent "real" cash most accurately, there is a requirement for **(i) value to be stored on the smartcard only**, and not recorded elsewhere (thus maintaining anonymity).

Clearly, therefore, a card applet needs to be created which will **(ii) prevent users from defrauding the system by increasing their balance without authorisation from the card issuer**; and **(iii) prevent debits from being made without the user's consent**.

Finally, there needs to be a means for **(iv) securely transferring value between the card and a remote host, over an insecure network** which is open to eavesdropping and modification of the data being transferred.

It was initially the intention to use *Mondex*, a proven smartcard-based electronic cash product, as the basis for value transfer within this project. However, Mondex International, creators of the system, were unwilling to provide the protocol specifications required for security reasons. Consequently, a suitable electronic cash system had to be developed from scratch.

2.2.2 Personal Profile

The basic requirement was for the smartcard to **(i) store textual information within a number of profile "fields"** and for a means to be provided for **(ii) a web site to retrieve this data**.

Additionally, provision had to be made for a user to **(iii) update his profile information whenever necessary**, and to **(iv) specify preferences as to whether, and how often, a PIN should be requested before sensitive information is released**.

The objective to develop a personal profile was not part of the original project specification. It was taken on at the request of Hitachi, providers of smartcard readers and development software for the project.

2.2.3 E-Commerce Web Site

The core functions provided by any E-commerce site are to **(i) allow users to browse or search within a database of items or services, (ii) select those required for purchase and place into a virtual "shopping basket"**, and **(iii) "check out" by authorising value transfer to the vendor**.

Additionally, for this project, it was necessary to **(iv) utilise the electronic cash and personal profile systems** previously described.

2.3 Overview of Report

Three fundamental components to the overall system clearly emerge, namely *Server*, *Client* and *Smartcard*. The following chapters deal with these individually: for each, a *Design Issues* chapter describes and justifies the major conceptual decisions made, and an *Implementation* chapter outlines key methods by which the design was realised.

The method by which value transfer was achieved is described separately in *9 Cryptographic Challenge and Response Cycle*, as it involves the server, client and smartcard equally, and cannot be satisfactorily described for each component in isolation.

The success of the project is then evaluated, and possibilities for future development identified. Finally, conclusions are drawn from the findings made.

3 Server-Side Design Issues

3.1 Choice of Web Server

Different permutations of operating system and web server software were assessed, and in many cases evaluated through installation, test, and review of their accompanying documentation.

UNIX servers tend to be highly regarded for their reliability and stability, hence initially appeared to be an attractive option for deployment. However, at the time when this assessment of alternatives was being conducted, Mondex was intended to be the means by which payments would be made within the site, and it was therefore assumed that a smartcard reader would be needed on the server in order to facilitate card-to-card value transfers. The majority of such devices are supplied only with Windows drivers - UNIX alternatives are generally slow to materialise and are usually unsupported - and as such the decision was made to select Windows NT Server as the platform upon which the web server should run.

The Apache HTTP Server and Microsoft's Internet Information Server (IIS) were then critically compared, the latter finally being selected due to the fact that its native scripting language (ASP[a]) was unique in providing support for instantiation of Windows COM[b] objects, thus permitting communication with the smartcard reader via vendor-supplied components. Various modules providing ASP support for Apache were located and tested, but these were found to possess incomplete feature sets - none of them providing COM support.

3.2 Server-Side Processing

An E-commerce site obviously requires some degree of server-side processing over and above simply relaying static content at the request of browsers - for example, supplying pages which show products that match a user's search criteria. The decision was made to utilise IIS's inline scripting capabilities to perform the majority of server-side processing, as this technique uses the application's memory space to process the scripts, draining fewer

[a] Active Server Pages
[b] Component Object Model

resources than using CGI[a], with which a new process needs to be created to serve every separate page request.

IIS is capable of supporting a number of different scripting languages (VBScript and JScript[b] as standard, Perl and other alternatives by deployment of appropriate modules). No one language appeared to offer any particular advantage, hence VBScript was chosen solely because this appears to be the most common choice amongst users of ASP, and thus more documentation and support is available for it.

3.3 Maintaining State

The protocol via which web pages are requested and served (HTTP[c]) is *stateless*, in that each page request is effectively an isolated event whereby a connection is maintained between client and server for the transmission of a single file only. Navigating to a particular page by entering its URL[d] into the address bar of a browser or by selecting a hyperlink causes a TCP[e] connection to be made to port 80 of the appropriate host, followed by an instruction of the form:

```
GET /filename.html HTTP/1.0
```

Assuming the file requested is available, the web server then responds by transmitting the page back to the client, and the connection between the two machines is immediately released.

The stateless nature of HTTP demands that consideration be given to how some form of persistence be created within the system. Clearly, for an E-commerce site, it is desirable for a user to move between several pages, browsing and searching for items, and adding those which are required to a virtual “shopping basket”. It is obviously necessary for the selections made to be retained for at least the duration of the user’s visit to the site, and preferably also between successive visits.

Various extensions to HTTP, which are now mature and supported by the vast majority of browsers, provide means to overcome this problem. The most established method is

[a] Common Gateway Interface
[b] Microsoft’s implementation of ECMAScript (Javascript)
[c] Hypertext Transfer Protocol
[d] Uniform Resource Locator
[e] Transmission Control Protocol

known as *HTTP Authentication*, whereby the web server inserts the following headers into its initial response to a page request:

```
WWW-Authenticate: Basic
HTTP/1.0 401 Unauthorized
```

Browsers supporting HTTP authentication will then prompt the user for a username and password, cache this information, and retransmit it within the headers of every subsequent page request to that site. Consequently, by observing the username and password accompanying each page request, the server can ascertain which user is being served and modify the contents of the information returned accordingly.

An alternative to this technique is to use *cookies*, which are small text files created by the browser on the user's system upon the request of the server. The information to be stored within these files is sent to the client within the response HTTP headers, and this is then retransmitted by the browser within the headers of every subsequent page request to the same site. Consequently, by using cookies to store a unique identifier for a particular user, page responses can be personalised by the server accordingly.

When cookies first came into existence (in Netscape Navigator 2.0), they were viewed with suspicion by some users who regarded the act of text files being saved on their own machines as a security threat. The fact that these files contain textual data which is managed by the browser, only ever sent to the site which originally created it, and never executed, has caused this attitude to be no longer widely held, and although users can still choose to reject them, all browsers in common use today silently accept cookies by default. In addition, creation of the concept of "session cookies" which exist in the browser process's volatile memory, and thus remain in existence only until the program is closed, have served to provide a technique for providing persistence to which few are opposed.

In general, cookies are used much more widely than HTTP Authentication, and as such the appearance of the browser's login box is now rarely witnessed and can be disconcerting for inexperienced users. In addition, because the manner in which the username and password are requested is unique to each individual browser, it would not be possible to provide a consistent mechanism for logging in using a smartcard should this method be employed for user identification. Consequently, despite the small possibility of user rejection, it was decided that the less controversial "session cookie" be used to provide login facilities to the site, together with an explanation of the security issues involved.

Additionally, it was decided to provide the option for creating a permanent cookie to provide automatic login to the site.

3.4 Database

There was a clear need for the existence of a database on the server, which would contain information relating to the products or services available for sale on the site (*e.g.* description, price and stock information).

In addition, it was believed to be desirable for the database to be the location in which the contents of users' "shopping baskets" was held. It would be possible for the session cookie created on the client's machine to retain this information; however, because cookie data is transmitted with every page request, use of a server-side database minimises the amount of data being transferred, and thus the speed of response. Additionally, it permits the contents of baskets to be remembered indefinitely (*i.e.* between visits).

A large number of different database products are available, all with their own particular advantages and disadvantages. Assessment of which was the most suitable for an E-commerce site would be dependent upon factors such as the expected server load, and a detailed evaluation of alternatives was not considered to be within the scope or budget of this project. Use of a standard query language and abstraction layer to connect to the database (specifically, SQL via an ODBC[a] connection) was therefore deemed essential, so that the underlying engine could be replaced should it become necessary (easing migration to a heavy-duty Oracle database should server load escalate, for example).

3.4.1 Database Transactions

Certain operations that may need to be carried out on the database are comprised of a number of separate actions - such as the process of checking out, where each product purchased must be deleted from the user's basket and a corresponding record made elsewhere to authorise dispatch.

It is important to ensure that, in the event of an error or problem occurring on the server, the entire operation would fail or succeed as a single unit (*i.e.* either *all* of the component actions would be completed, or *none* of them). Such an operation is said to be *atomic*.

[a] Object Database Connection

In order to achieve this requirement, it was decided to use transactional features within the database whereby a transaction is commenced immediately before the first critical operation and *committed* following the final one. Should any step in the transaction fail, all other steps are automatically rolled back, thus preserving consistency of the data.

The database engine also ensures that partial results of incomplete operations are never obtained by other concurrent processes (*e.g.* other ASP page requests) by locking the relevant fields for the duration of the transaction. Due to the fact that transactions only appear to be required for records and fields specific to individual users, no problems should occur with the database being made unavailable to other users of the site.

3.5 Encrypted Communication

Communication via the Internet is inherently insecure, as data transferred over it is open to eavesdropping at many different points. Consequently, there is a need for any system which involves the transmission of sensitive information (such as credit card numbers) to provide means for encrypting communication between server and client and to assure users of the true identity of the site with which they are dealing, hence the rationale behind deploying SSL[a].

3.5.1 Public and Private Keys (Asymmetric Cryptography)

When used for encryption, public and private keys are analogous to padlocks and their keys, whereby information can be encrypted using a public key (locked with a padlock) such that it can only be decrypted using its associated private key (unlocked with its key).

Consequently, public keys can be widely distributed to enable anyone to encrypt information in the knowledge that it can only be decrypted by the holder of the associated private key.

Additionally, however, data encrypted using the private key can be decrypted by anyone in possession of the associated public key. If meaningful information can be extracted using the public key, one can be certain that only the holder of the private key could have encrypted it initially (*i.e.* the origin of the data is assured). This technique forms the basis for digital signature systems.

[a] Secure Socket Layer

3.5.2 Digital Certificates

In order for asymmetric cryptography to be used effectively, it is necessary for the holder of a public key to be certain that the key is in fact associated with the private key owned by the intended recipient. Using techniques such as IP spoofing, it might be possible to masquerade under the identity of another user or server and supply a different public key which would allow unauthorised access to information encrypted with it.

In order for the true owner of a public key to be determined, *Certification Authorities* (CA's) such as Verisign and Thawte have been established to act as trusted third parties which will verify the identity of a person or organisation before providing them with a *digital certificate*. Such certificates are wrappers containing textual information concerning the owner's identity, together with the actual public key. The wrapper itself is signed using the CA's private key which gives anyone proof of its authenticity. The CAs' public keys, required to verify certificates as being valid, are supplied as standard within all common web browsers and web server software.

3.5.3 SSL and Certificate Authentication

SSL is a standard technique used for secure data exchange on the Internet and on private networks. Typically, a web server will present a browser with its digital certificate which will act as proof of its identity, and contain its public key using which data sent to it can be encoded.

As Internet Information Server and the majority of mainstream browsers support SSL, the decision was made to apply for a suitable digital certificate from a known CA, and to make use of SSL for the transmission of address and credit card information to the server.

4 Server Implementation

Please refer to the accompanying CD-ROM to view the commented code and other files created to implement the system ('Web' directory for ASP code, 'Server' directory for Access database and Java application source).

4.1 ASP Syntax

A description of ASP syntax and usage is not within the scope of this document. However, the reader may benefit from knowing that directives are distinguished from standard HTML code by being surrounded by <% and %> delimiters. In addition, the <!--# .. --> tag is used for specifying for server-side includes.

4.1.1 Code Convention

Within this section, highlighted text will be used to distinguish server-side instructions from HTML code, where necessary. Comments within code will be shown in green.

4.2 Separation of Code and Presentation

ASP inline scripting (where pre-processor instructions are embedded amongst standard HTML tags) is not well suited to separating code from presentation information. However, where possible, user redirects and function libraries were utilised to isolate significant code blocks to "dedicated" script files.

4.2.1 The need for inline code embedding

In some situations, it was however essential to embed code in the middle of an HTML page, as in the case of a product listing - illustrated by the following pseudocode:

```
Page header HTML: Title, Table header, etc.
FOR EACH Product that matches selection criteria
      New table row
      New table cell
      Print Product name
      New table cell
      Print Product price
NEXT Product
Page footer
```

Figure 4.1: Pseudocode indicating use of inline scripting

4.2.2 Function Libraries

In order to minimise the code content of pages, and to reduce coding repetition, a number of libraries were created containing related functions (*e.g.* those pertaining to database access). These were included, when required, by use of server-side includes, as shown in the following example:

```
<% @LANGUAGE = VBScript %>
<!--#include file="../std_lib.asp"-->
<!--#include file="../db_lib.asp"-->

<html>
<head>
<link rel="stylesheet" href="main.css">
</head>
<body>
<p>Current order total: <strong><%=getBasketTotal()%></strong></p>

etc...
```

Figure 4.2: ASP code showing use of function libraries and server-side includes

ASP does not support the concept of "libraries" as such; the #include directive simply causes the contents of the referenced document to be placed at that point within the file. Consequently, care had to be taken to avoid duplication of variable and function names across multiple libraries, as should more than one ever be included in a document, confusion would result. Suitably descriptive names were therefore used for functions and "global" variables that might need to be referenced externally; and "local" variables were prefixed with a common identifier (e.g. db_ for database variables), as shown in the example below:

```
<%
Dim objConn, db_strConn, db_strQuery, objRS
Set objConn = Server.CreateObject("ADODB.Connection")
db_strConn = "DSN=SmartMarket;Database=SmartMarket;UID=sa;PWD=;"
objConn.Open(db_strConn)

Function isEmptyRS(db_strQuery)
      Dim db_checkRS
      Set db_checkRS = objConn.Execute(db_strQuery)
      isEmptyRS = db_checkRS.EOF
      db_checkRS.Close
End Function

etc...
```

Figure 4.3: Extract from db_lib database access library

4.2.3 User Redirects

In many cases, the submission of a form to the server requires a considerable amount of processing to be carried out before a simple page is returned indicating success or failure.

As opposed to embedding HTML representing the user message within large amounts of ASP code, these messages were created in separate pages, and HTTP redirect responses sent to the client, causing it to request the appropriate “success” or “failure” page after processing of the form was complete. For example, the following script is an extract from the ASP script called on submission of the “Add new user” form:

```
'Check to see if user already exists in database
userExists = Not isEmptyRS("SELECT Username FROM Accounts WHERE Usern ...

If Request("password") <> Request("password2") Then
      'Passwords do not match
      db_finalise()
      Response.Redirect("adduser_diffpass.asp" & urlAppend)

ElseIf username="" Or password="" Or forename="" Or surname="" Then
      'Essential field left blank
      db_finalise()
      Response.Redirect("adduser_incomplete.asp" & urlAppend)

ElseIf userExists Then
      'User already exists
      db_finalise()
      Response.Redirect("adduser_exists.asp" & urlAppend)

Else
      'OK to insert user into database
      dbExecute("INSERT INTO Accounts (Username,Password,Forename,Sur ...
      db_finalise()
      Response.Redirect("addusersuccess.html")
End If
```

Figure 4.4: Extract from adduser.asp showing use of user redirects

In this case, if the two passwords entered by the user do not match, he is forwarded to “adduser_diffpass.asp”; if any essential fields are left blank, he is forwarded to “adduser_incomplete.asp”, and so on.

4.3 Database Structure

A Microsoft Access database was used for storing user and product information; this product being chosen solely for its ease of availability (see *3.4 Database Design Issues*).

4.3.1 Relationships

Being a relational database, it is possible to define relationships that exist between fields in different tables. Such usage enforces *referential integrity* - by defining the structure at such a low level, faulty code or interfaces are prevented from entering invalid data into the database, allowing problems to be recognised more easily and thus assisting in debugging.

The tables created, and relationships between fields are as shown below. Primary keys are indicated in bold.

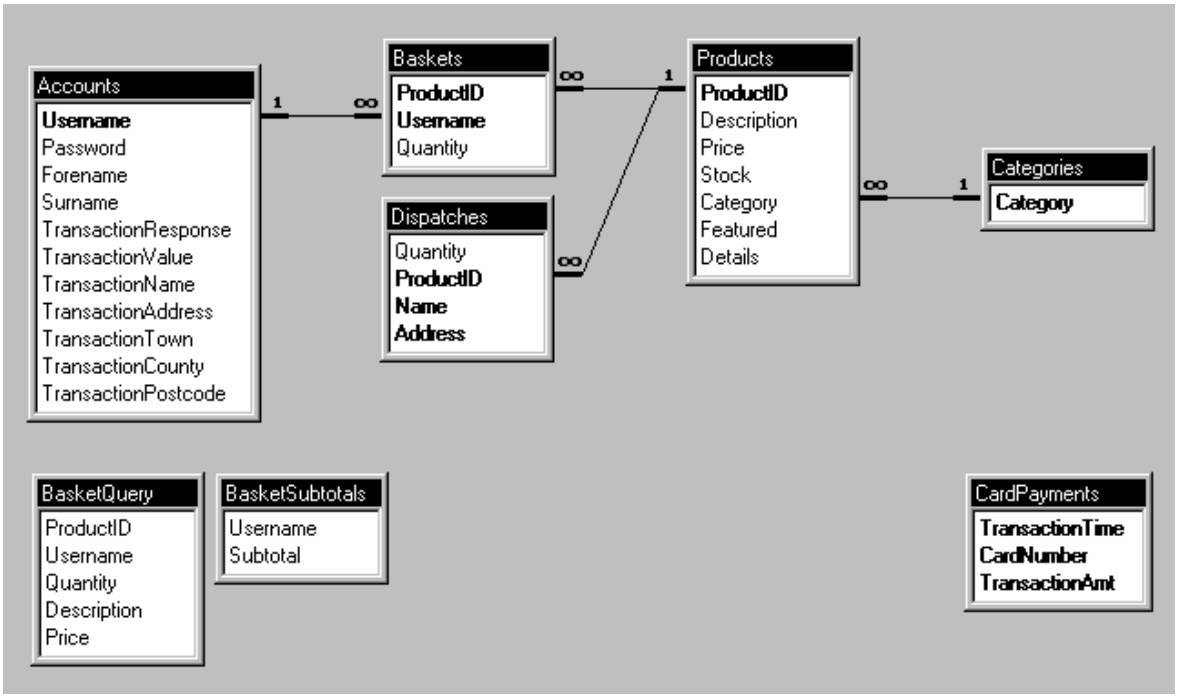


Figure 4.5: Database structure

4.3.2 Tables

The **Accounts** table contains a record for each registered user, comprising username and password, real name, and the shipping address most recently entered (*i.e.* for the current transaction). **Products** contains the full list of items available, each being assigned a category from the list defined in **Categories**. These categories are used within the browsing process - the user being able to view all relevant products by selecting the required category from a dynamically created drop-down list within his web browser.

Baskets brings together user and product information (from **Accounts** and **Products**, respectively) to indicate the items selected by users to be in their "shopping baskets". Within Access, the relationships defined between fields can only be of the type One-to-Many, as shown (*e.g.* a product may only be assigned to one category, but a category may be assigned to many products). In order to provide a Many-to-Many relationship for the shopping baskets (a user may select many products, and a product may be selected by many users), **Baskets** is required to act as a *junction table*, possessing two primary keys to ensure uniqueness only of the combinations of (ProductID,Username) tuples (*i.e.* it is possible for multiple records to exist with the same ProductID, providing the Username differs for each such record; and vice-versa).

Once a purchase is made, products (referenced by ProductID) are "moved" to the **Dispatches** table, which would be used by the shipping department to arrange for despatch. Each record relates to one product only, to allow individual components of an order to be considered separately. Hence, whenever sufficient stock was available, a delivery could be made, and the record deleted from the system. The Name and Address fields relate to the shipping details entered during the checkout stage, and are not stored elsewhere in the system to ensure no permanent record is made of them, for user privacy reasons.

CardPayments represents the list of credit card transactions that have been processed. In the case of this system, a credit card transaction is deemed to have been successfully carried out once an entry for it is made into this table. Obviously, in a real E-commerce site, transaction authorisation would be made by a bank or clearing house - entry of data into this table, however, simulates this process.

4.3.3 Queries

BasketQuery and **BasketSubtotals** are *queries* created within the database. Effectively, these are read-only tables which are dynamically created by the server.

BasketQuery defines a link between its ProductID field and that of **Products** and thus provides the data required for the server script to display details of a user's basket (without need for it to perform its own cross-referencing to obtain product description, for example).

BasketSubtotals links its Username field to that of **BasketQuery**, and calculates the total cost of products within each user's basket by defining the Subtotal field to be an expression of the form:

```
Subtotal: Sum([BasketQuery]![Quantity]*[BasketQuery]![Price])
```

i.e. the summation of (quantity × price) for each item in BasketQuery associated with a particular username.

4.4 Server-Side Java Application

VBScript's "long" integer type was found to use a 32-bit signed number representation, hence was capable of representing numbers with maximum magnitude of 2^{32}. This was insufficient for performing calculations with 64-bit numbers, as used for the challenges and responses within the smartcard debit and credit procedures (see *9 Cryptographic Challenge and Response Cycle*).

Consequently, a Java application was created which provided methods for performing the calculations, accepting arguments and returning values as strings, and utilising the java.math.BigInteger class which provides for representation of arbitrary length numbers.

Recent versions of the Java VM[a] for Windows allow any Java application to be instantiated as a COM component - this facility was exploited to provide a means for the ASP code to call the methods created, as illustrated below:

```
Dim sw_smartIDChallenge
Set sw_smartIDChallenge = Server.CreateObject("CSL.SmartIDChallenge")

Function preemptResponse(sw_value, sw_challenge)
      preemptResponse=sw_smartIDChallenge.preemptResponse(sw_value,
                                          sw_challenge, sw_privateKey)
End Function
```

Figure 4.6: ASP code to call 'preemptResponse' method in Java application

Here, 'CSL.SmartIDChallenge' was the identifier (CLSID[b]) used to register the Java applet as a COM component on the server (via the JAVAREG utility).

[a] Virtual Machine
[b] Class ID

4.5 SSL

A free “test” server certificate was obtained from the Verisign CA, and using this, SSL was successfully configured within IIS. However, due to the fact that such test certificates expire after two weeks and the difficulty of obtaining repeat issues, SSL was not deployed within the finished system. This should not be regarded as demeaning its necessity in any way - should a release system ever be created, a suitable server certificate would need to be purchased, and SSL re-enabled.

5 Client-Side Design Issues

5.1 Hypertext Markup Language (HTML)

All web pages are ultimately passed to a browser's rendering engine as HTML, this being an application of SGML[a]. In their simplest form, HTML documents consist of a basic block of text, interspersed with control codes ('tags' of the form <address>..</address>) to indicate the nature or desired appearance of the "marked up" text. Tags also exist for providing hyperlinks to other pages, displaying images, embedding Java applets, providing form components to receive user input, and so on.

The language is officially managed by the W3C[b], who define the complete list of tags, their usage, and the manner in which user agents (*i.e.* web browsers) should render content. However, HTML has been in a constant evolutionary process since its inception; consequently different browsers support different versions of the language and very few have ever correctly supported the standard in its entirety. Additionally, vendors have attempted to provide enhancements via their own proprietary additions (for example, Netscape Navigator's <blink> and Microsoft Internet Explorer's <marquee> tags).

User agents are required to handle tags which they do not recognise by simply ignoring them, and as such, careful design permits advanced features to be implemented transparently - providing enhanced functionality to those browsers which support it, but still permitting users of less well featured browsers to view the basic information present on the page. Internet usage statistics suggest that over 85%[c] of users view pages with a browser which supports HTML 4.0 (revised April 1998)[1]; consequently it was decided that this version be used as the basis for developing the site, whilst ensuring that older browsers (supporting HTML 3.2[2] and earlier) be capable of providing core functionality, albeit with a less refined appearance and user interface. Proprietary features have been avoided completely.

Differences in users' system configurations (*e.g.* screen resolution, colour depth, font availability), and in the interpretation and implementation of the HTML standard within browsers, mean that the manner in which web pages will appear can never be completely predictable. Despite the temptation to do so, considerable effort was made to avoid

[a] Standard Generalized Markup Language
[b] World Wide Web Consortium
[c] Source: browserwatch.internet.com

designing the site for use on particular browsers and systems, instead making correct use of HTML to ensure that the site be accessible from all platforms - both those currently in existence and those only just emerging, such as “Web TV” and other embedded systems. However, due to the fact that around 95% of web site hits are made using Microsoft Internet Explorer or Netscape Navigator, the site was rigorously tested using these two browsers to ensure an intuitive and aesthetically pleasing service for the majority of customers.

5.1.1 Frames

“HTML frames allow authors to present documents in multiple views, which may be independent windows or subwindows. Multiple views offer designers a way to keep certain information visible, while other views are scrolled or replaced.”[1]

The decision was made to employ frames to provide a persistent toolbar for easily navigating around the site (providing links to browse products, view shopping basket, log out, etc.), as illustrated below. Additionally, because the toolbar frame remains in existence for the duration of a visit, this was deemed the best place to embed the client-side Java applet for communicating with the smartcard.

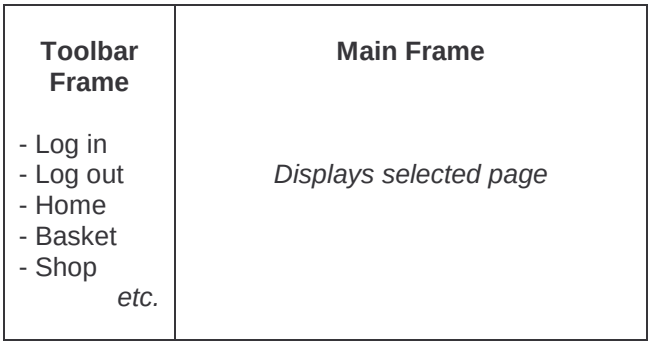


Figure 5.1: Main frame structure

Similarly, an additional frameset within the main frame itself was considered most appropriate for enabling users to search for and browse categories of products on the site:

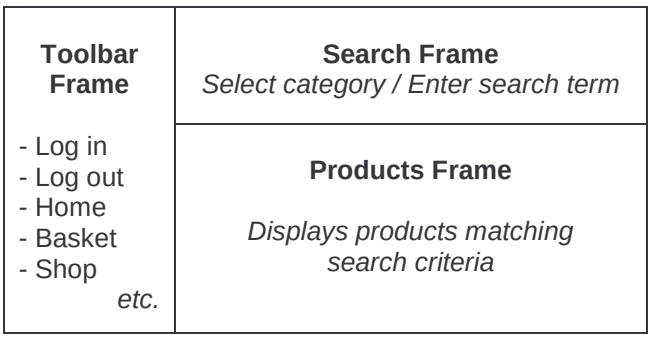


Figure 5.2: Browse/search products frame structure

Each frame within a frameset displays the contents of a separate HTML page, hence a frameset document itself is fairly small - typically defining only the structure, layout and URLs of the individual frames. It is also possible, however, to provide content within <noframes>..</noframes> tags which is rendered by browsers that do not support frames. This facility was used to provide links to the most relevant page (*e.g.* the "welcome" page within the opening frameset); and all pages within the site were designed to provide an alternative route to each page, should the static toolbars be absent. Links were used in preference to repetition of content within the <noframes> section in order to ease maintenance of the site (updates would otherwise require changes to be made in two places), and in order to prevent unnecessary amounts of data from being transferred to the majority of users who would be using frames.

5.1.2 Forms

"An HTML form is a section of a document containing normal content, markup, special elements called 'controls' (checkboxes, radio buttons, menus, etc.), and labels on those controls. Users generally 'complete' a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing." [1]

Being the only method for obtaining feedback from a user in a web application, forms were clearly required for such actions as selecting categories of products to view, and submitting search criteria, quantities and shipping details to the server.

Each control is given a unique name, and upon submission, key-value pairs of the form (control name, control state) are passed to the server either within the headers of the HTTP request (the 'POST' method), or as a suffix to the URL (the 'GET' method) thus:

```
http://www.aston.ac.uk/?forename=fred&surname=bloggs
```

Some browsers place an upper limit upon the length of a URL, constraining the number of key-value pairs that can be transmitted using GET. Additionally, this method is clearly unsuitable for transmitting sensitive information as the URL (in its entirety) is likely to be clearly displayed within the user's address bar, and may well be cached by the browser and/or a proxy server. For these reasons, it was decided that POST be used for the

submission of almost all forms within the site. This technique also has the advantage of causing the information sent to the server to be encrypted if an SSL[a] session is used. The only exception to this decision was for the submission of product category or search criteria to the "show products" page - in this case, only a small amount of information is ever transmitted (category name or search term), and the availability of URLs of the form:

```
http://www.server.com/products.asp?category=stationery
http://www.server.com/products.asp?searchterm=pens
```

provides an intuitive means for others to provide links to particular product pages on the site. Also, as the response obtained from a particular category or search term is likely to change infrequently (only when new products are added), it would be safe for a server or browser to cache the page returned, referenced by its extended URL.

5.1.3 Client-side Scripting

The vast majority of browsers used today possess a scripting engine which permits client-side code to be run within the browser environment (*e.g.* Netscape Navigator 3.0 and later, Internet Explorer 3.0 and later, and Opera 3.0 and later). Usage of this scripting capability can provide considerable benefits to both client and server - for example, it can be used to validate forms before they are submitted to the server, preventing the user from wasting time waiting for a response when the form data is invalid, reducing network traffic and relieving server load.

However, despite the fact that scripts run in a "sandbox", a plethora of security "holes" have been discovered in the major browsers during recent years, allowing scripts to observe or modify elements of the external system to which they should not be permitted access. Whilst the vendors have typically been quick to provide patches to correct such problems, some users have lost confidence in client-side scripting and have disabled it within their browsers. Consequently, it is not possible to rely upon this technology for achieving core functionality.

As a result, the decision has been made to utilise client scripting for the non-essential task of ensuring form data is valid before submission (in addition to server-side validation), and as the only means for providing an interface between the client-side Java applet and components on the web page. Use of the smartcard is not essential for accessing and using the site, hence this, again, does not detract from core functionality.

[a] Secure Socket Layer

The standard HTML scripting language in use today is ECMAScript, originally developed by Netscape (as JavaScript), and offered for standardisation in Autumn 1996. All browsers which support client-side scripting support this language, hence this was the natural choice for use within the project.

5.1.4 Dynamic HTML

Dynamic HTML ('DHTML') provides an enhanced user experience by permitting client-side scripts to modify the appearance of the web page once retrieved from the server, allowing - for example - objects to be moved and hidden in response to particular events.

The method by which such dynamic activity is achieved differs markedly between browsers, however. Many do not support any form of DHTML, whilst the two most commonly used products (Netscape Navigator 4 and Internet Explorer 4/5) possess such radically different document object models that any form of dynamic user interface effectively needs to be written twice - once for each browser. Despite the publication of a standardised DOM[a] by the W3C[3], it is only with the recent release of Navigator 6 that some form of commonality between the Netscape and Microsoft products has been recognisable.

Consequently, DHTML has not been employed at all within this project - due to the author's perception that it offers little benefit to the end user, and his reluctance to expend considerable effort over learning and supporting non-standard, proprietary implementations.

5.2 Cascading Style Sheets (CSS)

Separation of content from style is generally regarded as desirable, due to the fact that the role of providing and maintaining information ('content') is very different from that of designing an aesthetically pleasing and logical user interface. Historically, HTML has blurred the boundaries between the two - a page will typically contain information, perhaps sectioned using descriptive tags such as <address>, together with formatting commands such as <font size='+3'> (to increase text size).

CSS provides a means for defining style in a separate location from the main HTML body (usually in a separate file entirely), and offers a far richer set of commands to define appearance. For database-backed sites, where the underlying data is already separately

[a] Document Object Model

maintained within the database, CSS provides the advantage of allowing multiple pages to link to a single stylesheet, thus allowing changes to be made across the site by making just one set of modifications.

The majority of current browsers support CSS Level 1[4] (Internet Explorer 3.0 and later, Netscape 4.0 and later), hence it was decided that they be used throughout the site. Pages viewed on a non-supporting browser might appear less attractive, but would retain full functionality.

5.3 Client-Side Java Applet

In order for communication with the smartcard to be effected, the need became apparent for the creation of an intermediate program which was capable of interfacing both with the web browser and also with the smartcard reader attached to the computer. Client-side scripting technology is incapable of achieving the latter directly, hence the most obvious means for achieving this was to create a Java applet that would be capable of running within the Java Virtual Machines (VM's) present in the majority of mainstream browsers.

Old Java VM's prevented applets from conducting operations which were considered potentially harmful, such as accessing hardware, via a process known as *sandboxing*. However, the security model of the Java platform 1.1 does permit such operations to be conducted under certain circumstances (typically, after seeking the user's permission). Java 1.1 has been in existence for many years, hence is implemented in the vast majority of browsers used today.

Additionally, via use of a system created by Netscape known as *LiveConnect*, it has become possible for client-side scripts running in browsers to interface to Java applets embedded within the page - specifically, by use of the netscape.javascript.JSObject class in the Java code, and the 'mayscript' attribute in the <applet> tag within the host HTML.

It was clearly preferable for the applet to make use of an established standard to communicate with the smartcard reader, as opposed to making a direct connection via the serial/parallel port and being tied to one particular type of device. Consequently, it was decided to make use of a Java-based framework known as OCF[a] that acts as an abstraction layer between software and hardware, and for which drivers are available for a number of devices.

[a] Opencard Framework

6 Client Implementation

Please refer to the accompanying CD-ROM to view the commented code and other files created to implement the system ('Web' directory for user interface (HTML, CSS and images), 'Client' directory for client-side Java applet source).

At the time of writing, the finished sites were available from http://ee-pc43.aston.ac.uk/laceycs/sc and http://ee-pc43.aston.ac.uk/laceycs/sm .

6.1 Web User Interface

Two separate sites were ultimately created - one representing the card issuer (*SmartCentre*), and providing card personalisation and top-up facilities; the other (*Aston SmartMarket*) being the E-commerce site itself which makes use of the *SmartID* personal profile and *SmartWallet* electronic cash facilities.

Pages were created using a text editor - development tools such as Microsoft FrontPage and Macromedia Dreamweaver were avoided, as the code they generate was found to make use of proprietary features within mainstream browsers which can prove problematic when viewed in other environments.

Images were initially created in a vector-based drawing package, namely CorelXara 2.0, which allowed for the creation of standard styles and modification of graphics (*e.g.* enlargement and rotation) without loss of definition. When the design process was complete, anti-aliased images were saved in GIF[a] for buttons and logos, which is a compressed, non-lossy, format providing a maximum palette of 256 colours. Photographic images were saved using JPEG[b] format, which provides 24-bit colour depth, but uses a lossy compression algorithm (making it unsuitable for images containing highly defined edges, such as logos).

[a] Graphical Interchange Format
[b] Joint Photographic Experts Group

6.1.1 SmartCentre Site

Separate pages were created for:

- configuring the card (entering personal data, specifying security settings)
- changing the PIN
- unblocking the card by entering the correct unblock code
- viewing the balance
- crediting ('topping up') the card by use of a credit card
- repeating a top up which failed due to a network failure

together with various informational pages, including success/failure feedback for top-up attempts.

Most of the pages used forms to obtain user input, passing the data to the card by use of the client-side Java applet. Topping up and repeating top-ups were the only pages to require server-side processing of forms (to generate the correct cryptographic response); the remaining pages work correctly if run locally in a web browser (*i.e.* not fetched from a server), providing users a means for configuring their cards without connecting to the network, if required.

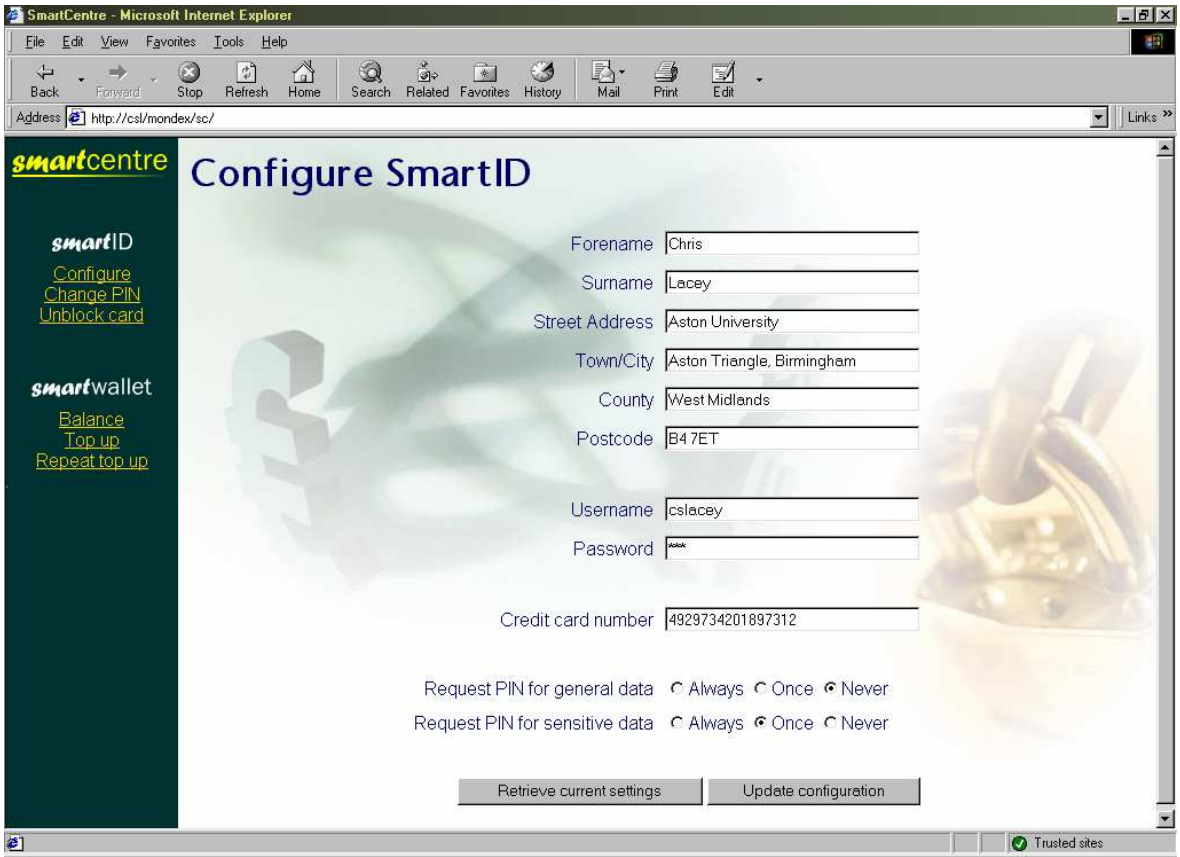


Figure 6.1: Form used to configure personal profile

Frames were used to provide an easy-access toolbar, as seen in figure 6.1 above, which also proved to be a suitable static location for embedding the applet.

6.1.2 Aston SmartMarket Site

Separate pages were created for:

- creating a new user account
- logging in
- viewing shopping basket contents and modifying quantities
- providing search facilities by category or textually
- displaying products matching the search criteria
- adding a specified quantity of a product to the shopping basket
- checking out using a credit card
- checking out using SmartWallet
- repeating a SmartWallet transaction that failed due to a system or network problem

together with various informational pages, including success/failure feedback for actions such as creating a new user and making credit card or smartcard payments.

Again, a frameset was created to provide a static toolbar, although navigation through pages was such that operation in a non-frames environment is possible.

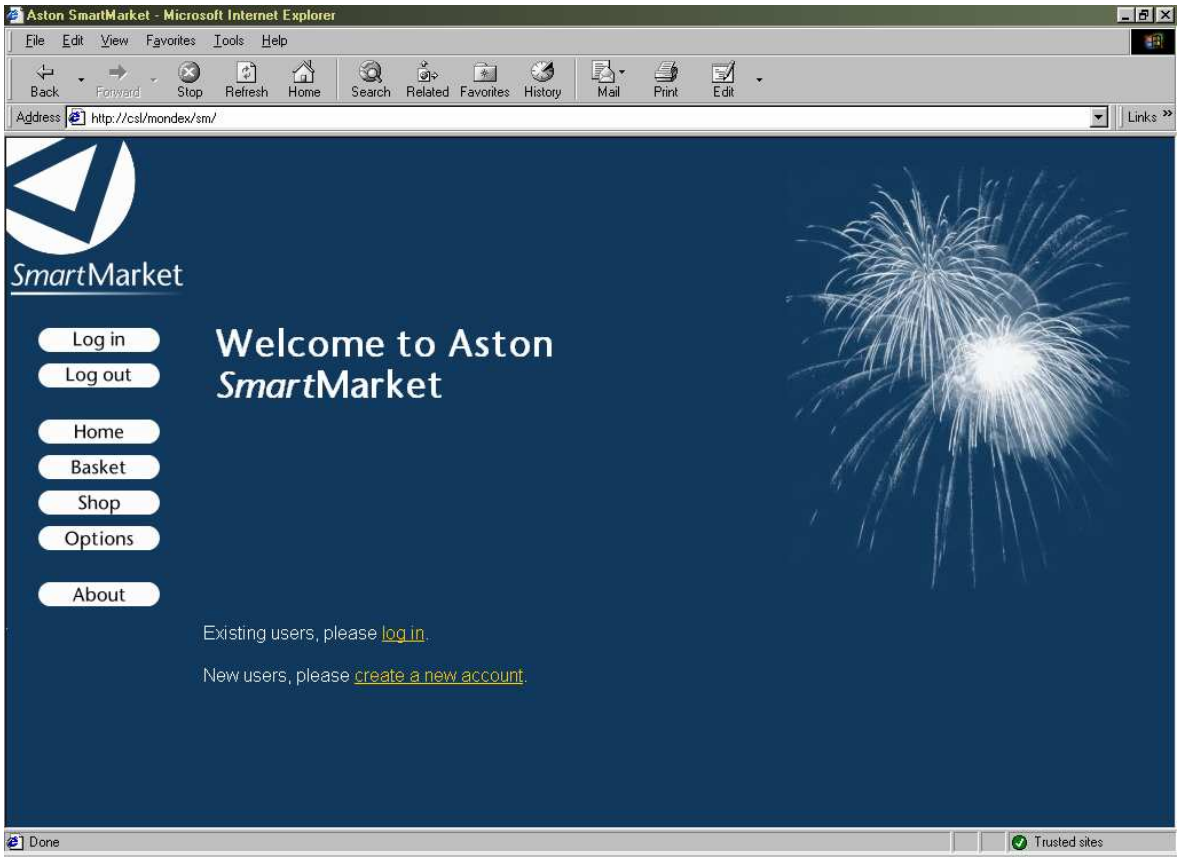


Figure 6.2: Aston SmartMarket front page

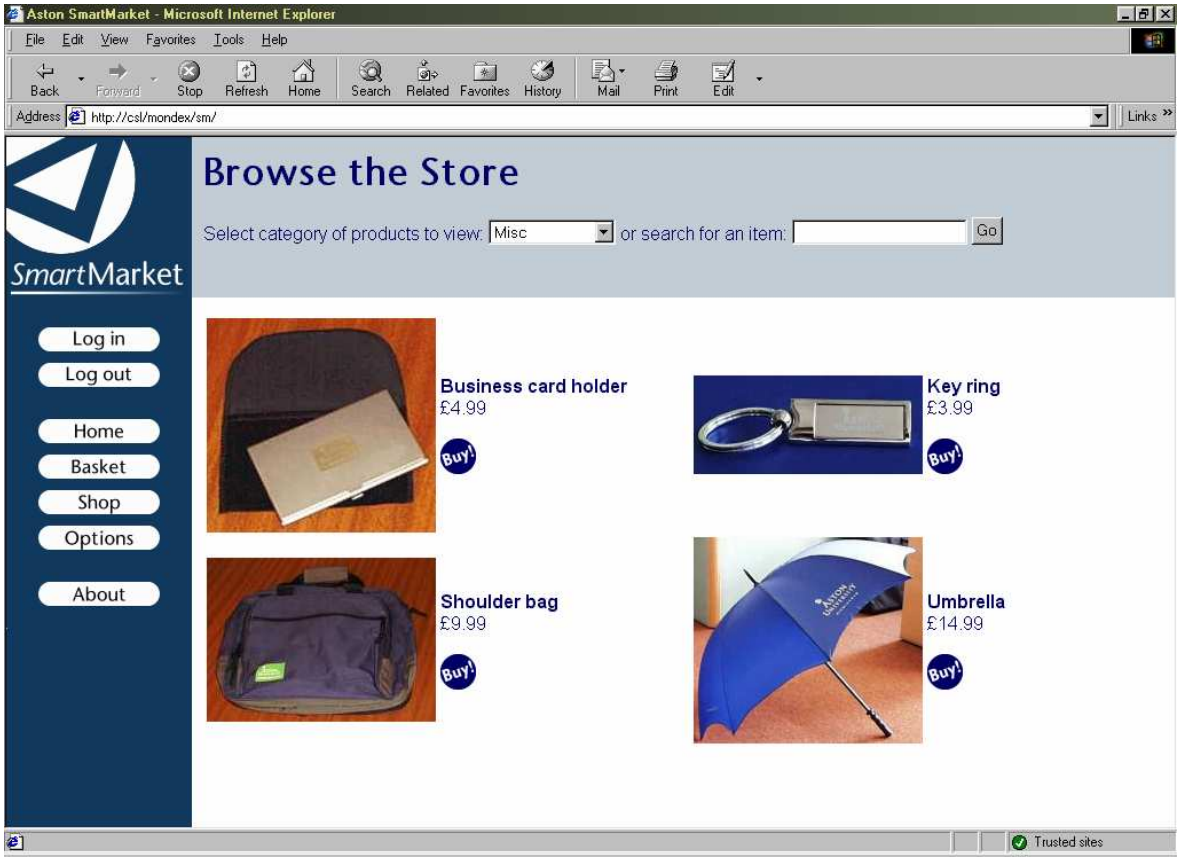


Figure 6.3: Pages to search product database and view results

6.2 Client-Side Java Applet

6.2.1 Interface Methods

The applet created provided the following methods which can be accessed from client-side scripts, thus allowing others to implement *SmartID* and *SmartWallet* functionality into an existing site relatively easily:

- getBalance() - returns balance as integer
- getField(String fieldName) - returns profile information as string
- setField(String fieldName, String fieldValue)
- debit(int value, String Challenge) - returns cryptographic response as string
- repeatDebitResponse() - returns cryptographic response as string
- prepareCredit(int value) - returns cryptographic challenge as string
- credit(String Response) - returns success status as boolean
- sendPin(int pin)
- setPin(int pin)
- setSecurity(boolean privateSec, boolean privateOnce,
  boolean generalSec, boolean generalOnce)
- getSecurity() - returns security settings as string of the form “0110”
- unblock(String code) - returns success status as boolean
- beginConversation()
- endConversation()

Most are fairly self-explanatory, and correspond to the core smartcard functions described in *9.1 Smartcard Feature Set*.

Possibilities for fieldName in **getField(..)** and **setField(..)** are “username”, “password”, “forename”, “surname”, “streetaddress”, “town”, “county”, “postcode” and “ccnumber”.

Within **setSecurity(..)**, the flags are as defined in *8.1.1 Smartcard PIN requests* (true=1 and false=0). **getSecurity()** returns the current security settings in the same order as specified in **setSecurity()**, where 0 represents false (off) and 1 represents true (on).

The concept of “conversations” was created in order to prevent the card PIN from having to be repeatedly entered if a page requested several profile fields from the card, and the card was configured always to require the PIN. A block of card instructions on a single page may be surrounded by **beginConversation()** and **endConversation()**, during which

time if the card requests the PIN, the applet will cache it and automatically return it to the card when required.

6.2.2 Netscape Navigator and Internet Explorer Security Models

Differences between the security models of browsers caused problems in the implementation of the applet, as the means for requesting permission to perform "unsafe" features differs in Netscape Navigator and Internet Explorer.

Within Netscape Navigator, additional code was required within the applet that caused an error to be thrown when run in Internet Explorer. Consequently, a slightly different version of the applet had to be created, and then placed in a JAR[a] file, and digitally signed using a Netscape-specific mechanism.

In Internet Explorer, permission was granted by adding the web site to the list of "trusted sites", for which full Java permissions then had to be enabled. Creation of a CAB[b] file which was digitally signed would allow this process to become more automated (presenting a screen to the user allowing him to allow or deny permissions upon download). However, such a mechanism is also proprietary, and insufficient time was available for this to be successfully implemented.

6.2.3 Drivers for Smartcard Readers

Although native OCF drivers have been created for many smartcard readers, it was found that their installation tends to be fairly complicated (involving manually editing configuration scripts), and is disconcerting for many users. PC/SC[c] drivers, however, now appear to be supplied as standard with the vast majority of readers, usually in a form more suited to installation by non-technical users (*e.g.* by means of an *InstallShield* "wizard").

Although they cannot be directly controlled within Java, OCF provides a means to access PC/SC drivers. Therefore, despite resulting in the creation of an extra abstraction layer, this configuration was used to permit the simplest means of client installation.

[a] Java Archive
[b] (Microsoft) Cabinet
[c] PC/Smartcard

7 Smartcard Design Issues

7.1 Choice of Operating System

Three multiple application smartcard operating systems are commonly in use - namely, Sun's *JavaCard*, Microsoft's *Windows for Smartcards* and Maosco's *MULTOS*. The functionality provided by each appears to be fairly similar - the major difference in implementations being the language in which development takes place (Java, Visual Basic and assembly language, respectively).

The decision was made to select MULTOS cards for development, solely because this is the system upon which the Aston University *Smart Campus Card* is based. Consequently, it should be possible for the applications written for this project to be loaded onto an Aston card should a suitable application load certificate be obtained from the MULTOS CA[a].

7.2 Card-Client Communication

ISO 7816 Part 4 defines a standard means for card-client communication which is supported by MULTOS and the vast majority of smartcard implementations. Communication takes place via a sequence of APDU's[b], which are effectively messages or packets of data formatted in a consistent manner.

7.2.1 Command APDU's

Command APDU's (*i.e.* those sent *to* the card) consist of a standard four-byte header, optionally followed by a body of arbitrary length containing data, as shown in *Figure 7.1* below:

Header				*Body*		
CLA	INS	P1	P2	Lc	*Data*	Le

Figure 7.1: ISO 7816-4 Command APDU structure

where the CLA (class) and INS (instruction) bytes identify which operation is being requested of the smartcard; P1 and P2 are parameters specific to the function called; Lc contains the amount (length) of data present in bytes; Le contains the expected length of the response data in bytes.

[a] Certification Authority
[b] Application Protocol Data Units

7.2.2 Response APDU's

Response APDU's consist of an optional data body, followed by a two-byte trailer, as shown in *Figure 7.2* below:

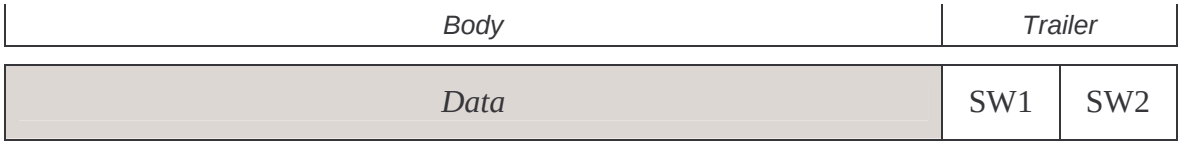


Figure 7.2: ISO 7816-4 Response APDU structure

where SW1 and SW2 comprise the *Status Word*, providing information as to the success or failure of the operation (*e.g.* 90,00 represents successful; 69,82 security status not satisfied).

7.2.3 APDU Cases

Consequently, four fundamental types of command exist, as defined by ISO 7816:

- *Case 1* - No command or response data present
- *Case 2* - Only response data (from card) present
- *Case 3* - Only command data (to card) present
- *Case 4* - Both command and response data present

Cases 2, 3 and 4 were deemed suitable for use with respect to the personal profile and wallet functions created for this project, *e.g.* case 2 for retrieving profile information, case 3 for updating profile information, and case 4 for issuing a debit instruction (where the client sends a cryptographic challenge and the value to be debited, and receives the card's cryptographic response).

8 Smartcard Implementation

Please refer to the accompanying CD-ROM ('Card' directory) or *Appendix 7* to view commented code for the card applet created.

8.1 Feature Set

As described in *2.2 Requirements*, the general functionality required of the smartcard was the storage and retrieval of data in the personal profile, and provision of a "wallet" for electronic cash. For more detailed information upon implementation of the latter, refer to *9 Cryptographic Challenge and Response Cycle*.

It would have been possible, and perhaps desirable, to implement two independent applets for providing the profile and wallet features. However, to minimise confusion, all functionality was provided within a single applet for the purposes of this project.

The following instruction set was defined and implemented within MAL[a] code. The numbers in parentheses represent length in bytes, and the prefix '0x' indicates a hexadecimal value.

INS	Feature	Input Data	Output Data
0x0*n*	Return profile information		ASCII {30}
0x1*n*	Set profile information	ASCII {30}	
0x20	Debit wallet	Crypto challenge {6} Debit value {2}	Crypto response {8}
0x21	Prepare for credit	Credit value {2}	Crypto challenge {8}
0x22	Attempt credit	Crypto response {8}	
0x23	Repeat last debit response		Crypto response {8}
0x30	Return balance		Balance {2}
0x40	Submit PIN	PIN {2}	
0x41	Set PIN	New PIN {2}	
0x42	Update security settings	Settings {4}	
0x43	Return current security settings		Settings {4}
0x44	Unblock card	Unblock code {8}	

Figure 8.1: Implemented smartcard feature set

8.1.1 PIN Requests

As an additional measure of security, the applet was designed not to perform "dangerous" operations (such as debits and profile changes) unless the correct PIN is submitted directly beforehand (using instruction 0x40).

[a] Multos Assembly Language

In addition, it was made possible for the user to configure the card as to whether and how often (always, once per session, or never) PIN entry is required for retrieving profile information. Options can be specified for "sensitive" data (password and credit card number) and for "general" data (all other fields).

The four bytes sent and received as security settings in instructions 0x42 and 0x43 represent four flags {privateSec, privateOnce, generalSec, generalOnce}. privateSec indicates whether the card should request PIN entry when retrieving private information, and privateOnce how often this should occur (1 = once per session, 0 = every request). The setting of the latter is irrelevant if privateSec is set false. Similarly, generalSec and generalOnce specify whether and how often the PIN should be requested for the retrieval of general data.

8.2 Developmental Process

Coding was carried out using a text editor, and Hitachi's *Multos Development Tools* used to compile, debug and load the MAL code onto a smartcard.

Very basic functionality was implemented in the early stages of development, with extra features being added once testing proved that those already in existence worked as intended. Extensive use was made of the "MSDT" card simulator to insert breakpoints into code, then transmit particular APDU's and step through each line of the applet. The system permitted the values of chosen variables to be observed ("watched") whilst stepping through the code, and showed the contents of registers, the stack and the transaction shadow data area - all of which proved invaluable in identifying errors within the code.

To permit testing of the applet upon the smartcard itself, the *Terminal Simulator* was used to upload the applet (by selection of a suitable load certificate, designed for the development cards), transmit APDU's and view the card response.

9 Cryptographic Challenge and Response Cycle

9.1 Requirements

Previous design decisions dictated that the only record of the value stored within a user's wallet was to be on the smartcard itself (see *2.2.1 Electronic cash requirements*). Consequently, a system needed to be created whereby:

1. it was impossible for a user to increase his balance by any means other than a transaction authorised by the card issuer.
2. a server issuing a debit instructions could be assured that the balance had been successfully decreased (before shipping goods, for example).

Furthermore, communication between server and card had to occur via an insecure, untrusted network/client terminal, open to eavesdropping and modification of the data being transferred.

9.2 Implemented Solution

The only means by which the requirements previously detailed could be met appeared, therefore, to be by means of cryptography and the storage of keys on both the server and the smartcard (see *3.5 Encrypted Communication*).

The fact that code run on the smartcard can be trusted (see *2.1.1 Applicability of smartcard technology*) means that the card can be relied upon to assess whether a credit instruction originated from an authorised source before increasing the balance. Additionally, the card applet can be created such that a particular response (verifiable by the server, and incalculable except using the private key on the card) is generated to indicate that a debit instruction was successful (subject to user authorisation by entry of a PIN).

The fact that, for this project, the card issuer was the same entity as the site which authorised transactions, meant that it was acceptable to use symmetric cryptography (*i.e.* make use of one private key only, stored on both the server and the card). Consequently, the security of the system would be broken if the private key was ever divulged to a third party. In a release system, the use of asymmetric cryptography or simply a different

private key for each card (as in GSM[a] SIM[b] cards) would reduce the potential for such catastrophic destruction of the system's security!

9.3 Debit Procedure

The procedure to debit the card involves the generation of a *challenge* by the server, which is transmitted to the client together with the value to be debited. The card then attempts to decrease its balance by the amount specified and - if successful - digitally signs the (challenge,value) data using its private key. The response thus generated can then be returned to the server which, using the same private key as that present on the card, can assess whether the correct response has been returned and thus whether the debit instruction was carried out.

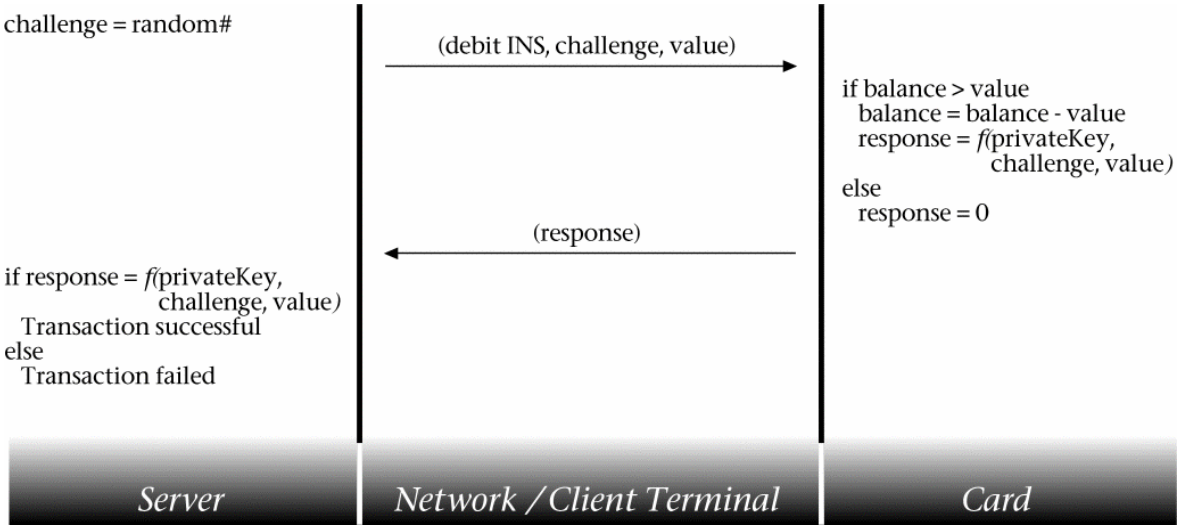


Figure 9.1: Debit communication sequence

The algorithms typically used to perform digital signatures are DES or 3-DES. However, neither was not available on the development card provided; consequently, the function performed was simply an exclusive-or of the 64-bit private key with a concatenation of the 48-bit challenge and the 16-bit debit value. Clearly, such a function is in no way suitable for digital signatures, as it is easily reversible, allowing the private key to be calculated. If implemented on a release card, the values would instead be passed to the digital signature algorithm provided; and the same function used on the server.

Each challenge generated by the server needs to be unique (a *nonce*), so that the correct response can never be obtained by reference to historic data. For the purposes of this

[a] Global System for Mobiles
[b] Subscriber Identity Module

project, use of a 48-bit random number with over 280 billion possible values, is regarded as being sufficient. Deployment in the real world may require the use of longer numbers and a different means for calculating the nonce, in order to provide higher levels of security, and reduce the possibility of *brute-force* attacks from succeeding.

9.4 Credit Procedure

For topping up the card, a similar strategy is used to that employed within the debit procedure. However, in this case the challenge is generated by the card, and thus an additional stage is required for the challenge to be requested, as shown:

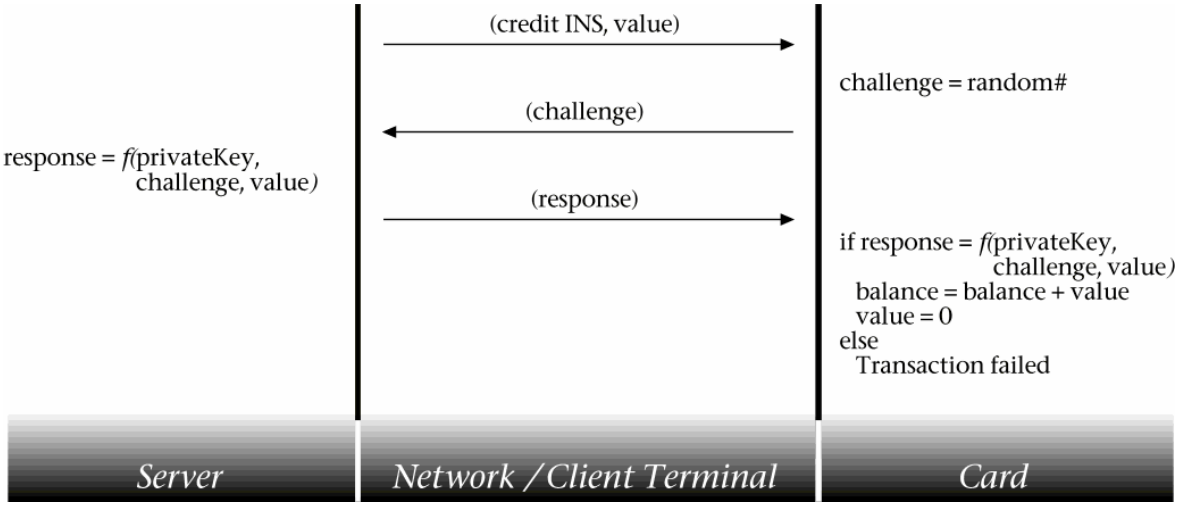


Figure 9.2: Credit communication sequence

'value' is set to zero after completion of the top-up to prevent replay of the reponse causing multiple credits.

9.5 Tolerance to Network Failures

In the event of a network failure, the situation may arise where value is successfully debited from the card, and the response is generated but never received by the server. Consequently, the card applet was coded such that the response, once generated, is stored in non-volatile memory; and a command added to the instruction set which causes the card to repeat its last response. Hence, this facility could be used repeatedly to re-attempt transmission of the response over the network until successful.

Similarly, in the event of a network failure preventing a response from reaching the card in the final credit stage, the server needs to be able to re-transmit this as many times as

necessary (*i.e.* take responsibility for making a record of the last response calculated for each user).

9.6 Tolerance to System Interruptions

In the same way that transactions were employed to prevent partial completion of database processes on the server (see *3.4.1 Database Transactions*), cards transactions have been used to encompass the critical actions shown in *Figures 9.1* and *9.2*. Consequently, if power to the card was cut (*e.g.* by removing (*tearing*) the card) between critical operations, any changes made would be rolled back when the card was next powered up - hence in the case of debits, subtraction of the balance would be undone if a response was not calculated.

9.7 Security

The security model of the cryptographic challenge/response cycle appears to be sound, resilient to fraud attempts such as replay attacks. However, there is no check made during the debit procedure that the debit request is being made by an authorised entity - instead, the user is relied upon to ensure that the site/application they are using is trustworthy (perhaps by reference to an SSL certificate). Within a release system, it would be beneficial to reduce the possibility of unauthorised debits by requiring requests to be themselves signed in some manner.

A test site was created indicating the different stages of the debit and credit procedures, and allowing values to be modified to simulate fraud attempts. This is included on the CD-ROM ('Web/test' directory) and at the time of writing was available from http://ee-pc43.aston.ac.uk/laceycs/test/debit.html and http://ee-pc43.aston.ac.uk/laceycs/test/credit.html .

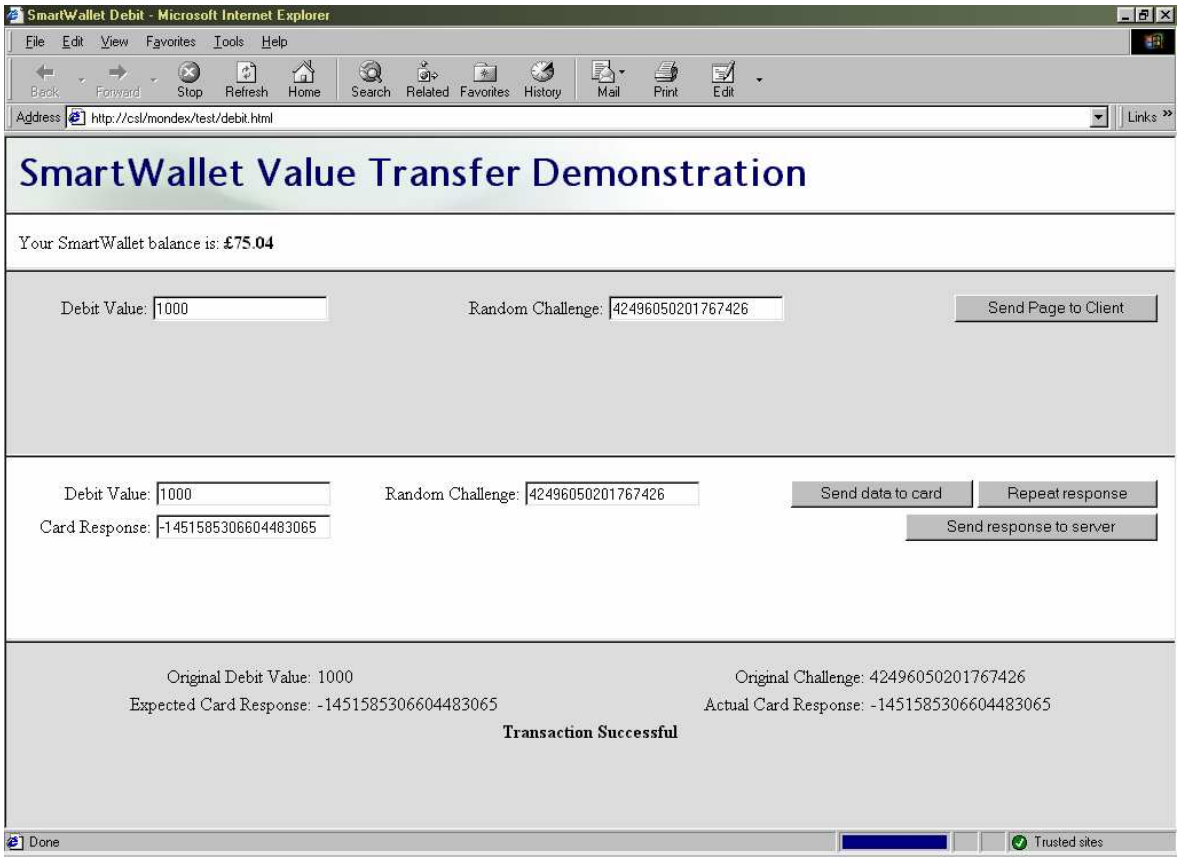


Figure 9.3: Debit test site

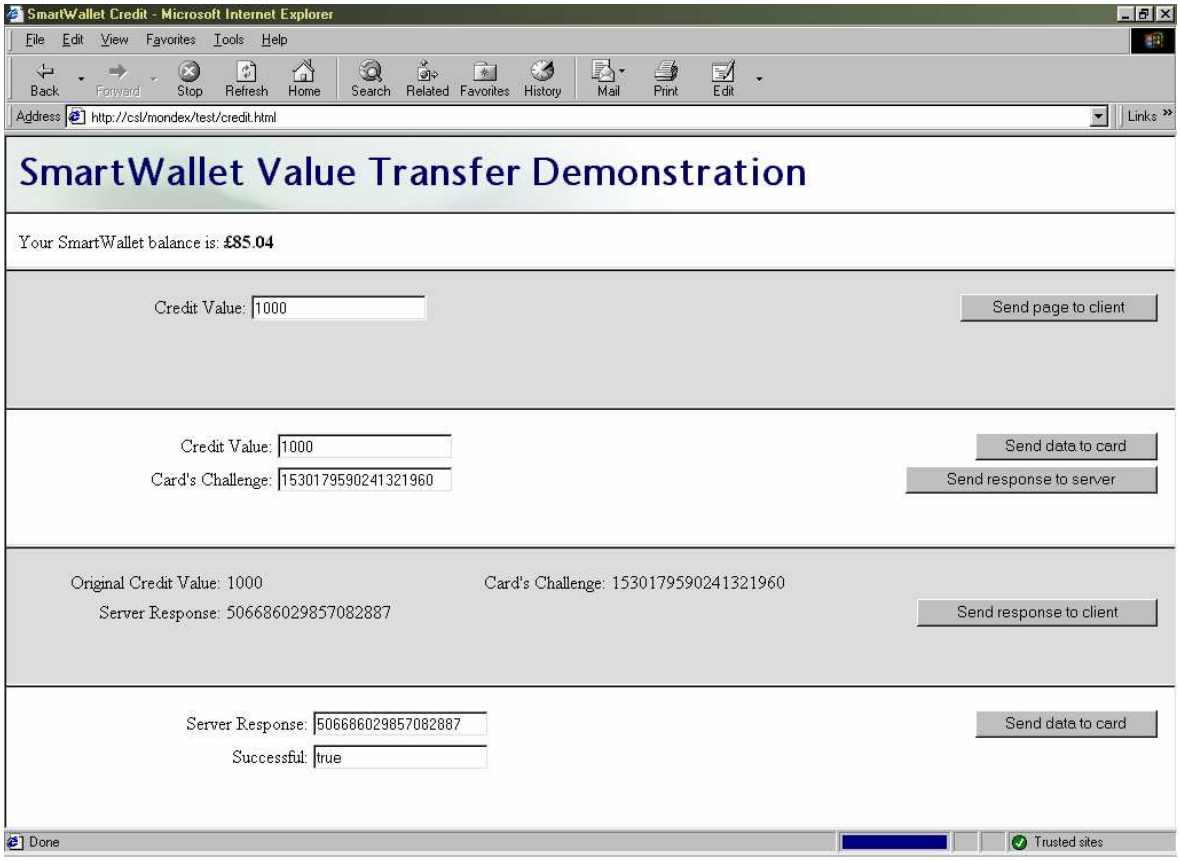


Figure 9.4: Credit test site

10 Evaluation

The author believes the project to have been successful, in meeting and exceeding the original aims. An E-commerce site has been successfully created, and - in the absence of suitable specifications from Mondex - a secure smartcard payment system has been created from scratch. Additionally, a smartcard personal profile system has been created in response to the request of Hitachi's Smart Commerce Division.

Several people were asked to test the sites created, and feedback obtained from them was used to refine the user interface. Considerable testing has also been carried out by the author to ensure the stability of the entire system.

A site was created to demonstrate the workings of the credit and debit procedures, permitting modifications to be made to the data passed to the card and server. Attempts to defraud the system (*e.g.* by decreasing the value passed to a card for a debit attempt, or conducting replay attacks) all failed.

10.1 Project Costing

The only direct costs involved with the development of the system were with respect to smartcard hardware and development tools (kindly donated by Hitachi):

1 GIS Smart Mouse smartcard reader	£23.50
2 MULTOS 3 Test Cards @ £20 each	£40.00
Total:	**£63.50**

Additionally, two PC's (a server and a workstation) and various operating systems and pieces of software (Windows, Multos Development Tools, Access) were utilised.

The author believes the time spent researching and developing the system to be in excess of four hundred hours. Additionally, around twelve hours were spent in consultation with academic staff and Hitachi representatives; and approximately three hours with support staff.

10.1 Possible Future Development

Should further time have been available, it would have been desirable to address the following issues:

1. Remove the need for the site to be added to the list of "trusted sites" and for full Java permissions to be explicitly granted in Internet Explorer, by providing a version of the client-side Java applet which was digitally signed using Microsoft's signing technology. Alternatively, research the feasibility of using Sun's Java Plug-in for Internet Explorer, Netscape Navigator and Opera to provide a standard means for running and granting access rights to code.
2. Reduce the potential for unauthorised card debits by requiring debit requests to be digitally signed.
3. Reduce the complexity of the installation process required by users, by creating an InstallShield wizard (or similar) to automate the process.
4. Increase the level of user feedback provided when transactions fail by providing a fuller range of card response codes to indicate different errors
5. Ensure validation of HTML forms occurs both on the server side and the client side in all cases.

In the longer term, the project could be developed further by:

1. Performing server load tests, and implementing a system more suited to high volume traffic (*e.g.* by deploying an Oracle database)
2. Providing additional means for accessing the site, *e.g.* via WAP/WML on mobile telephones
3. Creating client-side software to configure the smartcard and manage the personal profile, without need for accessing the "SmartCentre" web site.

11 Conclusion

The successful creation of smartcard-based electronic cash and personal profile systems, and the integration of these into a fully-functioning E-commerce site, suggest that smartcard technology does provide a feasible solution to some of the problems which prevent E-commerce sites from realising their full potential. Specifically, the end-product created for this project provides a portable, secure means for users to store and transfer electronic cash over the Internet, providing the same benefits that real cash has over credit cards with respect to micropayments, anonymity and speed of transaction. Additionally, the system allows users to complete online forms rapidly, and removes the need for sites to retain records of personal data - providing benefits in terms of privacy and removing the need for users to inform numerous sites whenever details change.

Currently, the use of smartcard technology for these purposes is most fundamentally constrained by the low number of card readers in general use. However, the increasing deployment of smartcards in banking, and the inclusion of smartcard facilities in Windows 2000, can only serve to expedite their increase in popularity.

Nevertheless, a number of other issues have been identified which, until resolved, appear likely to limit the utility of smartcard technology with respect to E-commerce.

Considerable difficulties were encountered in accessing the smartcard from within the web browser: the only means discovered being by use of a framework (OCF) which is too complicated to install for the majority of users. Also, differences in the security models of browsers meant that applets had to be developed which were application-specific or required complicated configuration steps to be carried out.

Additionally, the reluctance of Mondex International to provide their specifications to individuals with whom they have no commercial trust agreement, seems to be indicative of the attitudes of the developers of most electronic cash systems, and suggests that strong confidence still does not exist in the security of these systems.

Widespread adoption of smartcard technology for E-commerce will therefore probably not occur until a standard means for web browsers to interface with them has been developed, and until an open standard exists for the storage and transfer of electronic cash, allowing deployment by all.

References

[1] Ragget D et al: *HTML 4.0 Specification*, revised 1998, W3C

[2] Ragget D: *HTML 3.2 Reference Specification*, 1997, W3C

[3] Wood L et al: *Document Object Model (DOM) Level 1 Specification*, 1998, W3C

[4] Lie H, Bos B: *Cascading Style Sheets Level 1*, 1996, W3C

Bibliography

Cobley A: *The Complete Guide to Java*, 1997, Computer Step

Bakken S, *PHP User Manual*, 1999, PHP Documentation Group

Devargas M: *Smart Cards & Memory Cards*, 1992, NCC Blackwell

Johnson S: *Using Active Server Pages*, 1997, Que

Lie H, Bos B: *Cascading Style Sheets Level 1*, 1996, W3C

Mitchell S, Atkinson J: *Active Server Pages 3.0*, 2000, SAMS

Ragget D et al: *HTML 4.0 Specification*, revised 1998, W3C

Ragget D: *HTML 3.2 Reference Specification*, 1997, W3C

Rankl W, Effing W: *Smartcard Handbook*, 1997, John Wiley

Reynolds M, Wooldridge A: *Using JavaScript*, 1996, Que

Tanenbaun A: *Computer Networks*, 3rd ed, 1996, Prentice Hall

Wood L et al: *Document Object Model (DOM) Level 1 Specification*, 1998, W3C

Apache 1.3 User's Guide, 2000, Apache Group

Client-Side JavaScript Guide v1.3, 1999, Netscape Communications Corp.

Client-Side JavaScript Reference v1.3, 1999, Netscape Communications Corp.

ECMA Standard 262, *ECMAScript Language Specification*, 1997, ECMA

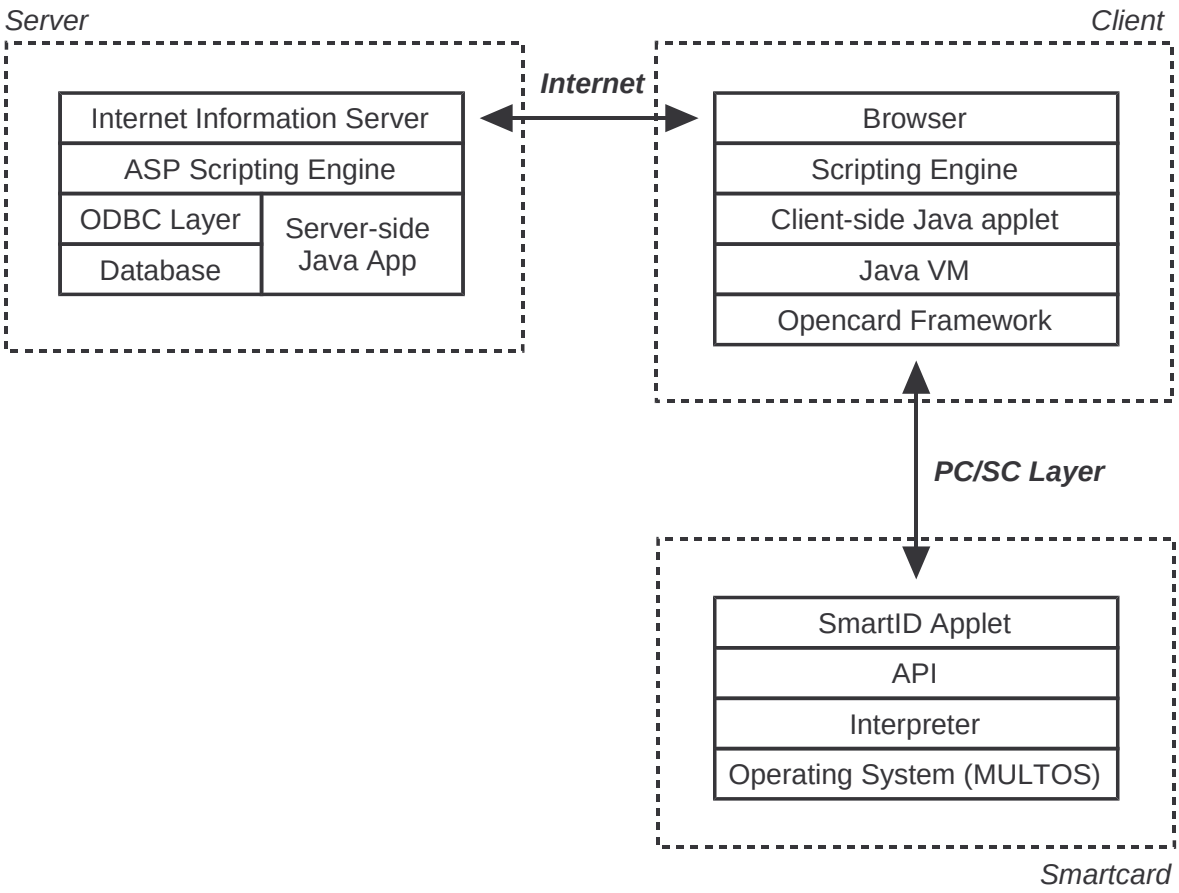
Java 2 SDK Standard Edition Documentation version 1.2.2, 1999, Sun Microsystems Inc.

Multos 4.0 Development Tools Documentation, 1999, Hitachi

Multos Developers Reference Manual v1.30, 1999, Maosco

Multos Developers Guide v1.20, 1999, Maosco

Appendix 1: System Overview



Appendix 2: Public Explanatory Material

Appendix 2.1: Introduction to SmartID and SmartWallet

Welcome to SmartCentre

This site allows configuration of your **SmartID** and **SmartWallet** applets.

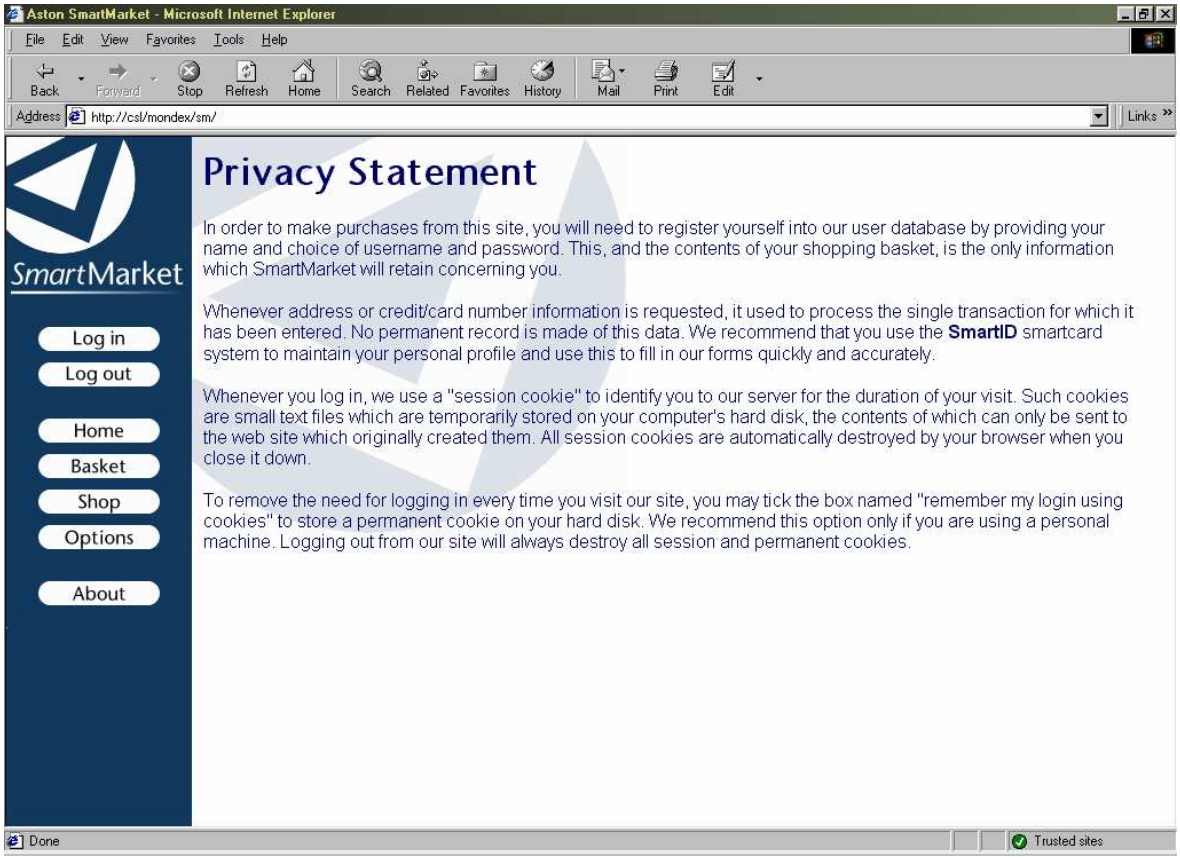
SmartID enables you to store and maintain your personal profile on your own smartcard. Sites using **SmartID** do not retain personal data, such as address and credit card details, but instead obtain it when required by querying your smartcard. **SmartID** will only release private information when you permit it to, giving you full control over how your data is handled. Additionally, by allowing you to update your profile whenever necessary, **SmartID** removes the need for informing numerous sites whenever your details change.

You can configure **SmartID** or change your PIN.

SmartWallet provides a secure means for storing and transferring value over insecure networks, such as the Internet. Sites employing **SmartWallet** as a means of payment will be able to debit or credit your card instantaneously, avoiding delays inherent within credit card clearing systems. **SmartWallet** will only permit debits to be made after seeking your permission, and as there is no need to hand over any credit or debit card details, there is no risk of your number being used unlawfully.

You can view your current balance or top up your wallet using a credit or debit card.

Appendix 2.2: Privacy Statement for SmartMarket



Privacy Statement

In order to make purchases from this site, you will need to register yourself into our user database by providing your name and choice of username and password. This, and the contents of your shopping basket, is the only information which SmartMarket will retain concerning you.

Whenever address or credit/card number information is requested, it used to process the single transaction for which it has been entered. No permanent record is made of this data. We recommend that you use the **SmartID** smartcard system to maintain your personal profile and use this to fill in our forms quickly and accurately.

Whenever you log in, we use a "session cookie" to identify you to our server for the duration of your visit. Such cookies are small text files which are temporarily stored on your computer's hard disk, the contents of which can only be sent to the web site which originally created them. All session cookies are automatically destroyed by your browser when you close it down.

To remove the need for logging in every time you visit our site, you may tick the box named "remember my login using cookies" to store a permanent cookie on your hard disk. We recommend this option only if you are using a personal machine. Logging out from our site will always destroy all session and permanent cookies.

Appendix 3: Server Installation Instructions

These instructions are specific Microsoft Windows NT 4 Server, with the Java Virtual Machine, Internet Information Server 3 (with ASP support), and Access ODBC drivers installed. Deployment on other platforms is untested.

1. Copy the web site files (on CD-ROM, 'Web' directory) into the web root directory of IIS (by default, C:\Inetpub\wwwroot) or create a new virtual directory to host the files elsewhere.

2. Copy the Access database (on CD-ROM, 'Server' directory) to a location on the server that is not a served web directory. Create an ODBC connection to this file using the Access ODBC driver
   (Control Panel → Data Sources (ODBC) → System DSN tab → Add...)

3. Copy the Server-side Java application to WINNT\Java\TRUSTLIB and register it as a COM component using the JAVAREG utility (if not on system, on CD-ROM, 'Server' directory) using the following command:
   ```
   javareg /register /class:SmartIDChallenge /progID:CSL.SmartIDChallenge
   ```

4. Reboot the system to complete registration of the COM component

Appendix 3.1: Implementing SSL

SSL is not required for the successful implementation of this system. The following procedure can, however, be used to implement it if required. Microsoft Certificate Server will need to be installed before proceeding.

Appendix 3.1.1 Generation of Server Certificate

- Start the *Internet Service Manager* (Start -> Windows NT 4 Option Pack -> Microsoft Internet Information Server -> Internet Service Manager)

- Right click on "Default Web Site" (or virtual directory if created), select "Properties"

- Directory Security tab -> Edit Secure Communication -> Key Manager

- Right click "WWW" in Key Manager, Create New Key

- Follow the wizard to create a certificate request file, ensuring "Put the request in a file" is selected on the first screen, as automatic transmission to an online authority appears

not to work correctly. On the third screen, ensure the entry for "Organisation" is the registered owner of the domain within which the web server resides (*e.g.* "Aston University" for aston.ac.uk). Also, "Common Name" should be the resolved IP address of the web server (*e.g.* ee-pc43.aston.ac.uk) - if this is unknown, determine using http://www.clara.net/dialup/support/ip.shtml .

- Request a server certificate from a Certification Authority (*e.g.* Verisign at http://www.verisign.com/site/index.html), following the online instructions and submitting the file generated by Key Manager (the CSR[a] file) when requested
- After receiving the certificate, right click the new key in Key Manager and select "Install Key Certificate". Locate the certificate and select it. If the certificate received was embedded as plain text within an Email message, it will be necessary to copy and paste the text between -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- into a new file with a .p7r extension, and then select this file.
- Close Key Manager and save changes when requested.

Appendix 3.1.2: Enabling SSL

- Right click on the virtual directory or "Default Web Site" as appropriate, select "Properties"
- Directory Security tab, enable "Require secure channel when accessing this resource" and "Accept client certificates"
- Apply settings by clicking "OK".

Once this procedure has been carried out, access to the web site will be available only via SSL (requiring https:// as the prefix to its URL).

[a] Certificate Signing Request

Appendix 4: Client Installation Instructions

These instructions are specific to Microsoft Windows 98 with Internet Explorer 5.0 and/or Netscape Navigator 4.7. Deployment on other platforms is untested.

Appendix 4.1: Drivers for Smartcard Reader

1. Install PC/SC driver for smartcard reader (driver for "SmartMouse" reader is on CD-ROM)

2. If not already present, install Java Runtime Environment (necessary to install OCF), available from http://java.sun.com

3. Install Opencard Framework library, provided on CD-ROM or available from http://www.opencard.org by issuing the command: `java installOCF`

4. Copy 'base-opt1.jar' and 'pcsc.jar' (new patch) from CD-ROM ('OCF Installation' directory) into OpenCard library directory (C:\OpenCard\OCF1.2\lib by default)

5. Add the OCF classes to the system CLASSPATH by making the following additions to AUTOEXEC.BAT:

```
CALL C:\OpenCard\OCF1.2\demos\setenv.bat
SET CLASSPATH=%CLASSPATH%;C:\OpenCard\OCF1.2\lib\base-core.jar
SET CLASSPATH=%CLASSPATH%;C:\OpenCard\OCF1.2\lib\base-opt1.jar
SET CLASSPATH=%CLASSPATH%;C:\OpenCard\OCF1.2\lib\pcsc.jar
SET CLASSPATH=%CLASSPATH%;C:\OpenCard\OCF1.2\lib\reference-services.jar
```

Appendix 4.2: Internet Explorer

1. Copy the 'opencard.properties' file installed by OCF (by default, in C:\Program Files\JavaSoft\JRE\1.2\lib) into IE's Java home directory (Windows\Java\lib)

2. Add the server upon which the site is hosted (*e.g.* ee-pc43.aston.ac.uk) to the list of "trusted sites" (Tools → Internet Options...→ Security tab → Trusted Sites → Sites...)

3. Grant full permissions to Java code originating from trusted sites (Trusted Sites → Custom Level... → Java Permissions, select "Custom"; Java Custom Settings... → Edit Permissions → Run Unsigned Content, select "Enable")

Appendix 4.3: Netscape Navigator

1. Copy the ‘opencard.properties’ file installed by OCF (by default, in C:\Program Files\JavaSoft\JRE\1.2\lib) into Netscape’s Java home directory (C:\Program Files\Netscape\Users\<username>) as ‘.opencard.properties’

2. Copy the DLL ‘OCFPCSC1.DLL’ installed by OCF (by default, in C:\OpenCard\OCF1.2\lib) to Netscape’s Java library directory (C:\Program Files\Netscape\Communicator\Program\java\bin)

3. Install the certificate for the “Aston” Certification Authority (used to create the certificate with which the applet was signed, as opposed to purchasing one) by dragging the ‘x509.cacert’ file (on CD-ROM, ‘Client/NS’ directory) into the browser window, and following the online instructions

Appendix 5: Server Code

Appendix 5.1: ASP Examples

Appendix 5.1.1: Library for calling cryptographic functions (sw_lib.asp)

```
<%
'Library of SmartWallet cryptographic functions

Const MaxBalance = 65000
Const sw_privateKey = "1311768467294899695"

'Inclusion of this library will automatically instantiate the server java COM component
Dim sw_smartIDChallenge
Set sw_smartIDChallenge = Server.CreateObject("CSL.SmartIDChallenge")

'Generate a response to a card's challenge for topup (i.e. authorise the credit)
Function createResponse(sw_value, sw_challenge)
       createResponse = sw_smartIDChallenge.createResponse(sw_value, sw_challenge,
                                                            sw_privateKey)
End Function

'Calculate the expected response from a card to indicate required debit occurred
Function preemptResponse(sw_value, sw_challenge)
       preemptResponse=sw_smartIDChallenge.preemptResponse(sw_value, sw_challenge,
                                                            sw_privateKey)
End Function

'Generate a challenge for the debit procedure
Function createChallenge
       createChallenge=sw_smartIDChallenge.createChallenge()
End Function

'Release connection to COM component
Function sw_finalise
       Set sw_smartIDChallenge = nothing
End Function
%>
```

Appendix 5.1.2: Server-side validation for registering a user (adduser.asp)

```
<% @LANGUAGE = VBScript %>
<!--#include file="../std_lib.asp"-->
<!--#include file="../db_lib.asp"-->
<%
Dim urlAppend, userExists, username, password, forename, surname
username=Request("username")
password=Request("password")
forename=Request("forename")
surname=Request("surname")

urlAppend = "?username=" & username & "&password=" & password & "&forename=" & forename &
"&surname=" & surname

'Check to see if user already exists in database
userExists = Not isEmptyRS("SELECT Username FROM Accounts WHERE Username='" & username & "'")

If Request("password") <> Request("password2") Then
     'Passwords do not match
     db_finalise()
     Response.Redirect("adduser_diffpass.asp" & urlAppend)

ElseIf username="" Or password="" Or forename="" Or surname="" Then
     'Essential field left blank
     db_finalise()
     Response.Redirect("adduser_incomplete.asp" & urlAppend)

ElseIf userExists Then
     'User already exists
     db_finalise()
     Response.Redirect("adduser_exists.asp" & urlAppend)

Else
     'OK to insert user into database
     dbExecute("INSERT INTO Accounts (Username,Password,Forename,Surname) VALUES ('" &_
          & username & "','" & password & "','" & forename & "','" & surname & "')")
     db_finalise()
     Response.Redirect("addusersuccess.html")
End If
%>
```

Appendix 5.1.3: Validating card's debit response (scauthorise.asp)

```
<% @LANGUAGE = VBScript %>
<!--#include file="../std_lib.asp"-->
<!--#include file="../db_lib.asp"-->
<!--#include file="../cc_lib.asp"-->
<!--#include file="../sw_lib.asp"-->
<%
Dim expectedResp, actualResp, transactionValue, basketTotal, name, address
actualResp=Request("cardResponse")    'Retrieves card's response to cryptographic challenge
basketTotal=getNumericBasketTotal()

'Obtain information for current transaction from database...
dbQuery("SELECT Username,TransactionResponse,TransactionValue,TransactionName,
TransactionAddress,TransactionTown,TransactionCounty,TransactionPostcode FROM Accounts WHERE
Username='" & Session("username") & "'")

expectedResp = objRS("TransactionResponse")
transactionValue = objRS("TransactionValue")
name = objRS("TransactionName")
address = objRS("TransactionAddress") & ", " & objRS("TransactionTown") & ", " &_
    & objRS("TransactionCounty") & ", " & objRS("TransactionPostcode")
objRS.Close

If transactionValue>MaxBalance Then
    'Debits of values greater than SmartWallet's maximum balance are not possible
    db_finalise()
    Response.Redirect("sc_excess.asp")

ElseIf actualResp=expectedResp And basketTotal=transactionValue Then
    'Response obtained from card is correct, and value of basket contents
    'has not changed since challenge was issued
    objConn.BeginTrans        'Begin database transaction because checking out of each item
                              'and forgetting expected response must be atomic
        purchaseBasket()
        dbQuery("UPDATE Accounts SET TransactionResponse=-1, TransactionValue=0 WHERE
                 Username='" & Session("username") & "'")
            'Clears transaction information to prevent replay attacks

    objConn.CommitTrans              'Commit transaction
    db_finalise()
    Response.Redirect("sc_success.asp")

Else
    db_finalise()
    Response.Redirect("sc_fail.asp")
End If
%>
```

Appendix 5.2: Server-Side Java Application

```
import java.io.*;
import java.math.*;

public class SmartIDChallenge {

   final int MaxCardBalance = 65000;

   public String createChallenge() {
      BigInteger num256 = new BigInteger("256");
      BigInteger result = new BigInteger("0");
      int random;
      for (int i=0; i<=6; i++) {             //Make a large random number by
         result = result.multiply(num256);   //concatanating several
         random = (int) (256*Math.random()); //random bytes (max value of byte=256)
         result = result.add( new BigInteger(Integer.toString(random)) );
      }
      return result.toString();
   }

   public String createResponse(int value, String challenge, String privateKey) {
      BigInteger num256 = new BigInteger("256");
      BigInteger result = new BigInteger("0");
      BigInteger challengeNum = new BigInteger(challenge);
      BigInteger challengeTest = new BigInteger(challenge);
      BigInteger privateKeyNum = new BigInteger(privateKey);
      byte txBlock[] = new byte[8];
      for (int i=7; i>=0; i--) {
         txBlock[i] = (byte) challengeTest.remainder(num256).intValue();
         challengeTest = challengeTest.divide(num256);
      }
      if (txBlock[7]==(byte)(value%256) && txBlock[6]==(byte)(value/256))
         //Value embedded in challenge is same as that specified (in 'value' parameter)
         //therefore return correct response...
         result=challengeNum.xor(privateKeyNum);
      return result.toString();
   }

   public String preemptResponse(int value, String challenge, String privateKey) {
      BigInteger num256 = new BigInteger("256");
      BigInteger result = new BigInteger("-1");
      BigInteger challengeNum = new BigInteger(challenge);
      if (value <= MaxCardBalance) {
         byte txBlock[] = new byte[8];
         //Determine bytes that will actually be transmitted to card...
         txBlock[7] = (byte)(value%256);
         txBlock[6] = (byte)(value/256);
         for (int i=5; i>=0; i--) {
            txBlock[i] = (byte) challengeNum.remainder(num256).intValue();
            challengeNum = challengeNum.divide(num256);
         }
         BigInteger txBlockNum = new BigInteger(txBlock);
         BigInteger privateKeyNum = new BigInteger(privateKey);
         result = txBlockNum.xor(privateKeyNum);
      }
      return result.toString();
   }

}
```

Appendix 6: Client Code

Appendix 6.1: HTML and ECMAScript Examples

Appendix 6.1.1: Using client-side Java applet with forms (configsid.html)

```
<html>
<head>
<link rel="stylesheet" href="sc.css" type="text/css">
<script language="javascript" type="text/javascript">
function getFields() {
   var pin, secSettings, generalSec, generalOnce, privateSec, privateOnce
   parent.toolbar.document.smartID.beginConversation();
   document.userform.username.value=parent.toolbar.document.smartID.getField("username");
   document.userform.password.value=parent.toolbar.document.smartID.getField("password");
   document.userform.forename.value=parent.toolbar.document.smartID.getField("forename");
   document.userform.surname.value=parent.toolbar.document.smartID.getField("surname");
   document.userform.address.value=parent.toolbar.document.smartID.getField("streetaddress");
   document.userform.town.value=parent.toolbar.document.smartID.getField("town");
   document.userform.county.value=parent.toolbar.document.smartID.getField("county");
   document.userform.postcode.value=parent.toolbar.document.smartID.getField("postcode");
   document.userform.ccNumber.value=parent.toolbar.document.smartID.getField("ccNumber");
   secSettings="x" + parent.toolbar.document.smartID.getSecurity();
   parent.toolbar.document.smartID.endConversation();
   privateSec=secSettings.charAt(1);
   privateOnce=secSettings.charAt(2);
   generalSec=secSettings.charAt(3);
   generalOnce=secSettings.charAt(4);
   document.userform.privatePin[0].checked = ((privateSec=="1") && (privateOnce=="0"));
   document.userform.privatePin[1].checked = (privateOnce=="1");
   document.userform.privatePin[2].checked = (privateSec=="0");
   document.userform.generalPin[0].checked = ((generalSec=="1") && (generalOnce=="0"));
   document.userform.generalPin[1].checked = (generalOnce=="1");
   document.userform.generalPin[2].checked = (generalSec=="0");
}
function setFields() {
   var pin, generalSec, generalOnce, privateSec, privateOnce
   generalSec = !(document.userform.generalPin[2].checked);
   generalOnce = document.userform.generalPin[1].checked;
   privateSec = !(document.userform.privatePin[2].checked);
   privateOnce = document.userform.privatePin[1].checked;
   parent.toolbar.document.smartID.beginConversation();
   parent.toolbar.document.smartID.setField("username",document.userform.username.value);
   parent.toolbar.document.smartID.setField("password",document.userform.password.value);
   parent.toolbar.document.smartID.setField("forename",document.userform.forename.value);
   parent.toolbar.document.smartID.setField("surname",document.userform.surname.value);
   parent.toolbar.document.smartID.setField("streetaddress",document.userform.address.value);
   parent.toolbar.document.smartID.setField("town",document.userform.town.value);
   parent.toolbar.document.smartID.setField("county",document.userform.county.value);
   parent.toolbar.document.smartID.setField("postcode",document.userform.postcode.value);
   parent.toolbar.document.smartID.setField("ccNumber",document.userform.ccNumber.value);
   parent.toolbar.document.smartID.setSecurity(privateSec, privateOnce, generalSec, generalOnce);
   parent.toolbar.document.smartID.endConversation();
}
</script>
</head>
<body>
<form name="userform">
<h1>Configure SmartID</h1>
<center>
<table border="0" cellpadding="0" cellspacing="10">
<tr><td align="right">Forename</td><td><input type="text" name="forename" size="30"></td></tr>
<tr><td align="right">Surname</td><td><input type="text" name="surname" size="30"></td></tr>
<tr><td align="right">Street Address</td><td><input type="text" name="address"
size="30"></td></tr>
<tr><td align="right">Town/City</td><td><input type="text" name="town" size="30"></td></tr>
<tr><td align="right">County</td><td><input type="text" name="county" size="30"></td></tr>
<tr><td align="right">Postcode</td><td><input type="text" name="postcode" size="30"></td></tr>
<tr><td> <br></td></tr>
<tr><td align="right">Username</td><td><input type="text" name="username" size="30"></td></tr>
<tr><td align="right">Password</td><td><input type="password" name="password"
size="30"></td></tr>
<tr><td> <br></td></tr>
```

```
<tr><td align="right">Credit card number</td><td><input type="text" name="ccNumber"
size="30"></td></tr>
<tr><td> <br></td></tr>
<tr><td align="right">Request PIN for general data</td><td>
<input type="radio" name="generalPin">Always
<input type="radio" name="generalPin">Once
<input type="radio" name="generalPin" checked>Never
</td></tr>
<tr><td align="right">Request PIN for sensitive data</td><td>
<input type="radio" name="privatePin">Always
<input type="radio" name="privatePin" checked>Once
<input type="radio" name="privatePin">Never
</td></tr>
<tr><td> <br></td></tr>
<tr><td colspan="2" align="right">
<input type="button" value="Retrieve current settings" onClick="getFields()">
<input type="button" value="Update configuration" onClick="setFields()">
</table>
</center>
</body>
</html>
```

Appendix 6.1.2: Client-side validation of forms (setpin.html)

```
<html>
<head>
<link rel="stylesheet" href="sc.css" type="text/css">
<script language="javascript" type="text/javascript">
function changePin() {
   var pin, secSettings, generalSec, generalOnce, privateSec, privateOnce
   if (document.userform.newPin.value==document.userform.newPin2.value) {
      parent.toolbar.document.smartID.sendPin(document.userform.currentPin.value);
      parent.toolbar.document.smartID.setPin(document.userform.newPin.value);
      document.userform.currentPin.value="";
      document.userform.newPin.value="";
      document.userform.newPin2.value="";
   }
   else {
      alert("Entries for new PIN do not match.");
   }
}
</script>
</head>
<body>
<form name="userform">
<h1>Change PIN</h1>
<p>Please note that SmartWallet and SmartID use the same PIN.</p>
<center>
<table border="0" cellpadding="0" cellspacing="10">
<tr><td align="right">Current PIN</td><td><input type="password" name="currentPin"
size="4"></td></tr>
<tr><td align="right">New PIN</td><td><input type="password" name="newPin"
size="4"></td></tr>
<tr><td align="right">Confirm new PIN</td><td><input type="password" name="newPin2"
size="4"></td></tr>
<tr><td> <br></td></tr>
<tr><td colspan="2" align="right">
<input type="button" value="Change PIN" onClick="changePin()">
</table>
</center>
</body>
</html>
```

Appendix 6.2: CSS Example (aston.css)

```
BODY
	{
	font-family: arial, helvetica, homerton, sans-serif;
	color: #000066;
	background: #FFFFFF;
	background-image: url("img/cornertriangle.gif");
	background-repeat: no-repeat;
	}

TD
	{
	font-family: arial, helvetica, homerton, sans-serif;
	color: #000066;
	}

H1,H2,H3,H4,H5,H6
	{
	font-family: stonesans, arial, helvetica, homerton, sans-serif;
	}

A
	{
	}

A:hover,A:active
	{
	color: #000066;
	}
```

Appendix 6.3: Client-Side Java Applet

Appendix 6.3.1: SmartID class

```
import java.awt.*;
import java.io.*;
import java.math.*;
import java.applet.Applet;
import netscape.javascript.*;
import opencard.core.terminal.*;
import opencard.core.service.*;
import com.ibm.opencard.terminal.pcsc10.*;
import opencard.opt.util.*;
import opencard.core.OpenCardRuntimeException;
import opencard.core.util.OpenCardPropertyLoadingException;
import opencard.core.terminal.CardTerminalException;
import opencard.core.service.CardServiceException;


public class SmartID extends Applet {

   final int maxFieldSize = 30;
   final int unknownPin = -1;
   final int wrongPin = -2;

   //APDU's for transmission to the card
   final byte selectCommand[] = {(byte)0x00,(byte)0xA4,(byte)0x04,(byte)0x0C,(byte)0x04,
                                 (byte)0xF2,(byte)0x02,(byte)0x00,(byte)0x0A};
   final byte sendPinCommand[] = {(byte)0x00,(byte)0x40,(byte)0x00,(byte)0x00,(byte)0x02};
   final byte setPinCommand[] = {(byte)0x00,(byte)0x41,(byte)0x00,(byte)0x00,(byte)0x02};
   final byte unblockCommand[] = {(byte)0x00,(byte)0x44,(byte)0x00,(byte)0x00,(byte)0x08};
   final byte getBalanceCommand[] = {(byte)0x00,(byte)0x30,(byte)0x00,(byte)0x00,(byte)0x02};
   final byte debitCommand[] = {(byte)0x00,(byte)0x20,(byte)0x00,(byte)0x00,(byte)0x08};
   final byte debitLe[] = {(byte)0x08};
   final byte repeatDebitResponseCommand[] = {(byte)0x00,(byte)0x23,(byte)0x00,(byte)0x00,
                                              (byte)0x08};
   final byte prepareCreditCommand[] = {(byte)0x00,(byte)0x21,(byte)0x00,(byte)0x00,(byte)0x02};
   final byte prepareCreditLe[] = {(byte)0x08};
   final byte creditCommand[] = {(byte)0x00,(byte)0x22,(byte)0x00,(byte)0x00,(byte)0x08};
   final byte setSecurityCommand[] =  {(byte)0x00,(byte)0x42,(byte)0x00,(byte)0x00,(byte)0x04};
   final byte getSecurityCommand[] =  {(byte)0x00,(byte)0x43,(byte)0x00,(byte)0x00,(byte)0x04};
   final byte getUsernameCommand[] = {(byte)0x00,(byte)0x01,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getPasswordCommand[] = {(byte)0x00,(byte)0x02,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getForenameCommand[] = {(byte)0x00,(byte)0x03,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getSurnameCommand[] = {(byte)0x00,(byte)0x04,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getStreetAddressCommand[] = {(byte)0x00,(byte)0x05,(byte)0x00,(byte)0x00,
                                           (byte)0x1E};
   final byte getTownCommand[] = {(byte)0x00,(byte)0x06,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getCountyCommand[] = {(byte)0x00,(byte)0x07,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getPostcodeCommand[] = {(byte)0x00,(byte)0x08,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte getCcNumberCommand[] = {(byte)0x00,(byte)0x09,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setUsernameCommand[] = {(byte)0x00,(byte)0x11,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setPasswordCommand[] = {(byte)0x00,(byte)0x12,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setForenameCommand[] = {(byte)0x00,(byte)0x13,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setSurnameCommand[] = {(byte)0x00,(byte)0x14,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setStreetAddressCommand[] = {(byte)0x00,(byte)0x15,(byte)0x00,(byte)0x00,
                                           (byte)0x1E};
   final byte setTownCommand[] = {(byte)0x00,(byte)0x16,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setCountyCommand[] = {(byte)0x00,(byte)0x17,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setPostcodeCommand[] = {(byte)0x00,(byte)0x18,(byte)0x00,(byte)0x00,(byte)0x1E};
   final byte setCcNumberCommand[] = {(byte)0x00,(byte)0x19,(byte)0x00,(byte)0x00,(byte)0x1E};

   SmartCard sc;
   CardRequest cr;
   PassThruCardService serv;
   JSObject browserWindow;
   MessageFrame message;
   MessageFrame pinMessage;
   CommandAPDU selectAPDU = new CommandAPDU(50);
   CommandAPDU commAPDU = new CommandAPDU(50);
   int userPin = -1;               //variable to hold cached user PIN for conversations
   boolean inConversation = false;
   boolean doneSetup = false;
```

```
public boolean handleEvent(Event event) {
      return super.handleEvent(event);
   }


   public void init() {
      super.init();
      addNotify();
      browserWindow = JSObject.getWindow(this);
      message = new MessageFrame();
      pinMessage = new MessageFrame();
      selectAPDU.setLength(0);
      selectAPDU.append(selectCommand);

//    Nestcape specific commands, to allow operation with Netscape security model
//    opencard.core.util.SystemAccess sys = new opencard.opt.netscape.NetscapeSystemAccess();
//    opencard.core.util.SystemAccess.setSystemAccess(sys);
//    initCard();

   }


   public void destroy() {
      super.destroy();
      if (!doneSetup) {
         try {
            SmartCard.shutdown();
         }
         catch (CardTerminalException e) {
            System.out.println("CardTerminalException: " + e.getMessage());
         }
      }
   }


   public int getBalance() {
      BigInteger result = new BigInteger("0");
      commAPDU.setLength(0);
      commAPDU.append(getBalanceCommand);
      try {
         ResponseAPDU response = sendCommand(commAPDU);
         byte[] responseArray = {(byte)0x00, response.data()[0], response.data()[1]};
         result = new BigInteger(responseArray);
      }
      catch (Exception e) {
         System.out.println("Exception: "+e.getMessage());
      }
      return Integer.parseInt(result.toString());
   }


   public String getField(String fieldName) {
      String result = "";
      commAPDU.setLength(0);
      if (fieldName.equalsIgnoreCase("username")) commAPDU.append(getUsernameCommand);
      if (fieldName.equalsIgnoreCase("password")) commAPDU.append(getPasswordCommand);
      if (fieldName.equalsIgnoreCase("forename")) commAPDU.append(getForenameCommand);
      if (fieldName.equalsIgnoreCase("surname")) commAPDU.append(getSurnameCommand);
      if (fieldName.equalsIgnoreCase("streetaddress")) commAPDU.append(getStreetAddressCommand);
      if (fieldName.equalsIgnoreCase("town")) commAPDU.append(getTownCommand);
      if (fieldName.equalsIgnoreCase("county")) commAPDU.append(getCountyCommand);
      if (fieldName.equalsIgnoreCase("postcode")) commAPDU.append(getPostcodeCommand);
      if (fieldName.equalsIgnoreCase("ccnumber")) commAPDU.append(getCcNumberCommand);
      message.setMessage("Reading from smartcard...");
      message.show();
      try {
         ResponseAPDU response = sendCommand(commAPDU);
         result = new String(response.data());
      }
      catch (Exception e) {
         System.out.println("Exception: "+e.getMessage());
      }
      if (!inConversation)
         message.hide();
      return result.trim();
   }
```

```
public void setField(String fieldName, String fieldValue) {
   StringBuffer actualValue = new StringBuffer(fieldValue);
   commAPDU.setLength(0);
   if (fieldName.equalsIgnoreCase("username")) commAPDU.append(setUsernameCommand);
   if (fieldName.equalsIgnoreCase("password")) commAPDU.append(setPasswordCommand);
   if (fieldName.equalsIgnoreCase("forename")) commAPDU.append(setForenameCommand);
   if (fieldName.equalsIgnoreCase("surname")) commAPDU.append(setSurnameCommand);
   if (fieldName.equalsIgnoreCase("streetaddress")) commAPDU.append(setStreetAddressCommand);
   if (fieldName.equalsIgnoreCase("town")) commAPDU.append(setTownCommand);
   if (fieldName.equalsIgnoreCase("county")) commAPDU.append(setCountyCommand);
   if (fieldName.equalsIgnoreCase("postcode")) commAPDU.append(setPostcodeCommand);
   if (fieldName.equalsIgnoreCase("ccnumber")) commAPDU.append(setCcNumberCommand);
   while (actualValue.length() < maxFieldSize) {
      actualValue.append(' ');   //Pad field entry with spaces up to maximum size
   }

   commAPDU.append(actualValue.toString().substring(0,(maxFieldSize)).getBytes());
      //Use substring to truncate entries over maximum size

   message.setMessage("Writing to smartcard...");
   message.show();
   try {
      ResponseAPDU response = sendCommand(commAPDU);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   if (!inConversation)
      message.hide();
}



public String debit(int value, String challenge) {
   BigInteger num256 = new BigInteger("256");
   BigInteger result = new BigInteger("0");
   BigInteger challengeNum = new BigInteger(challenge);
   byte txBlock[] = new byte[8];
   txBlock[7] = (byte)(value%256);  //Load value into
   txBlock[6] = (byte)(value/256);  //two separate bytes for tx to card
   for (int i=5; i>=0; i--) {       //Load challenge into six separate bytes for tx to card
      System.out.println(challengeNum.remainder(num256).toString());
      txBlock[i] = (byte) challengeNum.remainder(num256).intValue();
      challengeNum = challengeNum.divide(num256);
   }
   commAPDU.setLength(0);
   commAPDU.append(debitCommand);
   commAPDU.append(txBlock);
   commAPDU.append(debitLe);
   message.setMessage("Debiting smartcard...");
   message.show();
   try {
      ResponseAPDU response = sendCommand(commAPDU);
         //Returns cryptographic response received from card

      result = new BigInteger(response.data());
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   if (!inConversation)
      message.hide();
   return result.toString();
}
```

```
public String repeatDebitResponse() {
   BigInteger result = new BigInteger("0");
   commAPDU.setLength(0);
   commAPDU.append(repeatDebitResponseCommand);
   try {
      ResponseAPDU response = sendCommand(commAPDU);
         //Returns cryptograpic response received from card

      result = new BigInteger(response.data());
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   return result.toString();
}



public String prepareCredit(int value) {
   BigInteger result = new BigInteger("0");
   byte txBlock[] = new byte[2];
   txBlock[1] = (byte)(value%256);  //Loads value into
   txBlock[0] = (byte)(value/256);  //two separate bytes for tx to card
   commAPDU.setLength(0);
   commAPDU.append(prepareCreditCommand);
   commAPDU.append(txBlock);
   commAPDU.append(prepareCreditLe);
   message.setMessage("Preparing for credit...");
   message.show();
   try {
      ResponseAPDU response = sendCommand(commAPDU);
         //Returns cryptographic challenge received from card

      byte[] responseArray = {(byte)0x00, response.data()[0], response.data()[1],
                                          response.data()[2], response.data()[3],
                                          response.data()[4], response.data()[5],
                                          response.data()[6], response.data()[7]};
         //Prefixes number with zero byte before creating BigInteger to prevent
         //negative numbers from being created
         //(BigInteger uses two's-complement representation)

      result = new BigInteger(responseArray);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   if (!inConversation)
      message.hide();
   return result.toString();
}



public boolean credit(String cResponse) {
   boolean result = true;
   BigInteger num256 = new BigInteger("256");
   BigInteger cResponseNum = new BigInteger(cResponse);
   byte txBlock[] = new byte[8];
   for (int i=7; i>=0; i--) {    //Load response into eight separate bytes for tx to card
      txBlock[i] = (byte) cResponseNum.remainder(num256).intValue();
      cResponseNum = cResponseNum.divide(num256);
   }
   commAPDU.setLength(0);
   commAPDU.append(creditCommand);
   commAPDU.append(txBlock);
   message.setMessage("Crediting smartcard...");
   message.show();
   try {
      ResponseAPDU response = sendCommand(commAPDU);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
      result = false;            //Return false if credit unsuccessful
   }
   if (!inConversation)
      message.hide();
   return result;
}
```

```
public void sendPin(int pin) throws Exception {
   CommandAPDU sendPinAPDU = new CommandAPDU(50);
   sendPinAPDU.setLength(0);
   sendPinAPDU.append(sendPinCommand);
   sendPinAPDU.append((byte)(pin/256));   //Loads number into
   sendPinAPDU.append((byte)(pin%256));   //two separate bytes for tx to card
   initCard();
   System.out.println("> "+sendPinAPDU.toString());
   ResponseAPDU result = serv.sendCommandAPDU(sendPinAPDU);
   System.out.println("< "+result.toString());
   if (result.sw1()==(byte)0x69 && result.sw2()==(byte)0x82) {
      //PIN is incorrect (6982 response = Security status not satisfied)
      if (inConversation)
         userPin=wrongPin;               //Set cached PIN to error state
      pinMessage.setMessage("Incorrect PIN");
      pinMessage.show();
      throw(new Exception("Wrong PIN"));
   }
}


public void setPin(int pin) {
   message.setMessage("Setting PIN...");
   message.show();
   commAPDU.setLength(0);
   commAPDU.append(setPinCommand);
   commAPDU.append((byte)(pin/256));   //Loads number into
   commAPDU.append((byte)(pin%256));   //two separate bytes for tx to card
   try {
      sendCommand(commAPDU);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   if (!inConversation)
      message.hide();
}


public void setSecurity(boolean privateSec, boolean privateOnce,
                        boolean generalSec, boolean generalOnce) {
   message.setMessage("Setting security levels...");
   message.show();
   commAPDU.setLength(0);
   commAPDU.append(setSecurityCommand);
   commAPDU.append((byte)(privateSec?1:0));
   commAPDU.append((byte)(privateOnce?1:0));
   commAPDU.append((byte)(generalSec?1:0));
   commAPDU.append((byte)(generalOnce?1:0));
   try {
      sendCommand(commAPDU);
      sendCommand(selectAPDU);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   if (!inConversation)
      message.hide();
}


public String getSecurity() {
   String result = "";
   commAPDU.setLength(0);
   commAPDU.append(getSecurityCommand);
   try {
      ResponseAPDU response = sendCommand(commAPDU);
      result = Integer.toString((int)response.data()[0]) +
               Integer.toString((int)response.data()[1]) +
               Integer.toString((int)response.data()[2]) +
               Integer.toString((int)response.data()[3]);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
   }
   return result;
}
```

```
public boolean unblock(String code) {
   boolean result = true;
   BigInteger num256 = new BigInteger("256");
   BigInteger codeNum = new BigInteger(code);
   byte txBlock[] = new byte[8];
   for (int i=7; i>=0; i--) {    //Loads code into eight separate bytes for tx to card
      txBlock[i] = (byte) codeNum.remainder(num256).intValue();
      codeNum = codeNum.divide(num256);
   }
   commAPDU.setLength(0);
   commAPDU.append(unblockCommand);
   commAPDU.append(txBlock);
   message.setMessage("Attempting to unblock...");
   message.show();
   try {
      ResponseAPDU response = sendCommand(commAPDU);
   }
   catch (Exception e) {
      System.out.println("Exception: "+e.getMessage());
      result = false;
      }
   if (!inConversation)
      message.hide();
   return result;
}



public void beginConversation() {
   inConversation = true;
   userPin = unknownPin;
   message.setMessage("Communicating with smartcard...");
   message.show();
}



public void endConversation() {
   userPin = unknownPin;
   message.hide();
   inConversation = false;
}



private void initCard() {
   if (!doneSetup) {
      try {
         message.setMessage("Please insert your smartcard...");
         message.show();
         SmartCard.start();
         cr = new CardRequest();
         cr.setWaitBehavior (CardRequest.ANYCARD);
         sc = SmartCard.waitForCard(cr);
            //Waits for a smartcard to become available on any attached device

         if (sc != null) {
            serv = (PassThruCardService) sc.getCardService(PassThruCardService.class,true);
               //Instantiates service to permit raw APDU's to be used for tx and rx

            System.out.println("> "+selectAPDU.toString());
            ResponseAPDU result = serv.sendCommandAPDU(selectAPDU);
               //Selects SmartID applet on card

            System.out.println("< "+result.toString());
            doneSetup = true;
         } else {
            SmartCard.shutdown();
         }
      }
      catch (OpenCardPropertyLoadingException e) {
         System.out.println("OpenCardPropertyLoadingException: " + e.getMessage());
      }
      catch (ClassNotFoundException e) {
         System.out.println("ClassNotFoundException: " + e.getMessage());
      }
      catch (CardTerminalException e) {
         System.out.println("CardTerminalException: " + e.getMessage());
```

```
      }
      catch (CardServiceException e) {
         System.out.println("CardServiceException: " + e.getMessage());
      }
      message.hide();
   }
}



private ResponseAPDU sendCommand(CommandAPDU commAPDU) throws Exception {
   ResponseAPDU result = new ResponseAPDU(50);
   initCard();
   try {
      System.out.println("> "+commAPDU.toString());
         //Shows data tx in Java console for debugging
      result = serv.sendCommandAPDU(commAPDU);
         //Sends APDU to card via PassThruCardService
      System.out.println("< "+result.toString());
         //Shows data rx in Java console for debugging

      if (result.sw1()==(byte)0x6D && result.sw2()==(byte)0x00) {
         //'Wrong instruction code response' suggests SmartID applet has become deselected
         System.out.println("> "+selectAPDU.toString());
         result = serv.sendCommandAPDU(selectAPDU);
            //Selects SmartID applet on card
         System.out.println("> "+commAPDU.toString());
         result = serv.sendCommandAPDU(commAPDU);
            //Repeats the command previously attempted
      }

      if (result.sw1()==(byte)0x69 && result.sw2()==(byte)0x82) {
         //'Security status not satisfied' response i.e. card requires PIN to procede

         if (inConversation && userPin!=unknownPin && userPin!=wrongPin)
            sendPin(userPin);
               //Sends the cached PIN if in conversation

         if (!inConversation || userPin==unknownPin)
            sendPin(getPin());
               //Obtains PIN from user and if not cached and sends to card

         if (!(inConversation && userPin==wrongPin)) {
            System.out.println("> "+commAPDU.toString());
            result = serv.sendCommandAPDU(commAPDU);
               //Repeats the command previously attempted

            System.out.println("< "+result.toString());
         }
      }

      if (result.sw1()==(byte)0x69 && result.sw2()==(byte)0x83) {
         //'Authentication method blocked' i.e. card is blocked
         pinMessage.setMessage("Card blocked");
         pinMessage.show();
         throw(new Exception("Card blocked"));
      }
      if (result.sw1()==(byte)0x9D && result.sw2()==(byte)0x1A) {
         //'Invalid signature' i.e. incorrect response received by card for credit request
         pinMessage.setMessage("Corrupt server response");
         pinMessage.show();
         throw(new Exception("Invalid signature"));
      }
   }
   catch (CardTerminalException e) {
      System.out.println("CardTerminalException: " + e.getMessage());
      throw e;
   }
   catch (Exception e) {throw e;}
   return result;
}
```

```
   private int getPin() {
      int result;
      PinRequest pinWindow = new PinRequest();
      pinWindow.show();
      while (pinWindow.getPin()==0) {
         //Wait for PIN to be entered in window
      }
      result=pinWindow.getPin();
      pinWindow.dispose();
      if (inConversation)
         userPin = result; //Cache the PIN if in conversation
      return result;
   }

}
```

Appendix 6.3.2: MessageFrame class

```
import java.awt.*;
import java.io.*;

public class MessageFrame extends Frame {

   Label message;

   public boolean handleEvent(Event ev) {
      if (ev.id==Event.WINDOW_DESTROY) {
         this.hide();
         return true;
      }
      return super.handleEvent(ev);
   }

   public void setMessage(String text) {
      message.setText(text);
   }

   public MessageFrame() {
      GridLayout layout = new GridLayout(1,1);
      setLayout(layout);
      message = new Label();
      message.setFont(new Font("SansSerif",Font.ITALIC,20));
      message.setAlignment(Label.CENTER);
      message.setSize(350,80);
      add(message);
      setTitle("SmartID");
      setSize((insets().left+insets().right+350),(insets().top+insets().bottom+100));
      setLocation(100,100);
   }
}
```

Appendix 6.3.3: PinRequest Class

```
import java.awt.*;
import java.io.*;

public class PinRequest extends Frame {

   Label enterPinLabel;
   TextField userPin;
   Button okButton;
   int userPinNum = 0;

   public boolean handleEvent(Event ev) {
      if (ev.target==okButton && ev.id==Event.ACTION_EVENT) {
         //OK button clicked
         finish();
         return true;
      }
      if (ev.id==Event.WINDOW_DESTROY ) {
         //Window closed
         finish();
         return true;
      }
      return super.handleEvent(ev);
   }

   public boolean keyDown(Event ev, int key) {
      if ((key==10)||(key==13)) {
         //Enter or Return pressed
         finish();
         return true;
      }
      else
         return false;
   }

   public int getPin() {
      return userPinNum;
   }

   private void finish() {
      userPinNum=(new Integer(userPin.getText())).intValue();
         //Sets variable containing PIN as entered by user (read by parent)
      this.hide();
   }

   public PinRequest() {
      GridBagLayout layout = new GridBagLayout();
      GridBagConstraints constraints = new GridBagConstraints();
      setLayout(layout);
      constraints.gridx=0;
      constraints.gridy=0;
      enterPinLabel = new Label("Please enter PIN:");
      layout.setConstraints(enterPinLabel,constraints);
      add(enterPinLabel);
      userPin = new TextField("",4);
      constraints.gridx=GridBagConstraints.RELATIVE;
      layout.setConstraints(userPin,constraints);
      add(userPin);
      okButton = new Button("OK");
      constraints.gridx=GridBagConstraints.RELATIVE;
      layout.setConstraints(okButton,constraints);
      add(okButton);

      setTitle("Please enter PIN:");
      setSize((insets().left+insets().right+250),(insets().top+insets().bottom+100));
      setLocation(150,150);
   }
}
```

Appendix 7: Smartcard Code

```
constant
   RETURNUSERNAMEINS = 0x01        ;Retrieve profile instructions
   RETURNPASSWORDINS = 0x02
   RETURNFORENAMEINS = 0x03
   RETURNSURNAMEINS = 0x04
   RETURNSTREETADDRESSINS = 0x05
   RETURNTOWNINS = 0x06
   RETURNCOUNTYINS = 0x07
   RETURNPOSTCODEINS = 0x08
   RETURNCCNUMBERINS = 0x09

   SETUSERNAMEINS = 0x11           ;Set profile instructions
   SETPASSWORDINS = 0x12
   SETFORENAMEINS = 0x13
   SETSURNAMEINS = 0x14
   SETSTREETADDRESSINS = 0x15
   SETTOWNINS = 0x16
   SETCOUNTYINS = 0x17
   SETPOSTCODEINS = 0x18
   SETCCNUMBERINS = 0x19

   DEBITINS = 0x20                 ;Credit and debit instructions
   PREPARECREDITINS = 0x21
   ATTEMPTCREDITINS = 0x22
   REPEATDEBITRESPONSEINS = 0x23

   RETURNBALANCEINS = 0x30         ;Balance instructions

   RECEIVEPININS = 0x40            ;Security instructions
   SETPININS = 0x41
   SETSECURITYINS = 0x42
   GETSECURITYINS = 0x43
   UNBLOCKINS = 0x44

   MAXATTEMPTS = 3                 ;Maximum number of PIN attempts before blocking


static
   balance:        word #0x0000    ;Balance initially zero
   pin:            word #0x04D2    ;PIN initially set to 1234 (in decimal)
   attempts:       byte 0          ;Number of incorrect PIN attempts
   privateSec:     byte 1          ;Require pin for private data (1=true, 0=false)
   privateOnce:    byte 0          ; --once only (1=true, 0=false)
   generalSec:     byte 0          ;Require pin for general data (1=true, 0=false)
   generalOnce:    byte 0          ; --once only (1=true, 0=false)
   expResponse:    byte[8]         ;Expected response from last challenge
   lastResponse:   byte[8]         ;Last response given to credit challenge
   creditValue:    word            ;Top-up value for which last credit challenge created

   privateKey:     byte[8] 0x12,0x34,0x56,0x78,0x90,0xAB,0xCD,0xEF
   unblockCode:    byte[8] 0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88
   username:       byte[30] "None set                      "
   password:       byte[30] "None set                      "
   forename:       byte[30] "None set                      "
   surname:        byte[30] "None set                      "
   streetAddress:  byte[30] "None set                      "
   town:           byte[30] "None set                      "
   county:         byte[30] "None set                      "
   postcode:       byte[30] "None set                      "
   ccNumber:       byte[30] "None set                      "


session                    ;All automatically set to zero on reset
   criticalAccess: byte    ;Access control for critical functions (1=granted, 0=denied)
   privateAccess:  byte    ;Access control for private data (1=granted, 0=denied)
   generalAccess:  byte    ;Access control for general data (1=granted, 0=denied)
   fieldName:      word
```

```
code                                    ;Point at which execution starts on receipt of APDU

   cmpb           apduIns, UNBLOCKINS     ;Check INS byte for Unblock instruction
   jumpeq         unblockCard             ;Jump to relevant code if received

   cmpb           attempts, MAXATTEMPTS   ;Check if PIN has been entered wrong too many times
   jumpge         blocked                 ;Bypass remaining code if true

   cmpb           apduIns, RECEIVEPININS          ;Ops with no access restrictions
   jumpeq         receivePin
   cmpb           apduIns, PREPARECREDITINS
   jumpeq         prepareCredit
   cmpb           apduIns, ATTEMPTCREDITINS
   jumpeq         attemptCredit
   cmpb           apduIns, REPEATDEBITRESPONSEINS
   jumpeq         repeatDebitResponse

   loadn          1, generalSec                   ;Check if general
   cmpn           1, generalAccess                ; security status satisfied
   jumplt         notAllowed                      ; Bypass remaining code if false

   cmpb           apduIns, RETURNBALANCEINS       ;Ops with general access restrictions
   jumpeq         returnBalance
   loada          username
   cmpb           apduIns, RETURNUSERNAMEINS
   jumpeq         returnField
   cmpb           apduIns, SETUSERNAMEINS
   jumpeq         setField
   popw
   loada          forename
   cmpb           apduIns, RETURNFORENAMEINS
   jumpeq         returnField
   cmpb           apduIns, SETFORENAMEINS
   jumpeq         setField
   popw
   loada          surname
   cmpb           apduIns, RETURNSURNAMEINS
   jumpeq         returnField
   cmpb           apduIns, SETSURNAMEINS
   jumpeq         setField
   popw
   loada          streetAddress
   cmpb           apduIns, RETURNSTREETADDRESSINS
   jumpeq         returnField
   cmpb           apduIns, SETSTREETADDRESSINS
   jumpeq         setField
   popw
   loada          town
   cmpb           apduIns, RETURNTOWNINS
   jumpeq         returnField
   cmpb           apduIns, SETTOWNINS
   jumpeq         setField
   popw
   loada          county
   cmpb           apduIns, RETURNCOUNTYINS
   jumpeq         returnField
   cmpb           apduIns, SETCOUNTYINS
   jumpeq         setField
   popw
   loada          postcode
   cmpb           apduIns, RETURNPOSTCODEINS
   jumpeq         returnField
   cmpb           apduIns, SETPOSTCODEINS
   jumpeq         setField
   popw
   cmpb           apduIns, GETSECURITYINS
   jumpeq         getSecurity

   loadn          1, privateSec                   ;Check if private
   cmpn           1, privateAccess                ; security status satisfied
   jumplt         notAllowed                      ; Bypass remaining code if false

   loada          password                        ;Ops with private access restrictions
   cmpb           apduIns, RETURNPASSWORDINS
   jumpeq         returnField
   cmpb           apduIns, SETPASSWORDINS
   jumpeq         setField
   popw
```

```
   loada           ccNumber
   cmpb            apduIns, RETURNCCNUMBERINS
   jumpeq          returnField
   cmpb            apduIns, SETCCNUMBERINS
   jumpeq          setField
   popw

   cmpb            criticalAccess, 1               ;Check if critical security status satisfied
   jumplt          notAllowed                      ;Bypass remaining code if false

   cmpb            apduIns, DEBITINS               ;Ops with critical access restrictions
   jumpeq          debitCard
   cmpb            apduIns, SETPININS
   jumpeq          setPin
   cmpb            apduIns, SETSECURITYINS
   jumpeq          setSecurity

   popw
   pushb           1               ;Command has been Case 1 (No command or response data present)
   CheckCase                       ;Inform card OS of case before returning
   exitstat        #0x6D00         ;Return with Wrong instruction code


notAllowed:
   pushb           1
   CheckCase
   exitstat        #0x6982         ;Return with Security status not satisfied


blocked:
   pushb           1
   CheckCase
   exitstat        #0x6983         ;Return with Authentication method blocked



receivePin:
   pushb           3
   CheckCase
   incn            1, attempts     ;Increment before comparison to prevent timely card removal
   loadn           2, apduBody
   cmpn            2, pin
   braeq           correctPin
   exitstat        #0x6982;        ;Return with Security status not satisfied
correctPin:
   setb            attempts, 0
   setb            generalAccess, 1
   setb            privateAccess, 1
   setb            criticalAccess, 1
   exitstat        #0x9000;



setPin:
   call            resetAccess
   pushb           3
   CheckCase
   loadn           2, apduBody     ;Input: PIN{2}
   storen          2, pin
   exitstat        #0x9000



setSecurity:
   call            resetAccess
   pushb           3
   CheckCase
   loadn           4, apduBody     ;Input: Security settings{4}
   storen          1, generalOnce
   storen          1, generalSec
   storen          1, privateOnce
   storen          1, privateSec
   exitstat        #0x9000
```

```
getSecurity:
   call            resetAccess
   pushb           2
   CheckCase
   loadn           1, privateSec
   loadn           1, privateOnce
   loadn           1, generalSec
   loadn           1, generalOnce
   storen          4, apduBody     ;Output: Security settings{4}
   exitstatla      #0x9000, 4


unblockCard:
   pushb           3
   CheckCase
   loadn           8, apduBody
   cmpn            8, unblockCode  ;Input: Unblock code{2}
   braeq           correctUnblock
   exitstat        #0x6983         ;Return with Authentication method blocked
correctUnblock:
   setb            attempts, 0
   exitstat        #0x9000


returnBalance:
   call            resetAccess
   pushb           2
   CheckCase
   loadn           2, balance
   storen          2, apduBody     ;Output: Balance{2}
   exitstatla      #0x9000, 2


returnField:
   call            resetAccess
   pushb           2
   CheckCase
   loadin          30              ;Load data onto stack from location specified at top of stack
   storen          30, apduBody    ;Output: Field data{30}
   exitstatla      #0x9000, 30


setField:
   cmpb            criticalAccess, 1
   jumplt          notAllowed
   call            resetAccess
   pushb           3
   CheckCase
   loadn           30, apduBody    ;Input: Field data {30}
   storein         30              ;Store data in location specified at top of stack
   exitstat        #0x9000


debitCard:
   call            resetAccess
   pushb           4
   CheckCase
   loadn           8, apduBody     ;Input: Challenge{6} Debit value{2}
   cmpn            2, balance      ;Ensure balance is greater than debit value
   brage           performDebit
   pushb           2
   CheckCase
   exitstat        #0x6400         ;Return with Process aborted - memory unchanged
performDebit:
   CommitThenProtect               ;Start transaction protection
   subn            2, balance      ;Subtract debit value from balance
   loadn           8, privateKey   ;Generate response
   xorn            8               ; by XORing private key with (challenge,debit value)
   popn            8               ;Remove private key from stack to access result
   storen          8, lastResponse ;Store response
   Commit                          ;Commit transaction
   loadn           8, lastResponse ;Output: Response{8}
   storen          8, apduBody
   exitstatla      #0x9000, 8
```

```
repeatDebitResponse:
   pushb           2
   CheckCase
   loadn           8, lastResponse
   storen          8, apduBody     ;Output: Response{8}
   exitstatla      #0x9000, 8



prepareCredit:
   pushb           4
   CheckCase
   GetRandomNumber                 ;Place 8-byte random number at top of stack
   loadn           2, apduBody     ;Input: Credit value{2} (add to top of stack)
                                   ;Challenge is now top 8 bytes of stack (random#,credit value)
   storen          8, expResponse  ;Store challenge in expResponse temporarily
   loadn           8, expResponse
   loadn           8, privateKey   ;Calculate expected response
   xorn            8, expResponse  ; by XORing private key with challenge
   popn            8               ;Remove private key from stack to access challenge
   storen          2, creditValue  ;Store credit value for last challenge created
   loadn           2, creditValue
   storen          8, apduBody     ;Output: Challenge{8}
   exitstatla      #0x9000, 8



attemptCredit:
   pushb           3
   CheckCase
   pushw           #0000           ;Check if credit value
   cmpn            2, creditValue  ; is zero
   braeq           creditFailed    ;If true, no challenge has yet been generated
   loadn           8, apduBody     ;Input: Response{8}
   cmpn            8, expResponse  ;Check if response is correct
   braeq           performCredit
creditFailed:
   exitstat        #0x9D1A         ;Return with Invalid signature
performCredit:
   loadn           2, creditValue
   CommitThenProtect               ;Start transaction protection
   addn            2, balance      ;Add credit value to balance
   pushw           #0000           ;Set stored credit value
   storen          2, creditValue  ; to zero (preventing replay)
   Commit
   exitstat        #0x9000



resetAccess:
   setb            criticalAccess, 0       ;Always reset critical access to denied after an op
   cmpb            privateOnce, 1          ;Check if OK to enter pin for private data once only
   braeq           dontResetPrivate        ;  --Don't reset private access if true
   setb            privateAccess, 0        ;  --Do reset private access if false
dontResetPrivate:
   cmpb            generalOnce, 0          ;Check if OK to enter pin for general data once only
   braeq           dontResetGeneral        ;  --Don't reset general access if true
   setb            generalAccess, 0        ;  --Do reset general access if false
dontResetGeneral:
   return



end
```