Regular Paper

A Framework for Distributed Inter-smartcard Communication

Masayuki Terada,[†] Kensaku Mori,[†] Kazuhiko Ishii,[†]
Sadayuki Hongo,[†] Tomonori Usaka,[††] Noboru Koshizuka[††]
and Ken Sakamura[††]

This paper proposes a framework based on a new architecture that allows distributed smartcards to interact with one another as well as with application programs on their hosts. Since these interactions are handled distribution-transparently through message dispatching agents deployed on each host, the smartcards can autonomously conduct distributed protocols without turning to off-card application programs. The proposed framework thus reduces the complexity of application programs and makes it easier to develop smartcard-based services that offer a high level of functionality. The feasibility of the framework is evaluated and confirmed by implementing a smartcard-based optimistic fair trading protocol for electronic vouchers on this framework.

1. Introduction

Personal trusted devices, of which smartcards are the most typical example, are becoming the preferred tools for realizing secure and convenient electronic commerce.

The early smartcards had limited computing power and were therefore used as data carriers with simple access controls. Recent smartcards, however, are far more powerful and can be treated as external secure computing devices rather than mere secure memory devices[12]. Smartcards are now being used as auxiliary devices of regular computers (i.e., PCs), as well as trusted conductors of distributed protocols in several applications, such as systems for trading electronic money and vouchers among smartcards[7],[17],[18]. Smartcards are no longer auxiliary devices in these systems, but rather their hosts can be considered as auxiliary devices which supply I/O functionality to the smartcards.

One impediment is that the standard interface and architecture for interacting with smartcards, namely ISO7816-4, is rather outdated and does not support modern smartcard functions such as conducting distributed protocols. In ISO7816-4, the data format named APDU (Application Protocol Data Unit) depends too much on the original internal smartcard structure, interactions are asymmetric, and smartcards must always be "passive" transponders; there is no way of getting the smartcard to issue a command to its host. These drawbacks make ISO7816-4 unacceptable as a platform on which to develop application programs for systems that are intended to fully utilize the latest smartcards' high functionality.

Several approaches have been proposed to offer more sophisticated and abstracted interactions with smartcards, which can be accessed in similar ways to familiar devices such as file systems, web servers, and remote objects[4],[9],[11],[13]. These approaches reduce the cost of developing application programs, since they conceal the internal details of the smartcards' data format. Unfortunately, the architectures proposed so far are still asymmetric, and smartcards are considered as merely passive transponders. Such architectures cannot facilitate the development of systems that depend on smartcards to conduct distributed protocols.

In this paper, we propose a framework named TENeT (Trusted Environment with Networking eTRON), which provides symmetric and distribution transparent interactions among smartcards and application programs in a distributed environment.

Every smartcard and application program (on hosts) in this architecture interacts with the others by distributed message passing. Since each message is delivered automatically according to its destination, a smartcard can (logically) interact with another smartcard autonomously. Since the interaction is concealed from the application program on the host, there is no need for the program to parse or mediate messages between smartcards. Instead, the ap-

† NTT DoCoMo, Inc.
†† The University of Tokyo

plication program simply sends a message that asks the smartcard to start the protocol and waits to receive the result of the protocol.

The feasibility of the framework is evaluated and confirmed by implementing a smartcard-based optimistic fair exchange protocol for trading electronic vouchers. The implementation result shows that such a protocol could be efficiently implemented on this framework without any significant performance overheads, and the execution of the protocol by the smartcards is well abstracted and encapsulated from the application programs on the hosts: all those programs have to do is to start an exchange (for the initiator of the trade) and to confirm the offer of the exchange (for the responder).

The rest of the paper is organized as follows: Section 2 describes the characteristics of smartcards and the problems of ISO7816-4, which is commonly used as the specification for interacting with smartcards. Section 3 looks at the previous approaches related to the problems of smartcards. Section 4 illustrates our proposal, TENeT, and details its structure and security mechanisms. Section 5 reviews an implementation result of a distributed protocol, which exchanges electronic vouchers between two smartcards, using the proposed framework. Section 6 discusses the security aspect of the framework.

2. Smartcards

A smartcard is a tamper-resistant device that prevents external entities from illegally accessing stored data and programs. Although some low-end cards are not programmable and merely offer simple authorization using shared secret and read/write (or increment/decrement) access to their memory, this paper assumes a programmable smartcard that has a micro-processor for executing programs.

Since a smartcard usually has no (long-life) battery or user interface, it cannot be used in a standalone manner; instead, it must be connected to another device that supplies power and I/O support. We refer to such a device as the *host* of the card. A PC that has a smartcard reader/writer and a mobile phone with a SIM or UICC slot are typical hosts.

Both a smartcard and its host have application programs that interact with each other. These application programs are called off-card (off-card APs) and on-card application programs (on-card APs), respectively.

Interactions between an on-card AP and an off-card AP usually follow a part of an international standard specification for smartcards, ISO7816 part 4[10] (ISO7816-4), which specifies a protocol and data format named Application Program Data Unit (APDU) for issuing commands from an off-card AP to an on-card AP.

A smartcard based on ISO7816-4 functions as a reactive transponder; the host sends a command (Command APDU) to the smartcard and the smartcard responds with the result of the command execution (Response APDU). The smartcard never sends any command to the host (or another smartcard) in this scheme.

This scheme works well for smartcards that provide only low-level functionalities such as read/write or simple sign/verify; however, it does not support the high-end functionalities of recent smartcards, such as conducting distributed protocols. A description of the problems with ISO7816-4 is given below:

Asymmetry of the architecture. Since ISO7816-4 assumes that smartcards will not behave proactively, it is difficult for a smartcard to send a request or notification to an external entity; for example, it is difficult to apply the Observer pattern[8], which is commonly used for sending notifications of events in object-oriented designed systems. A series of APDUs may be able to provide a roughly equivalent function; however, it makes both the on-card APs and off-card APs much more complicated.

No support for distribution. Since there is no support for a distributed environment in ISO7816-4, a smartcard cannot interact with remote entities such as smartcards on remote hosts and server programs on remote servers unless an off-card AP on its host mediates the interactions. Because of the asymmetry of the architecture, such a mediation procedure makes off-card APs cumbersome and complicated.

Figure 1 illustrates the drawbacks of ISO7816-4 style architectures when they are used to implement a distributed protocol conducted between smartcards. In this figure, smartcards merely conduct a 2-round protocol that consists of 4 messages, which looks more complicated than it is because of a number of mediations of the messages by off-card APs.

This is one of the demerits of such an architecture. Because of the asymmetry and lack of distribution support, smartcards are unable

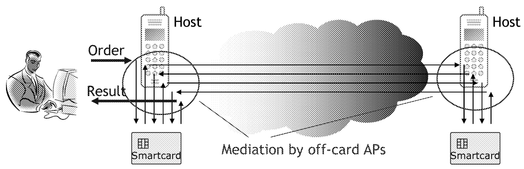

Fig. 1 Implementing an inter-smartcard communication protocol using ISO7816-4.

to conduct the protocol by themselves, and off-card APs need to mediate the interactions; for each message to be exchanged between smartcards, an off-card AP has to feed it into a smartcard as a command, extract the next message from the response, and forward it to an appropriate off-card AP on a remote host according to the extracted message. Off-card APs thus have to be responsible for conducting the protocol, even though they are intrinsically interested only in the result of the protocol and should not have to consider details of the behavior of the smartcards; for example, when smartcards conduct mutual authentication, off-card APs need to know whether the authentication was successful or not, but they may not need to know how the smartcards authenticate each other.

The implementation of such mediation processes in off-card APs is not only tedious but also complicates the structure of an off-card AP, since it compromises the principle of encapsulation. It will accordingly be fertile soil for bugs that must be avoided for security or commerce applications.

3. Previous Work

Several approaches have been proposed to make it easier to develop smartcard-based application systems.

One approach is to develop an application-oriented framework, which encapsulates interactions with smartcards from off-card APs and provides off-card APs with an abstracted view of the functions close to the application logic. For example, VTS-API (Voucher Trading System API)[16] is designed to provide "electronic wallet" off-card APs with simple and abstracted functions to manage and transfer electronic vouchers including money and tickets. Although VTS-API is well generalized to permit the transfer of diverse kinds of vouchers, it is useless for other applications, since these require another framework.

This approach might be promising in a certain application area, but it is difficult to apply it to applications that differ from the intended target of the framework. We therefore focus on generic approaches that can be applied to different kinds of applications and should be helpful when developing application-oriented frameworks.

Webcard[13] and WebSIM[9] enable a smartcard to mimic an HTTP server, with which users and off-card APs can interact using the HTTP protocol; that is to say, a user can access a smartcard by using his/her favorite web browser. This approach is quite useful in certain applications in which users directly interact with their smartcards, and the generality of HTTP does not strictly limit the application area.

JASON[4] and JavaCard 2.2[15] provide facilities similar to Remote Method Invocation (RMI), the facility for invoking methods on remote objects in Java. These facilities, which are called Secure Method Invocation in JASON and JavaCard RMI in JavaCard 2.2, enable off-card APs to interact with smartcards as if they were interacting with remote objects by means of RMI. Since the detailed format of the interactions is concealed from off-card APs, the developers do not need to be aware of the detailed format or indeed the protocol used in RMI.

These approaches enable an off-card AP to access a smartcard in an abstracted and unified way, which is likely easier than APDUs, particularly when remotely accessed. However, their architecture remains asymmetric: smartcards are still passive and are not allowed to access external entities. Such approaches therefore do not allow smartcards to communicate with other devices, including other smartcards.

To address the disadvantages caused by asymmetric interactions between a smartcard and its host, several interaction schemes have been proposed on the basis of the Intelligent Adjuncts paradigm[2),3)], which enables a smartcard to access off-card resources of the host.

Proactive UICC[1),6)] provides a mechanism whereby a smartcard can issue commands to its host to display some messages, for example, or to fetch input from a user. In the initial state, an off-card AP issues a command to a smartcard and the smartcard responds to the command, as is the case with the usual ISO7816-4 smartcards. When the smartcard wants to issue a command to the off-card AP, it sends back a response that includes a special state code ("91 XX", where XX is the size of the command to be issued by the card) instead of the normal state code ("90 00"). The off-card AP then issues a FETCH command that requests the card to send back the command to be issued. The proactive UICC mechanism thus enables smartcards to reverse the command/response direction.

Card-Centric Framework (CCF)[5] takes a similar but more aggressive approach, wherein the roles of smartcard and host are completely swapped; a smartcard becomes proactive as soon as it is attached to the host.

The disadvantage of the proactive UICC approach is that the switching of the command/response direction requires an extra handshake cycle, which causes considerable performance overheads and complicates off-card AP design. CCF eliminates these overheads by considering only the case where a smartcard makes use of resources on the host; however, this approach consumes much more processing power of the smartcard, and the implementation result[5] shows that it is not yet feasible for current smartcards.

Apart from the performance problem, neither Proactive UICC nor CCF offers distribution transparency; help of an off-card AP is needed for a smartcard to access remote resources.

4. Our Proposal

In this section, we propose a distributed smartcard communication framework named TENeT (Trusted Environment with Networking eTRON). This framework efficiently and securely provides a symmetric and distribution transparent interaction means based on message passing, which enables smartcards to autonomously conduct distributed protocols with one another, without any mediation by off-card APs.

To dispense with such extrinsic mediation by off-card APs and enable them to dedicate themselves to their primary tasks, smartcards should autonomously conduct a protocol by directly interacting with one another after receiving the order from an off-card AP (**Fig. 2**). Since a smartcard usually has no means of physical access to a network itself, however, interactions among smartcards are inevitably mediated by hosts in physical sense. TENeT logically realizes such direct inter-smartcard communication by deploying distribution-transparent messaging facilities on each host.

In this architecture, off-card APs do not have

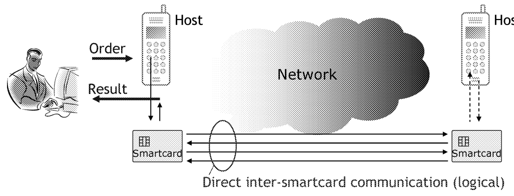

Fig. 2 Direct inter-smartcard communication.

to deal with the details of interactions among smartcards; TENeT offers much better encapsulation of smartcard behaviors and therefore simplifies the implementation of off-card APs in comparison with previous architectures.

4.1 Design Goals

In order to realize the inter-smartcard communication described above, we set the main design goals of the TENeT framework as follows:

Provide efficient symmetric interactions. While the importance of smartcard proactivity is emphasized in the above descriptions, the reactive use of smartcards such as sign/verify still remains and is important. To consistently support both uses, a symmetric architecture in which smartcards and off-card APs can interact with one another in the same manner is desirable, and should be implemented efficiently.

Provide distribution transparency. As mentioned above, distribution transparency of interaction should be provided to enable a smartcard to autonomously conduct distributed protocols with another smartcard on a remote host, without off-card APs being responsible for routing or dispatching the messages of the protocol.

Provide security. Distribution transparency suggests that it is possible for malicious remote entities to conduct illegal acts on a smartcard without alerting the off-card AP or its user. The framework must prevent such illegal acts to avoid this situation. In addition, when considering the implementation of off-card APs through the use of an on-line downloaded program execution environment such as a Java applet, which is the most popular way to implement add-on application programs on mobile phones, illegal accesses from malicious off-card APs should be prevented as well; for example, an applet should be prevented from initiating harmful operations, such as deleting or sending vouchers while pretending to merely draw an animation on a display.

To achieve these goals, we took the following approaches in designing TENeT.

The most straightforward approach to providing symmetric interactions is to realize a call/return mechanism that allows smartcards to invoke external entities and vice versa, without handshakes to switch the direction. However, this approach requires a smartcard to preserve its execution context (i.e., the execution stack) until the invocation is returned from the external entity. This is not easy for smartcards, which tend to suffer from a shortage of volatile memory (i.e., RAM), to efficiently preserve execution contexts, especially in view of the need to support nested (reentrant) invocations.

We therefore dropped the call/return approach and instead adopted the message-passing approach, in which all interactions among smartcards and external entities, including off-card APs and remote servers, are performed by exchanging messages. This requires that off-card APs be executed in an event-driven manner instead of a command/response or call/return manner, but we consider that this is not a problem, since the event-driven architecture is commonly used in object-oriented programming, especially when developing GUI applications.

To provide distribution transparency, each entity exchanging messages has its own identifier, and each message includes two identifiers: the source and the destination of the message. Messages are routed automatically according to the identifier of its destination. The details of the identifier are described in Section 4.2, and the architecture of the framework for distributing the messages is depicted in Section 4.3.

Providing security against malicious entities, including remote entities and malicious off-card APs, requires a different authentication scheme from that of the current smartcards. Most smartcards provide an "external authentication" mechanism to prevent external entities from performing possibly harmful operations. A typical external authentication offers "per smartcard" authentication; that is to say, it manages whether a particular smartcard has been authenticated. When this mechanism is applied to TENeT unaltered, any messages from any entities will be accepted by a smartcard once an off-card AP has made a successful authentication.

An approach that avoids this situation is to apply the logical channel extension in ISO7816-4. This extension enables a smartcard to virtually establish up to four channels with different off-card APs, and requires that activities on each channel be independent; an authentication made by an off-card AP using a dedicated channel does not affect the authentication status of another off-card AP using a different channel. The logical channel extension, however, cannot be easily adapted to TENeT, since its channel-based mechanism does not suit the message-passing architecture and does not offer any means to prevent an off-card AP from using a channel dedicated to another AP.

To avoid the above problems and provide enough security, TENeT provides a "per entity" authentication scheme based on capability objects and message-filtering facilities named Guard. Details are given in Section 4.4.

4.2 Message Structure

The messages exchanged among entities in TENeT, including smartcards and off-card APs, are named eTP messages.

Like IP packets on the Internet, an eTP message consists of a header, which includes its source and destination, and a payload. For easier implementation of message parsers in smartcards, the header includes a code designating "message type," which determines the format of the payload; that is to say, an eTP message is represented by the 4-tuple `(src, dst, mtype, param)`: `src` and `dst` represent the sender and destination, respectively, `mtype` represents the code designating the format of the `param`, and `param` is a set of parameters described in the format designated by `mtype`.

To realize distribution-transparent message passing, it is required that (1) identifiers of entities including smartcards and off-card APs should be unique, and (2) it should be easy to route messages according to the identifier.

In TENeT, each smartcard has a 128-bit identifier named uCode, which is a globally unique identifier assigned to smartcards and RF-ID tags by an organization named the uID Center[14].

Since it is not practical to similarly identify each off-card AP, which would require the assignment of an identifier for each AP installation, an off-card AP in TENeT acquires its identifier from a smartcard on the same host; the 16-byte (128-bit) identifier is divided into 12 + 4 bytes. The 12-byte set is called a "do-

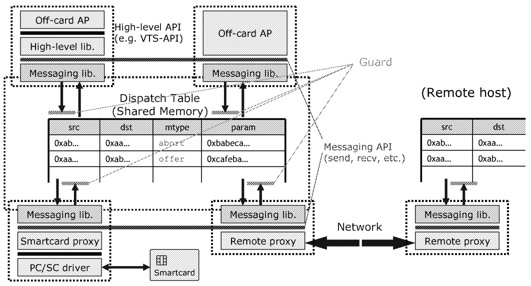

Fig. 3 Architecture of TENeT.

main" and the 4-byte set is called a "port."

Each smartcard is assigned its own domain D from the center. An off-card AP requests the smartcard on the same host to assign a port P, which is a unique number in the domain and is never reused. In the case of a successful assignment, the identifier of the application is $D|P$, where $x|y$ is the concatenation of x and y. $P = 0$ is reserved for the identifier of the smartcard itself, and is never assigned to APs; that is, the identifier of a smartcard is $D|0$ (the 12-byte domain D followed by 4 bytes of 0's).

This dynamic identifier assignment scheme efficiently ensures the uniqueness of identifiers and makes message routing easier. Since a smartcard and all off-card APs on the same host have the same domain, messages can be routed by managing domain and host network address pairs instead of managing each identifier; this reduces the cost of address resolution for message routing.

4.3 Architecture of TENeT

Figure 3 depicts the architecture and modules of the framework for symmetric and distribution transparent delivery of eTP messages. The role of each module is described below:

4.3.1 Dispatch Table

The dispatch table stores messages sent from smartcards and off-card APs until they are received by another entity. Every sent message is temporarily stored in this table. A message is sent by storing the message in the table via the messaging library, and received by reading the message.

4.3.2 Messaging Library

The messaging library provides a means of sending or receiving messages to or from off-card APs and proxies, namely, the messaging API, by providing access to the dispatch table; i.e., store a message in the table (sending a message), read a message from the table (receiving a message), and register a handler that notifies of the storage of the specified message in the table (notifying of message reception).

This library also offers an off-card AP a means of acquiring its own identifier from a smartcard. When acquiring the identifier, the library creates an object named a "capability object" that retains the acquired identifier, and passes it to the off-card AP.

From off-card APs, most of the functions provided by this library can be accessed only through a capability object in order to prevent another off-card AP from pretending. The details of the security offered by the capability object are described in Section 4.4.

4.3.3 High-level Library

To make it easier to implement off-card APs for a particular application, high-level libraries provide application-oriented APIs that consist of the functions needed to convert the data format used in off-card APs from and to that used in eTP messages. These functions are used as wrappers that construct and parse messages so as to conceal from the off-card APs the detailed data format of messages.

High-level libraries are not mandatory in TENeT, but they may be helpful to offer better encapsulation of data structure to off-card APs when implemented.

4.3.4 Smartcard Proxy

The smartcard proxy provides a means of sending and receiving messages in place of a smartcard. This proxy is an independent process that accesses the messages using the messaging library; that is, when notified of the arrival of a message for the smartcard by the handler registered to the messaging library, this proxy receives the message and forwards it to the card, and upon receiving a message from the smartcard, it stores the message in the dispatch table via the messaging library.

To detect the arrival of messages for the smartcard, the smartcard proxy registers a handler with the messaging library for any message whose destination is the smartcard.

4.3.5 Remote Proxy

The remote proxy provides a means of routing messages from/to remote entities. This proxy is also an independent process; it forwards a message to another remote proxy located on the corresponding remote host when the destination of the message is a remote entity. Upon receiving a message from a remote proxy of another host, it stores the message in the dispatch table.

This proxy manages a routing table, which includes pairs of domain and network address of the host (cf. Section 4.2) and registers a handler with the messaging library for any message whose destination matches the managed domains.

4.3.6 Guard

Guard consists of filter conditions and handlers to be invoked when a message matching the conditions arrives at the dispatch table. When the handler is invoked, it typically inspects the matched message and discards it if it is considered harmful. The behavior of the handler can be customized by the user.

Guard is mainly used to filter messages to prevent malicious remote messages from being forwarded to a smartcard. Details of this role of Guard are described in Section 4.4.

4.4 Security Mechanisms

The authentication scheme used in TENeT is similar to the "per channel" authentication mechanism of the logical channel extension mentioned in Section 4.1, but differs in that it offers secure "per entity" authentication.

In this scheme, authentications are made for each entity; even if an off-card AP is successfully authenticated on a smartcard, another entity cannot send messages that require authentication to the smartcard unless it makes its own authentication. A smartcard manages an "authenticated ID" set, which consists of identifiers of successfully authenticated off-card APs. The authenticated off-card APs whose identifiers are included in the set are assumed to be "privileged" and are granted the privilege of sending the smartcard potentially harmful messages such as deleting vouchers.

To gain privilege from a smartcard by authentication, the off-card AP has to belong to the domain of the smartcard; that is to say, only off-card APs that acquired their identifiers from the smartcard can gain privilege by successful authentication. When the authentication succeeds for an off-card AP, the smartcard adds the identifier of the AP to the set of the authenticated IDs, and messages whose sender ID (`src` field) is included in the set are considered as messages sent by the privileged entity.

To secure this scheme, it is necessary to prevent malicious entities, including malicious off-card APs and remote entities, from pretending to be privileged (correctly authenticated) off-card APs by forging the sender ID; when the sender ID can be forged, such an authentication becomes useless. TENeT utilizes message-filtering facilities, i.e., Guard, against forgeries by remote entities, while using capability objects against forgeries by malicious off-card APs.

By default, the handler of Guard adapted to the remote proxy discards any remote messages whose sender ID belongs to the domain of a smartcard (cf. Section 4.2) on the local host. Since domains are uniquely assigned for each smartcard, any entity belonging to such a domain is unlikely to exist on another host; therefore, a remote message that has the same domain as that of a local smartcard must always be considered malformed. Consequently, no remote entity can successfully send a message that pretends to have been sent by an authenticated off-card AP, whose identifier has to belong to the domain of the smartcard.

A capability object is instantiated when an off-card AP requests a smartcard to assign its identifier via the messaging library and then is passed to the off-card AP. The capability object offers the off-card AP a means of sending a message, whose destination (`dst`), message type (`mtype`), and message parameter (`param`) can be freely set by the off-card AP, while its sender ID (`src`) is bound to the identifier assigned when the object is instantiated.

Provided that this is the sole means for off-card APs to send messages to smartcards and the capability object is accessible only by the off-card AP that instantiated it, no other off-card APs can pretend to have the sender ID. The feasibility of this assumption is discussed in Section 6.

5. Implementation Results

This section describes the results of implementing a distributed protocol, which optimistically exchanges electronic vouchers between two smartcards, using the proposed framework.

5.1 Optimistic Fair Exchange Protocol for Trading Electronic Vouchers

We used this approach to implement a prototype voucher trading system that enables users to securely trade electronic vouchers stored in their smartcards, by using the optimistic fair exchange protocol proposed in Terada, et al.[17]. This protocol fairly exchanges vouchers v_1 and v_2, each of which is stored in smartcards S_A and

Table 1 Definitions of the symbols used in the protocol description.

$\mathrm{S_A}$,$\mathrm{S_B}$	smartcards located on hosts A and B, respectively
$\mathrm{P_A}$,$\mathrm{P_B}$	(off-card) wallet application programs that manage $\mathrm{S_A}$ and $\mathrm{S_B}$
$\mathrm{v_1}, \mathrm{v_2}$	vouchers initially stored in $\mathrm{S_A}$ and $\mathrm{S_B}$
v_1, v_2	identifiers of $\mathrm{v_1}$ and $\mathrm{v_2}$
n_1, n_2	random numbers generated by $\mathrm{S_A}$ and $\mathrm{S_B}$
h()	a secure hash function (e.g., SHA1)
PkX	a public key of asymmetric cryptography
$(m)_{\mathrm{Pk}X}$	a signed message that consists of m and a signature verifiable by PkX
Cert_X	a public key certificate of PkX

$\mathrm{S_B}$, as follows[☆], where **Table 1** gives the definitions of the symbols used in describing the protocol:

(1) $\mathrm{P_A}$ orders $\mathrm{S_A}$ to start the exchange of $\mathrm{v_1}$ stored in $\mathrm{S_A}$ and $\mathrm{v_2}$ stored in $\mathrm{S_B}$.

(2) $\mathrm{S_A}$ sends $\mathrm{S_B}$ an offer message m_1: $\{v_1, v_2, n_1\}$, which consists of an offer to exchange $\mathrm{v_1}$ and $\mathrm{v_2}$.

(3) $\mathrm{S_B}$ deletes $\mathrm{v_2}$ and sends $\mathrm{S_A}$ an agreement message m_2: $\{(\mathrm{h}(v_1|v_2|n_1) \mid \mathrm{h}(n_2))_{\mathrm{PkB}}, \mathrm{Cert_B}\}$ iff offer m_1 is acceptable.

(4) $\mathrm{S_A}$ deletes $\mathrm{v_1}$ and sends $\mathrm{S_B}$ a confirmation message m_3: $\{(\mathrm{h}(n_2))_{\mathrm{PkA}}, \mathrm{Cert_A}\}$ iff agreement m_2 is successfully verified.

(5) $\mathrm{S_B}$ stores $\mathrm{v_1}$ and sends $\mathrm{S_A}$ a commitment message m_4: n_2 iff confirmation m_3 is successfully verified; otherwise, $\mathrm{S_B}$ runs the abort subprotocol.

(6) $\mathrm{S_A}$ stores $\mathrm{v_2}$ [☆☆] iff commitment m_4 is successfully verified; otherwise, $\mathrm{S_A}$ runs the resolve subprotocol.

When the above protocol is interrupted due to network errors or other causes, a user may fall into an unfair condition, wherein the smartcard of the user has deleted its voucher without storing the voucher to be received. In this case, both $\mathrm{P_A}$ and $\mathrm{P_B}$ can recover fairness by performing a resolve subprotocol and an abort subprotocol, respectively, with a trusted third-party T. Refer to the appendix for details of these subprotocols.

5.2 Implementation

The protocol can be quite easily implemented by using the proposed framework; each message m_i can be 1-to-1 mapped into an eTP message. Most of these messages can be exchanged between smartcards $\mathrm{S_A}$ and $\mathrm{S_B}$ autonomously, and therefore the off-card wallet APs have almost nothing to do with running the protocol, except that (1) wallet $\mathrm{P_A}$ sends a message to start the exchange, and (2) $\mathrm{P_B}$ should confirm whether offer m_1 is acceptable or not to its user.

[☆] To simplify the explanation, the details of the verification procedure and management of n_1 and n_2 are omitted.

[☆☆] $\mathrm{v_2}$ is assumed to be generatable (i.e., able to be stored) from v_2 by $\mathrm{S_A}$.

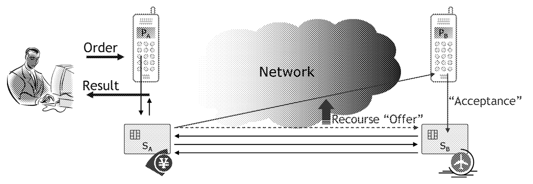

Fig. 4 Modified message flow.

The former action also can obviously be mapped to an eTP message, and the latter can be implemented in the following ways: (2a) use Guard, which prompts the user if the offer is accepted and discards the offer if the user rejects it; or (2b) change the destination of offer m_1 to $\mathrm{P_B}$ instead of $\mathrm{S_B}$, and insert an additional message m'_1, which indicates acceptance of the offer sent from $\mathrm{P_B}$ to $\mathrm{S_B}$. Both approaches should work well, but we took approach (2b) for our prototype implementation.

The protocol thus can be implemented using the proposed framework as follows, where `pa`, `pb`, `sa`, and `sb` are the identifiers of $\mathrm{P_A}$, $\mathrm{P_B}$, $\mathrm{S_A}$, and $\mathrm{S_B}$, respectively (**Fig. 4**):

(1) $\mathrm{P_A}$ sends $\mathrm{S_A}$ `(pa, sa, start-exchange,` $\{v_1$`,` v_2`, pb})`.

(2) $\mathrm{S_A}$ sends $\mathrm{P_B}$ `(sa, pb, offer,` m_1`)`.

(3) $\mathrm{P_B}$ examines m_1 in the received message, and sends $\mathrm{S_B}$ `(pb, sb, accept,` m_1`)` iff m_1 is considered acceptable.

(4) $\mathrm{S_B}$ deletes $\mathrm{v_2}$ and sends $\mathrm{S_A}$ `(sb, sa, agreement,` m_2`)`.

(5) $\mathrm{S_A}$ deletes $\mathrm{v_1}$ and sends $\mathrm{S_B}$ `(sa, sb, confirmation,` m_3`)` iff m_2 is successfully verified.

(6) $\mathrm{S_B}$ stores $\mathrm{v_1}$ and sends $\mathrm{S_A}$ `(sb, sa, commitment,` m_4`)` iff m_3 is successfully verified; otherwise, $\mathrm{S_B}$ runs the abort subprotocol.

(7) $\mathrm{S_A}$ stores $\mathrm{v_2}$ iff m_4 is successfully verified; otherwise, $\mathrm{S_A}$ runs the resolve subprotocol.

Table 2 Time taken to process messages.

Message type	CPU (ms)	I/O (ms)	Total (ms)
start-excg.	50	129	179
accept (m_1)	191	153	344
agreement (m_2)	553	153	706
confirmation (m_3)	402	91	493
commitment (m_4)	24	42	66
Whole exchange	1,220	568	1,788

Table 3 Specifications of the system.

Smartcard	Infineon SLE66CLX320P
CPU clock	15 MHz
Card R/W	Gemplus GemPC Twin
Interface speed	38.4 kbps (Card–R/W), 12 Mbps (R/W–Host)
EEPROM	32 KB
RAM	5 KB
To generate a signature	125 ms (ECDSA, 163 bit)
To verify a signature	185 ms (ECDSA, 163 bit)

The resolve and abort subprotocols can be implemented in a similar way.

5.3 Performance Results

Table 2 shows a performance result of the implementation described above, and **Table 3** gives the specification of the system used.

In Table 2, the columns show (from left to right) the message type, the CPU time consumed by the smartcard (and crypt co-processor), the time for I/O interaction (including the time needed to send the next message), and the sum of the two. The last row lists the time taken to process all messages. We can see that it took only about 1.8 seconds to complete an exchange. These figures do not include the time needed for networking or message handling in the host; however, the message handling cost is negligible in comparison with the time consumed by smartcards, and the networking cost is not large because only a 2-round communication is needed for each exchange.

5.4 Discussion of the Results

5.4.1 Encapsulation of Smartcard Behaviors

As shown in Section 5.2, the implementation of off-card APs is quite simple; what they have to do is to trigger the start of the exchange (for P_A) and to confirm the offer of the exchange (for P_B). The framework encapsulates from off-card APs all other detailed behaviors of the smartcards needed to conduct the protocol.

This encapsulation not only provides simplicity of off-card APs but also offers implementation flexibility, as follows.

Since off-card APs are not concerned with how smartcards perform exchanges, modification of the smartcard implementation has no impact on the implementation of off-card APs; even if the protocol is totally replaced by another protocol, there is no need to modify the off-card APs as long as the messages used to start or confirm an exchange are not modified.

Furthermore, although neither wallet P_A nor P_B is responsible or required to execute an exchange, they can check the progress of the exchange if necessary by hooking messages exchanged between smartcards, using handlers registered with the messaging library; this can be implemented without modifying the smartcard implementation.

The previous studies introduced in Section 3 do not offer such encapsulation and flexibility. An application-oriented framework will be able to provide the same or better abstraction when appropriately designed for exchanging vouchers, but modifications of the protocol are likely to affect the implementation of the framework; when the protocol is replaced, the framework has to be re-implemented. Webcard, WebSIM, JASON, and JavaCard2.2 require at least one off-card AP to be involved in a protocol and mediate messages because of asymmetry. Proactive UICC is potentially extensible to provide similar flexibility by implementing middleware to provide distribution transparency, wherein FETCH commands are automatically issued and the fetched remote messages are forwarded to the appropriate destinations, but developing such an extension would not be an easy task. The situation with CCF is the worst; since smartcards in CCF are always proactive and cannot be reactive, implementation of off-card APs would need some synchronization tricks like "rendezvous" in Ada. This might be possible, but would make off-card APs much more complicated than even those on ISO7816-4.

5.4.2 Efficiency and Performance

The number of messages handled by the smartcards is exactly the same as in the original protocol[☆]. The proposed framework does not demand any extra interactions to implement the protocol; this is the optimum solution.

Most previous architectures, except Proactive UICC, can also offer the same efficiency regarding the number of the messages, provided that the protocol is adequately implemented. Proactive UICC requires additional FETCH commands to allow smartcards to send messages; although it is even possible to implement the protocol without issuing any FETCH commands, this is exactly the same as ISO7816-4 and does not provide any advantages.

The overhead of processing each message is slightly larger than that of ISO7816-4, which is the most efficient, because of the longer headers needed for distribution transparency. The length of the header (or the trailer) of Command APDU and Response APDU is, respectively, 4 bytes and 2 bytes, while that of an eTP message is 34 bytes (where `mtype` is 2 bytes). TENeT therefore requires 62 bytes ($= 34 \times 2 - (4 + 2)$) more than ISO7816-4 for a round trip interaction, which takes about 15 ms, given that the effective throughput of a card R/W is 32 kbps. Since the optimum implementation of the protocol requires $2 + 2$ round interactions between smartcards and hosts, the implementation shown in Section 5.2 is approximately 60 ms slower than ISO7816-4.

Since this penalty is 3% of the total execution time, it does not affect the feasibility of the proposed framework.

6. Security Discussion

6.1 Prevention of Malicious Remote Messages

In this framework, Guard prevents malicious messages that pretend to be messages from an authenticated off-card AP from arriving at a smartcard, as described in Section 4.4.

Guard does not prevent remote message pretending by another remote entity; however, this limitation is not critical since acquiring the privilege from a smartcard by authentication is restricted to off-card APs that belong to the domain of the smartcard, and therefore no remote entity can gain privilege from a smartcard.

Although restricting privileged access within a domain prevents a remote entity from ordering a smartcard to perform potentially harmful operations, it is unlikely to obstruct performance of a distributed protocol that exchanges messages remotely. Such messages are unlikely to need privilege by authentication; the protocol itself would provide adequate security.

For example, in the exchange protocol introduced in Section 5, just two messages require privileges to be granted by a prior authentication: `start-exchange`, which orders a smartcard to start the protocol, and `acceptance`, which informs the other smartcard of the user's acceptance of the exchange[☆☆]. No other messages require privilege, because all are harmless even if they have been forged or falsified; the protocol itself offers smartcards the means to detect such malformed messages.

6.2 Prevention of Pretended Privilege by an Off-card AP

As mentioned in Section 4.4, the security for preventing an unauthenticated off-card AP from pretending to be a privileged off-card AP, which has been successfully authenticated, depends upon access controls of the capability object; if an off-card AP X had some means to access the capability object of a privileged off-card AP Y, X could pretend to be Y and fraudulently gain privilege unless authenticated. In addition, off-card APs must be prevented from directly accessing the dispatch table.

Most modern operation systems and program execution environments for downloaded programs such as Java applets provide means of protecting resources, and these can be adopted to realize the access controls needed.

For example, Java offers a resource protection mechanism called sandbox, which restricts downloaded Java programs, namely applets, to accessing the given limited resources. An applet in the sandbox basically cannot access objects in another applet even if they are executed in

[☆] The accept message is an addition, but the number of the messages the smartcards handle does not change, since S_B does not need to handle the offer message.

[☆☆] The messages that order each smartcard to recover fairness (to start the resolve/abort subprotocol) should be included when considering the recovery subprotocols.

the same virtual machine, except by way of the AppletContext class discussed later; moreover, it cannot access the other objects in the heap memory unless they are explicitly provided by APIs and access has been granted. The sandbox accordingly provides enough access control when off-card APs are implemented as Java applets; the dispatch table can merely be implemented as a collection of heap objects.

When off-card APs are implemented as Java applets, however, it should be noted that Java provides the AppletContext class, whereby an applet can acquire the object reference of another applet if both are downloaded from the same document (e.g., the same web page). This facility is helpful for inter-applet communication, but potentially allows access violations of the capability object. To avoid this problem, implementations of off-card APs should not make the capability object traversable by using public methods of the applet instance.

6.3 Providing Confidentiality and Integrity

While most previous systems that offer remote access to smartcards make an effort to provide access confidentiality and integrity, messages in (our current implementation of) TENeT are not encrypted, nor do they follow MAC. This is one of our design decisions.

These schemes require a key exchange in advance of message exchanges unless it is assumed that pre-shared secret keys are shared by all entities exchanging messages. Performing key exchange protocols, however, is too “heavy” for smartcards, whose I/O performance is quite limited, to apply at the beginning of every interaction.

As mentioned in Section 6.1, most cryptographic application protocols utilizing smartcards are designed without the assumption of secure channels among entities; each message which needs confidentiality or integrity has already been protected by adequate cryptographic techniques such as encryption or digital signatures. Applying them in the message transport layer would be redundant, and should be avoided to ease performance concerns.

If it is required to secure messages from third parties for privacy reasons, providing a secure connection between remote proxies using SSL or IPSec would be sufficient and much faster than providing secure channels between smartcards. Considering the performance disadvantage, encrypting/decrypting messages in smartcards should be avoided even in this case.

7. Conclusion

In this paper, we have proposed a new framework for a distributed smartcard environment, named TENeT, which provides message-based symmetrical interaction among smartcards and application programs. Since this framework enables smartcards to directly interact with one another, application programs that utilize smartcards can be much simpler and more modular than those in previous systems.

Although the proposed framework offers flexible interactions among smartcards and application programs, its feasibility and efficiency are not compromised; it can be efficiently implemented using current smartcards, since nested smartcard invocations are avoided by adopting the message-passing approach, and the security overhead is minimized by avoiding the redundant application of cryptography.

The feasibility and efficiency of this approach have been confirmed by implementing a prototype fair voucher trading system that enables users to trade electronic vouchers fairly among the distributed smartcards, using an optimistic fair exchange protocol. The implementation results show that a system utilizing distributed smartcards can be quite easily implemented using the proposed framework, without incurring any expensive performance overheads.

Acknowledgments The helpful comments of Hiroshi Aono and the anonymous reviewers are gratefully acknowledged.

References

1) 3rd Generation Partnership Project (3GPP): *3GPP TS 31.111: USIM Application Toolkit* (*USAT*) (*Release 6*) (2004).
2) Balacheff, B., Chan, D., Chen, L., Pearson, S. and Proudler, G.: Securing Intelligent Adjuncts using Trusted Computing Platform Technology, *Proc. 4th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2000*), IFIP, pp.177–195 (2000).
3) Balacheff, B., Wilder, B.V. and Chan, D.: Smartcards — From Security Tokens to Intelligent Adjuncts, *Proc. 3rd Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 1998*), IFIP, pp.71–84 (1998).
4) Brinkman, R. and Hoepman, J.-H.: Secure Method Invocation in JASON, *Proc. 5th Working Conference on Smart Card Research*

and Advanced Applications (*CARDIS 2002*), USENIX/IFIP (2002).
5) Chan, P.-K., Choy, C.-S., Chan, C.-F. and Pun, K.-P.: Card-Centric Framework — Providing I/O Resources for Smart Cards, *Proc. 6th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2004*), IFIP, pp.225–240 (2004).
6) European Telecommunications Standards Institute (ETSI): *ETSI TS 102 223: Card Application Toolkit* (*CAT*) (*Release 6*) (2004).
7) Fujimura, K. and Eastlake, D.: *RFC 3506: Requirements and Design for Voucher Trading System* (*VTS*) (2003).
8) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns*, Addison Wesley Longman (1995).
9) Guthery, S., Kehr, R. and Posegga, J.: How to Turn a GSM SIM into a Web Server, *Proc. 4th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2000*), IFIP, pp.209–222 (2000).
10) ISO/IEC: *Integrated circuit*(*s*) *cards with contacts — Part 4: Interindustry commands for interchange* (1995). ISO/IEC 7816-4:1995(E).
11) Itoi, N., Honeyman, P. and Rees, J.: SCFS: A UNIX Filesystem for Smartcards, *Proc. USENIX Workshop on Smartcard Technology*, USENIX (1999).
12) Rankl, W. and Effing, W.: *Smart Card Handbook*, John Wiley & Sons, 2nd ed. (2001).
13) Rees, J. and Honeyman, P.: Webcard: a Java Card Web Server, *Proc. 4th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2000*), IFIP, pp.197–208 (2000).
14) Sakamura, K. and Koshizuka, N.: Business of the Ubiquitous ID Center, *Barcode*, Vol.16, No.5, pp.15–20 (2003). (in Japanese).
15) Sun Microsystems: *Java Card Specification 2.2.1 Final Release* (2003).
16) Terada, M. and Fujimura, K.: *RFC 4154: Voucher Trading System Application Program Interface* (*VTS-API*), IETF (2005).
17) Terada, M., Iguchi, M., Hanadate, M. and Fujimura, K.: An Optimistic Fair Exchange Protocol for Trading Electronic Rights, *Proc. 6th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2004*), IFIP, pp.255–270 (2004).
18) Terada, M., Kuno, H., Hanadate, M. and Fujimura, K.: Copy Prevention Scheme for Rights Trading Infrastructure, *Proc. 4th Working Conference on Smart Card Research and Advanced Applications* (*CARDIS 2000*), IFIP, pp.51–70 (2000).

Appendix: Subprotocols for Recovering Fairness

The following are two subprotocols for recovering fairness if the main protocol introduced in Section 5.1 is interrupted. Note that Terada, et al.[17] use n_1 as the identifier of an exchange in these subprotocols; however, n_2 should be used instead in order to prevent illegal abort requests. When using n_1 as the identifier, the (malicious) user of $\mathrm{S_B}$ can force $\mathrm{S_A}$ to abort the exchange that $\mathrm{S_B}$ has already completed, by replaying m_1 followed by an abort request.

The resolve subprotocol is performed as follows:

(1) $\mathrm{P_A}$ orders $\mathrm{S_A}$ to start the resolve subprotocol.

(2) $\mathrm{S_A}$ sends T a resolve request message m_{r1}: $\{(\mathsf{resolve}|\mathrm{h}(n_2))_{\mathrm{PkA}}, \mathrm{Cert_A}\}$, where n_2 is used as the identifier of the exchange to be resolved or aborted.

(3) T sends $\mathrm{S_A}$ a resolve admission m_{r2}: $\{(\mathsf{resolve}|\mathrm{h}(n_2))_{\mathrm{PkT}}, \mathrm{Cert_T}\}$ iff m_{r1} is successfully verified and the exchange identified by n_2 has not yet been aborted; otherwise, T sends an abort admission m_{a2}: $\{(\mathsf{abort}|\mathrm{h}(n_2))_{\mathrm{PkT}}, \mathrm{Cert_T}\}$ iff m_{r1} is successfully verified and the exchange has already been aborted by $\mathrm{S_B}$.

(4) $\mathrm{S_A}$ stores v_2 iff it received and successfully verified resolve admission m_{r2}, or (re-)stores v_1 iff it received and successfully verified abort admission m_{a2}.

The abort subprotocol is performed in a similar way:

(1) $\mathrm{P_B}$ orders $\mathrm{S_B}$ to start the abort subprotocol.

(2) $\mathrm{S_B}$ sends T an abort request message m_{a1}: $\{(\mathsf{abort}|\mathrm{h}(n_2))_{\mathrm{PkB}}, \mathrm{Cert_B}\}$.

(3) T sends $\mathrm{S_B}$ an abort admission m_{a2} iff m_{a1} is successfully verified and the exchange has not yet been resolved; otherwise, T sends a resolve admission m_{r2} iff m_{a1} is successfully verified and the exchange has already been resolved by $\mathrm{S_A}$.

(4) $\mathrm{S_B}$ (re-)stores v_2 iff it received and successfully verified abort admission m_{a2}, or stores v_1 iff it received and successfully verified resolve admission m_{r2}.

(Received May 19, 2005)
(Accepted November 1, 2005)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.2, pp.120–132.)

Masayuki Terada received his M.E. degree from Kobe University in 1995. He joined NTT in 1995 and engaged in research on electronic rights management systems. Since 2003 he has worked for NTT DoCoMo and engaged in research on fairness in electronic commerce. His current research interests are in the areas of security protocols and application of smartcards. He is a member of IPSJ and IEICE.

Kensaku Mori received his M.E. degree from Kyushu University in 2000. He joined NTT DoCoMo in 2000. He has engaged in R&D on location information services and e-commerce security. He is a member of IPSJ.

Kazuhiko Ishii received his B.E. degree from the University of Electro-Communications in 1990. He joined NTT DoCoMo in 1999. He has engaged in R&D on software development networks, network multimedia technology and mobile e-commerce security. He is a member of IPSJ.

Sadayuki Hongo received his M.E. degree from Iwate University in 1984. He joined NTT in 1984. From 1987 to 1991 he belonged to ATR. He transfered to NTT DoCoMo in 1999. He has engaged in research on intelligent telephone terminals, telephone-terminal operation behavior, icon recognition, computational theory of visual information processing, multimedia education, and information security. He is a member of IPSJ and IEICE.

Tomonori Usaka is a research associate at the University Museum, the University of Tokyo. His research interests include human machine interface, digital museum, and ubiquitous computing. He has a B.S., M.S., and Ph.D., all in information science, from the University of Tokyo. He is a member of the IEEE Computer Society and the ACM.

Noboru Koshizuka is an Associate Professor in Information Technology Center at the University of Tokyo, Japan. He has graduated Graduate School of Science at the University of Tokyo in 1994, and received the D.S. degree from the University of Tokyo. His research interests are ubiquitous computing, embedded system, secure system, human interface, and computer networks. He is participating in Ubiquitous ID Project, T-Engine Project, and TRON Project.

Ken Sakamura was born in Tokyo in 1951, He obtained his Ph.D. degree in Electrical Engineering from Keio University in 1979. His main research area is computer architecture. He has been the leader of the TRON project since 1984. In this capacity, he has designed and published the TRON computer system architecture, which will be useful for ubiquitous computing of the future. Today, the real-time operating systems based on the TRON specification are used for engine control on automobiles, mobile phones, digital cameras, and many other devices, and is believed to be the most popular such operating systems of the kind all over the world. Aside from the computer systems, he has paid attention to the artistic design of electronic appliances, furniture, houses, buildings, urban landscape, and museums, all of which contain the networked embedded computers based on the TRON specification. In a nutshell, he has led the research of the total system architecture of ubiquitous computing in our daily lives. At present, he is a professor of the Interfaculty Initiative in Information Studies at the Graduate School of the University of Tokyo, director of the YRP Ubiquitous Networking Laboratory, and a fellow of the IEEE Computer Society. He was the recipient of The 33rd Ichimura Prizes in Technology — Main Prize, the recipient of Takeda Award 2001, the recipient of the Medal with Purple Ribbon in 2003, and the recipient of Okawa Prize in 2004.