Towards a Secure and Practical Multifunctional Smart Card

Idir Bakdi

Lehrstuhl für Wirtschaftsinformatik II
Universität Regensburg
93040 Regensburg, Germany
idir.bakdi@wiwi.uni-regensburg.de

Abstract. One of the most promising features of smart card technology is its potential to serve several applications using a single hardware token. Existing multifunctional smart cards, however, are either simple and suffer from serious limitations or they have a high complexity that is not justified for most applications. This paper describes a new scheme permitting different applications to flexibly share a hardware token. The proposed solution supports off-line transactions as well as post-issuance loading. Each application can load one or more "virtual tokens" (remotely) into a common smart card. Despite its simplicity, the scheme guarantees the authenticity and integrity of virtual tokens and prevents their duplication. Moreover, it protects the privacy of card holders by providing a possibility to use pseudonymous identities that cannot be linked to one another.

Keywords: multifunctional smart card; secure hardware token; privacy

1 Introduction

Nowadays, we all carry a lot of tokens in our pockets. These are keys, magnetic stripe cards, smart cards, tickets, etc. Each application has to implement its own infrastructure to issue and subsequently use these tokens. From an ergonomic as well as from an economic point of view, it would be beneficial to virtualize all those tokens, i.e. to convert them into digital files that can be loaded onto a single medium. However, as most tokens are critical for the security of the applications they serve, one cannot just load their virtual counterparts on a storage medium such as a floppy disk. On the one hand, such a medium would not offer any protection against misuse in case it gets lost or stolen. On the other hand, a virtual token could be copied at will as it can be read by anyone holding the medium. One possibility to overcome these difficulties is to use a microchip as implemented on smart cards to hold the files representing tokens. This way, they are protected both against misuse by someone else than the legitimate user and against unauthorized duplication by the holder himself. This paper describes a new scheme that uses such a chip to realize a secure and practical multifunctional token.

The remainder of this paper is organized as follows. After a short review of existing solutions and their most serious shortcomings in the next section, the main security requirements of a multifunctional smart card are outlined in section 3. Section 4 contains a description of the new approach. Finally, the last section is devoted to some conclusions summarizing this work.

2 State of the Art

The idea of using a single token for multiple purposes is quite an old one. Its simplest and most famous implementation is perhaps that of a master key able to open several locks. Recently, many solutions appeared which aim at grouping many applications on a single smart card, or more generally, on a single microchip. Most of them can be categorized into one of two classes. The first class, denoted by *single ID cards* in this paper, is the simpler one. So called *multi-application smart cards* constitute the second class. They are more comprehensive and consequently more complex than single ID cards. In the following, these two classes are described in more detail sketching their pros and cons.

2.1 Single ID Cards

The simplest realization of a single ID card is to have a number (an ID) that uniquely identifies a person across several back-end systems stored on a smart card. Each system keeps its own set of data associated with a given user in its central database. The smart card merely serves as a reference to that account. As all systems share the same number for a given user, a single card suffices to identify him in all of them. Put another way, single ID cards realize an authentication by possession of a person already known to the different systems through a common identification scheme. A more advanced version of this solution uses smart cards capable of producing digital signatures. The idea is to have the user's private key stored on his card, whereas the corresponding public key is shared by all participating back-end systems. The main advantages and drawbacks of this class are presented in the following.

Advantages

Simplicity: Solutions built on single ID cards are very simple. A single ID card merely identifies a person. Each application maps the ID to a data set associated with the corresponding user. The management of relevant application data is thus differed from the card to the more powerful back-end systems, which facilitates implementation.

Generic Digital Signature: In case a digital signature card is used, the card can additionally be used to digitally sign electronic documents. Thus, a large category of applications requiring the authenticity and non-repudiation of electronic documents can be covered. Many countries already have laws that guarantee to certain kinds of digital signatures a legal status equivalent to that of a classical (i.e. manual) signature.

Drawbacks

No Off-line Functionality: As the card merely represents a pointer to a data set stored in the back-end system, it can only be used given an online access to the central database. This is especially difficult when the verifier does not belong to the issuer's organization. To verify a student card, an employee sitting at the entrance of a cinema would have to access data from the student office of the corresponding university.

Privacy Concerns: As all the back-end systems use the same ID, comprehensive user profiles can be easily constructed by matching the different data sets belonging to a person. This raises privacy concerns because of possible misuse.

Single Point of Failure: The different back-end systems all rely on a unique ID per user which is tightly bound to a single card. If this hardware token is lost, stolen, or compromised, all those systems are affected at once. That is, the card holder has no possibility to mitigate the risk resulting from a loss or theft by employing more than one smart card.

High Demands: In case a signature card is used, the requirements concerning the card's protection and the key management are very high. This makes sense when the card is used to generate legally recoverable digital signatures. For most use cases, however, such functionality is not needed. A train conductor only has to be convinced that a passenger possesses a valid ticket. He does not need to get the traveller's legally recoverable signature. In such cases, high demands would unnecessarily burden the solution.

2.2 Multi-Application Smart Cards

This class employs so called *multi-application card operating systems (MACOS)* [14, p. 308] which try, in analogy to computer operating systems, to abstract the underlying hardware in order to make it possible for different applications to run on it. They offer an application programmer interface (API) that can be used to access the card's services. The newer MACOS (e.g. MULTOS [9] or Java Card [19] [8] [5]) do not only provide for the hosting of many applications on the same microprocessor, but also make it possible for the same application to run on different chips by employing a virtual machine. Using the Java Card platform for instance, each system can load its applet (a small application written in Java) on the card where it can be executed together with other applets. In the remainder of this section, the main advantage of multi-application smart cards as well as their drawbacks are discussed.

Advantage

Universality: The vision of multi-application smart cards is to have a universal chip able to execute arbitrary code. That is to say, the goal consists in the miniaturization of multi-purpose devices such as personal computers or handhelds. This would enable everyone to write applications that do whatever he wants and load them to be executed on card. The employed smart card would bear all the necessary functionality and would not need to rely on any back-end system, thus providing a high level of flexibility and autonomy.

Drawbacks

Complexity: Due to their complexity these systems are far from being mature. This results in the following limitations of current solutions.

- The capabilities of MACOS are restricted due to limitations in processing performance and storage capacities of the underlying hardware. It will take some time before they reach the universality of operating systems running nowadays on personal computers, for instance.
- Many of the current solutions do not offer the possibility of post-issuance loading. The applications are installed on the card before the latter is issued. This is for instance the case when using MULTOS [12]. Java applets can be loaded after the card is issued, but until version 2.2 of the Java Card specification there was no possibility to remove already loaded applets. This shows the kind of difficulties encountered in practice.

Security: The main advantage of multi-application smart cards, consisting in their ability to host arbitrary applications and to execute their code, constitutes at the same time a considerable security risk. Some applications running on the card may not be trusted. They could access sensitive data of other applications residing on the same chip. Currently, many efforts are made to secure smart card applets against one another using e.g. so called firewalls [6] [18] or byte code verification [13]. This is a cumbersome task. The cost of evaluating, for instance, an application written for MULTOS according to ITSEC E6 [7] is estimated to be 150% of the overall development cost [4]. Besides, their genericity does not allow to hard-wire once and for ever protection mechanisms needed to ensure the requirements of uniqueness and privacy described in the next section. Instead, each application has to implement its own security framework, possibly leading to new vulnerabilities.

3 Security Requirements

A multifunctional smart card as introduced in the first section should satisfy at least the following security requirements.

Authenticity: Only a legitimate issuer should be able to produce authentic virtual tokens for a given application. No one besides the student office should be capable of issuing valid student cards.

Integrity: A solution has to make sure that, once issued, a virtual token can not be modified, not even by its holder. The student must not be able to change the validity period of his card himself.

Uniqueness: Virtual tokens have to be protected against duplication. This is especially important when considering applications where the token is used as a dongle or as a ticket. Imagine a railway ticket that can be duplicated at will.

Privacy: This requirement could also be entitled "separability". Physical tokens, although perhaps belonging to the same user, are not a priori linked to each other. Often they are not even associated with their holder. In no way can a car key be linked to any of the other tokens a user has in his pocket, nor is it related to his person. The same should hold when replacing physical tokens by virtual ones. Else, profiles existing in the different systems could be easily matched to get a comprehensive picture of the user. Also, when virtual tokens need to be verified it should be possible to present them separated from each other, though residing on the same chip. All a train conductor has to know is that the traveller paid for his trip. This requires him to look at the passenger's ticket but not to learn his identity.

The requirements of authenticity and integrity are satisfied by letting the issuer digitally sign each virtual token he issues. Verifying this signature ensures that only authentic and untampered virtual tokens are accepted as valid. The scheme described in the next section also guarantees uniqueness and privacy.

4 New Approach

The proposed solution is based on the concept of *virtual tokens (VT)* introduced above. These are hosted by a microchip called *digital pocket (DP)* in this paper, in analogy to a pocket that holds physical tokens. VTs can be loaded (e.g. over the Internet) into a DP after the latter is issued (post-issuance loading). They can even be moved by their holder from one DP to another without any intervention by the issuer, which makes the scheme very flexible. In a certain way, VTs resemble attribute certificates as described e.g. in [3]. The main difference is that a VT can only be used in conjunction with a single hardware token (i.e. a DP) at a given time, thus preventing its duplication.

Each DP is embedded into a container that provides it with power supply and a communication interface. The most obvious realization of this idea is to use a smart card in conjunction with a reader providing the required infrastructure. However, it could be implemented as well in a mobile phone, in a wrist watch, or in any other object the user bears with him. For the sake of simplicity a smart card realization is assumed in this paper.

The main actors taking part in the scheme are identified in the next subsection. After sketching DP's architecture and describing the involved key pairs, the scheme is outlined in subsection 4.4. Finally, a short analysis of the trust relationships that have to be assumed among the different roles and a discussion comparing the new scheme to existing solutions are presented.

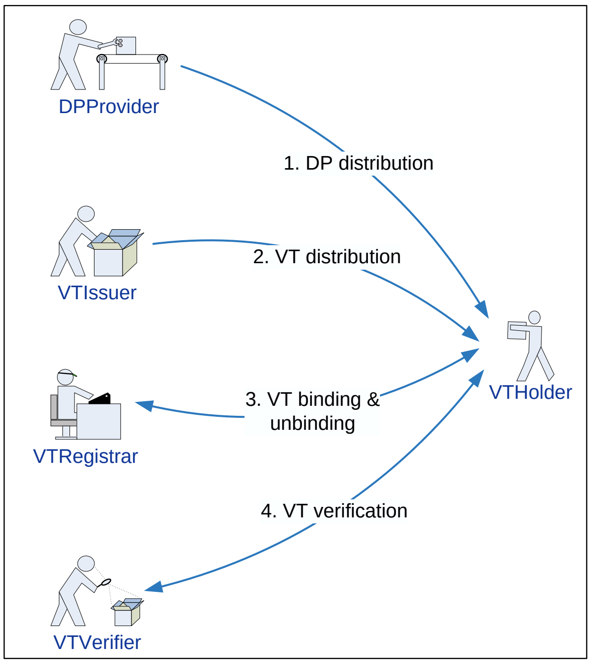

Fig. 1. The different roles and their interactions

4.1 Roles

The following actors take part in the considered setting. Fig. 1 summarizes their interactions.

- *Digital Pocket Provider (DPProvider)*: is the issuer of DPs.
- *Virtual Token Issuer (VTIssuer)*: issues VTs (e.g. the railway company).
- *Virtual Token Holder (VTHolder)*: is a person holding one or more VTs (e.g. a railway passenger).
- *Virtual Token Verifier (VTVerifier)*: verifies VTs (e.g. a train conductor).
- *Virtual Token Registrar (VTRegistrar)*: represents a trusted institution assuring that there is no more than one active copy of a given VT at any time.

These roles may be assumed by distinct persons and/or institutions. However, a single organization may also assume several of them. DPProvider and VTRegistrar could for example be embodied by the same infrastructure operator. Another option is to have VTIssuer ensure the uniqueness of VTs he issues, thereby additionally playing VTRegistrar's role.

4.2 Architecture

The proposed solution can be readily implemented using standard hardware components as they can be found on modern smart cards. More precisely, DP's architecture consists of the following elements.

- *Protected Memory (PM)*: represents the area of the chip where VTs and all relevant public keys are stored. This area is accessible to VTHolder after authorization, e.g. using a PIN or some kind of biometrics. The protection is meant to prevent VTVerifier or someone else from randomly reading the content of a DP without VTHolder's consent.[1]
- *Tamper-Resistant Memory (TRM)*[2]: is an area that can not be read from the outside, not even by VTHolder. It serves as a storage for private keys.
- *Tamper-Resistant Processing Unit (TRPU)*: represents a processing unit for the execution of computations involving secrets.
- *Controller*: coordinates the single actions of a DP and provides an interface to the outside world. It handles the communication with VTHolder, VTVerifier and VTRegistrar.

4.3 Involved Key Pairs

The presented solution relies on public key cryptography. In the remainder of this paper, a private key used to create a digital signature is always denoted by S_x (for some index x), whereas P_x stands for the corresponding public key employed for signature verification. More precisely, the following key pairs are used to generate and verify digital signatures in the different phases of the scheme.

(S_{VTIssuer} , P_{VTIssuer}): Every VT issued by a VTIssuer is signed with his private key S_{VTIssuer} and can be checked for validity using the corresponding public key P_{VTIssuer}.

($S_{\mathrm{VTRegistrar}}$, $P_{\mathrm{VTRegistrar}}$): VTRegistrar employs his private key $S_{\mathrm{VTRegistrar}}$ to generate binding confirmations that can be verified using $P_{\mathrm{VTRegistrar}}$.

(S_{VT} , P_{VT}): Each VT is bound to a DP using a dedicated key pair (S_{VT}, P_{VT}).

(S_g , P_g): Before their distribution, DPs are divided by DPProvider into groups. All the DPs of a given group are assigned the same key pair (S_g , P_g).

DPs are grouped in order to protect the privacy of their holders. That is, all DPs belonging to the same group g share a common key pair (S_g , P_g). Thus,

[1] To make such a protection effective a secure communication channel between VTHolder and DP is needed. This involves an input device (e.g. a keypad) and an output device (e.g. a small display) that are tamper-resistant. However, to keep the system's description simple this point will not be further elaborated in this paper.

[2] As a perfect protection of hardware tokens averting every attack can hardly be achieved [1], the term "tamper-resistant" is used instead of "tamper-proof" to make clear that despite great efforts to protect the microchip, a risk of compromise still exists. The assumption is of course, as with other schemes, that the token's physical protection is sufficient for its purpose.

they cannot be distinguished from one another. If each DP had its own key pair, the different identities connected to it could be linked together. This is prevented by letting an individual DP hide in its group much in the same way that a single Internet user hides in a group of surfers when using anonymizer services built on crowds [15]. Although the fact that several chips carry the same private key would appear to increase the security risk, such is not the case. As will become clear from the following description of the scheme, the damage caused by a compromised DP is independent of whether it was assigned a unique key or whether it shares it with a number of other DPs.

4.4 How it Works

The main phases of the scheme comprise:

a) Initialization: Before its delivery to a VTHolder, each DP is initialized by DPProvider. To initialize a group of DPs, DPProvider generates a new key pair (S_g , P_g). The private key S_g is stored in the tamper-resistant memory (TRM) of each card in the group. Further, the corresponding public key P_g is included into *PubList*, which is the list of the public keys of all DPs issued so far.[3]

b) Distribution: Once the DPs have been initialized, they can be distributed to VTHolders through any channel. At this stage, all the DPs are identical in the sense that they are neither VTHolder specific nor VTIssuer specific. A VTHolder could just buy an "empty" DP in the supermarket to load his VTs on it.

c) Virtual Token Generation: In order to issue a VT, VTIssuer has to write the application dependent data into a file and to sign it with his private key S_{VTIssuer}. The format of this file may be freely chosen by VTIssuer (as long as VTVerifier is able to make sense of it). The signed file constituting a VT can be transferred to VTHolder via e-mail or any other means. If the VT contains confidential data it must, of course, be protected on its way to VTHolder. After receiving a VT, VTHolder has to bind it to a particular DP before it will be accepted by VTVerifier.

d) Binding: Binding a VT to a DP ensures that it can not be duplicated (see the requirement of uniqueness in section 3). VTRegistrar knows about every VT he has bound to a DP and is responsible for the prevention of multiple bindings. To do so, he stores the hash value of each VT that he binds in a list called *BoundList*. Storing only a hash value and not the VT itself prevents VTRegistrar from learning the token's content, thus guaranteeing VTHolder's privacy as required in section 3. Moreover, the use of a hash value improves the system's efficiency, especially when a big number of VTs has to be managed. The details of the binding process are depicted in protocol 1 (see also Fig. 2).

[3] An update of PubList is regularly propagated to VTRegistrar (e.g. using a public key infrastructure).

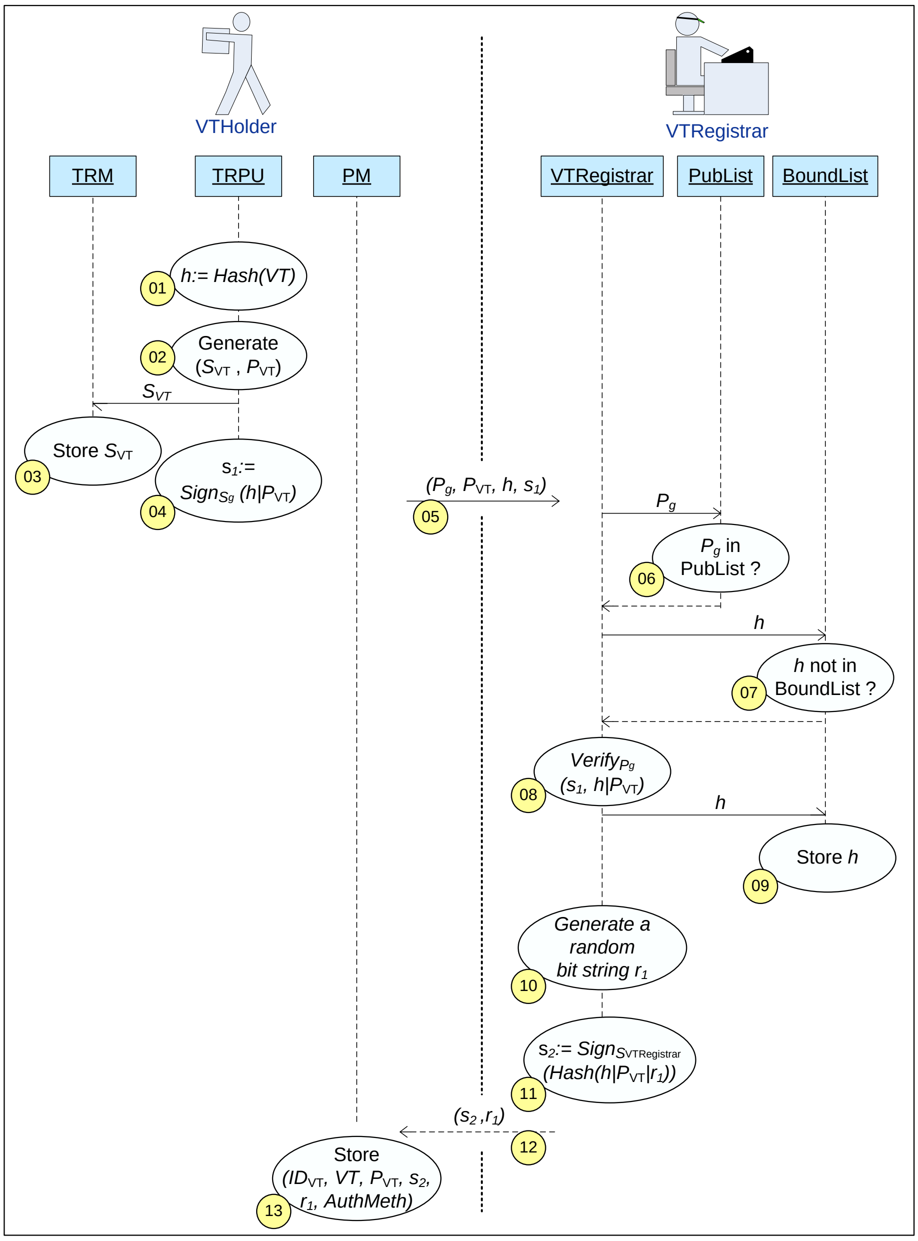
VTHolder
VTRegistrar
TRM
TRPU
PM
VTRegistrar
PubList
BoundList
01 h:= $Hash(VT)$
02 Generate (S_{VT} , P_{VT})
S_{VT}
03 Store S_{VT}
04 s_1:= $Sign_{S_g}$ ($h|P_{VT}$)
05 (P_g, P_{VT}, h, s_1)
P_g
06 P_g in PubList ?
h
07 h not in BoundList ?
08 $Verify_{P_g}$ (s_1, $h|P_{VT}$)
h
09 Store h
10 Generate a random bit string r_1
11 s_2:= $Sign_{S_{VTRegistrar}}$ ($Hash(h|P_{VT}|r_1)$)
12 (s_2 ,r_1)
13 Store (ID_{VT}, VT, P_{VT}, s_2, r_1, $AuthMeth$)

Fig. 2. Binding a VT to a DP by VTRegistrar (Protocol 1)

Protocol 1 (Binding of a VT to a DP by VTRegistrar):

1. TRPU calculates a hash value of VT:
$$h := Hash(\mathrm{VT}).^{4}$$
2. A new key pair $(S_{\mathrm{VT}}\ , P_{\mathrm{VT}})$ is generated inside TRPU.
3. S_{VT} is stored in TRM.
4. TRPU signs the bit string $h|P_{\mathrm{VT}}$ using S_g:
$$s_1 := Sign_{S_g}(h|P_{\mathrm{VT}}).^{5}$$
5. The tuple $(P_g, P_{\mathrm{VT}}, h, s_1)$ is sent to VTRegistrar.
6. VTRegistrar verifies that P_g is contained in PubList.
7. VTRegistrar verifies that h is not yet contained in his list of bound VTs (BoundList).
8. VTRegistrar checks the validity of s_1 using P_g:
$$Verify_{P_g}(s_1, h|P_{\mathrm{VT}}).$$
9. VTRegistrar stores h in BoundList.
10. VTRegistrar generates a random bit string r_1.
11. VTRegistrar signs $Hash(h|P_{\mathrm{VT}}|r_1)$ with his private key $S_{\mathrm{VTRegistrar}}$:
$$s_2 := Sign_{S_{\mathrm{VTRegistrar}}}(Hash(h|P_{\mathrm{VT}}|r_1)).$$
12. VTRegistrar sends the pair (s_2, r_1) back to VTHolder.
13. The tuple $(ID_{\mathrm{VT}}, \mathrm{VT}, P_{\mathrm{VT}}, s_2, r_1, AuthMeth)$ is stored in PM, where ID_{VT} stands for the ID of the application VT belongs to and $AuthMeth$ for the authentication method to enforce before granting access to this particular VT.[6]

By signing $Hash(h|P_{\mathrm{VT}}|r_1)$ in step 11 VTRegistrar confirms that the VT which hashes to h was bound to the DP that holds the corresponding private key S_{VT} in its TRM. Such a confirmation is only issued if three conditions are met:

i. The binding was actually requested using a DP (step 8 of the previous protocol),
ii. that DP is genuine, i.e. its public key P_g is contained in PubList (step 6), and
iii. the VT in case is not yet bound to another DP (step 7).

[4] $Hash(\cdot)$ is assumed to be a cryptographically secure hash function, i.e. one that is collision and preimage resistant (see e.g. [11, p. 323]).
[5] $b_1|b_2$ stands for the concatenation of the bit strings b_1 and b_2.
[6] Depending on the use case, *AuthMeth* is determined either by VTIssuer or by VTHolder.

Remark 1. If in step 11 VTRegistrar just signed $h|P_{\mathrm{VT}}$ instead of $Hash(h|P_{\mathrm{VT}}|r_1)$ then chosen-ciphertext attacks could be feasible. This is why [16, p. 54] points out that "it is foolish to encrypt arbitrary strings".

Remark 2. The key pair (S_{VT} , P_{VT}) has to be generated securely inside the TRPU so that nobody learns its value. Some smart cards use pseudo random number generators with a seed set by the manufacturer [14]. Such smart cards are unsuitable for this scheme as everyone knowing the seed could deduct the entire pseudo random number sequence. The keys rather have to stem from a physical source of randomness (see e.g. [2] for a true random number generator suitable for integration on smart cards).

e) Verification: When a DP is asked to present a given VT to VTVerifier, it first requests an authorization from VTHolder as mentioned in section 4.2. In case VTHolder approves, DP sends VT together with the binding confirmation to VTVerifier who checks their validity. Protocol 2 (depicted by Fig. 3) contains the necessary steps.

Protocol 2 (Verification of a VT by VTVerifier):

1. VTVerifier generates a random challenge c.
2. VTVerifier requests VT from DP's Controller by sending it the pair (ID_{VT}, c).
3. Controller uses the specified $AuthMeth$ to ask VTHolder for permission to show VTVerifier the VT in case.
4. TRPU generates a random bit string r_2.
5. TRPU generates the following signature:
$$s_3 := Sign_{S_{\mathrm{VT}}}(Hash(c|r_2)).$$
6. Controller sends the tuple $(\mathrm{VT}, P_{\mathrm{VT}}, s_2, r_1, s_3, r_2)$ to VTVerifier.
7. VTVerifier checks the validity of s_2 using $P_{\mathrm{VTRegistrar}}$:
$$Verify_{P_{\mathrm{VTRegistrar}}}(s_2, Hash(Hash(\mathrm{VT})|P_{\mathrm{VT}}|r_1)).$$
8. VTVerifier checks the validity of s_3 using P_{VT}:
$$Verify_{P_{\mathrm{VT}}}(s_3, Hash(c|r_2)).$$
9. VTVerifier checks the authenticity and integrity of VT using P_{VTIssuer}.

In step 7 of this protocol, VTVerifier gets convinced that the VT he received is uniquely bound to the DP that holds the private key S_{VT} corresponding to P_{VT}. By looking at s_3 in step 8, he verifies that he is actually communicating with that DP. Finally, the authenticity and the integrity of the VT itself are verified in the last step.

Remark 3. DP could use a suitable zero-knowledge protocol as described e.g. in [11, pp. 405–417] to convince VTVerifier that it holds S_{VT} without divulging it. However, in this paper a digital signature was chosen for this purpose in order to simplify matters.

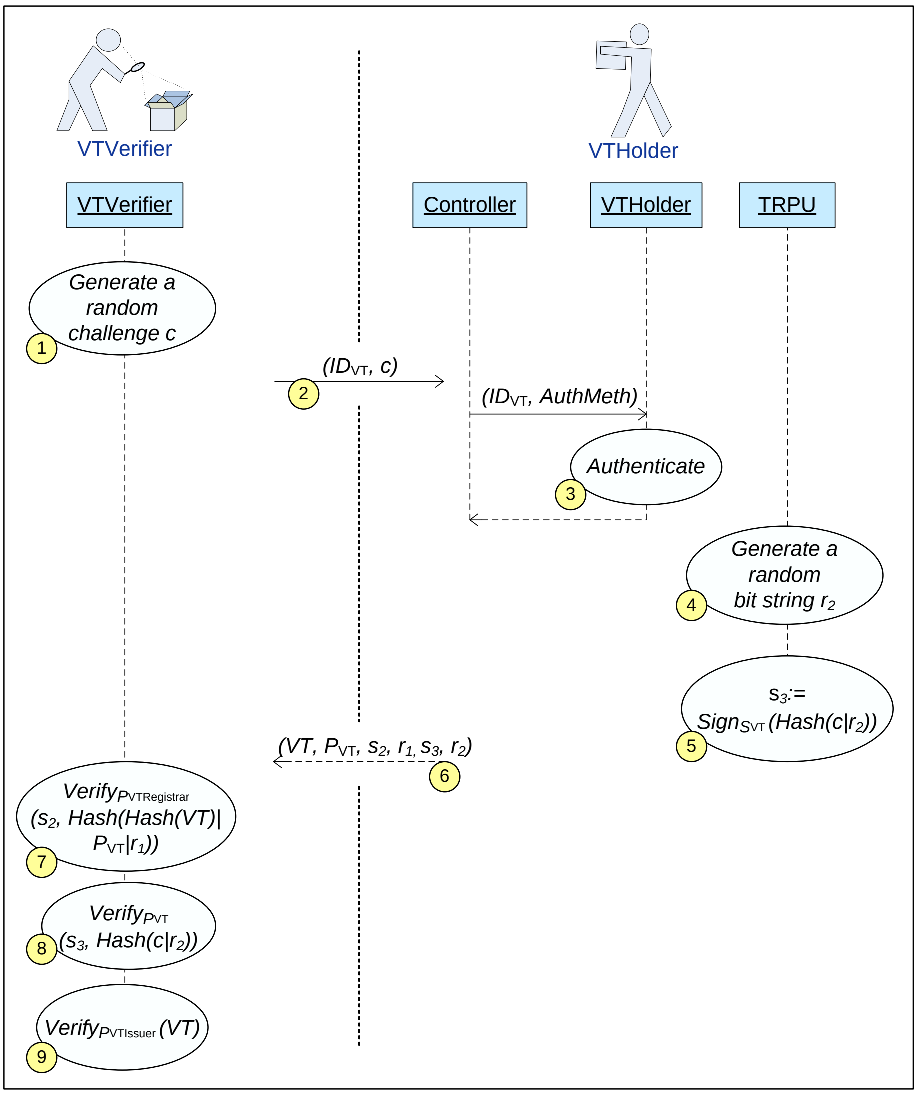
VTVerifier
VTHolder
VTVerifier
Controller
VTHolder
TRPU
Generate a random challenge c
1
(ID_{VT}, c)
2
$(ID_{VT}, AuthMeth)$
Authenticate
3
Generate a random bit string r_2
4
$s_3 := Sign_{SVT}(Hash(c|r_2))$
5
$(VT, P_{VT}, s_2, r_1, s_3, r_2)$
6
$Verify_{PVTRegistrar}(s_2, Hash(Hash(VT)|P_{VT}|r_1))$
7
$Verify_{PVT}(s_3, Hash(c|r_2))$
8
$Verify_{PVTIssuer}(VT)$
9

Fig. 3. Verification of a VT by VTVerifier (Protocol 2)

f) Unbinding: VTHolder may want to use more than just one DP and to be able to transfer VTs among them. He could use a DP for his private VTs and a separate one for the VTs he needs at work. When going on a vacation trip he may choose to only take certain VTs with him in order to reduce the damage caused by a possible loss or theft. To satisfy this requirement, a mechanism is needed which enables the unbinding of a VT from the DP it is bound to. Protocol 3 realizes this task.

Protocol 3 (Unbinding of a VT from a DP by VTRegistrar):

1. TRPU calculates
$$h := Hash(\text{VT}).$$
2. TRPU signs h with S_{VT}:
$$s_4 := Sign_{S_{\text{VT}}}(h).$$
3. S_{VT} is removed from TRM.
4. The tuple $(h, P_{\text{VT}}, s_2, r_1, s_4)$ is sent to VTRegistrar.
5. VTRegistrar checks the validity of s_2 using his own public key $P_{\text{VTRegistrar}}$:
$$Verify_{P_{\text{VTRegistrar}}}(s_2, h|P_{\text{VT}}|r_1)).$$
6. VTRegistrar checks the validity of s_4 using P_{VT}
$$Verify_{P_{\text{VT}}}(s_4, h).$$
7. VTRegistrar removes h from BoundList.

Step 5 of the above protocol ensures that the VT which hashes to h was actually bound to the key pair (S_{VT}, P_{VT}). In Step 6 VTRegistrar gets convinced that the unbinding request comes from the DP to which VT is currently bound, namely the one holding S_{VT}.

Remark 4. So far, a single VTRegistrar was assumed in order to keep the description simple. Nevertheless, the scheme is able to accommodate any number of VTRegistrars. By letting each VTRegistrar manage his own BoundList, a decentralized solution is obtained. The different VTRegistrars do not even need to communicate with each other. In order to designate a VTRegistrar responsible for the uniqueness of a given VT, VTIssuer would include the public key $P_{\text{VTRegistrar}}$ of that VTRegistrar into the VT before signing it. This way, VTVerifier learns which public key he has to employ in order to verify s_2 (step 7 of protocol 2). This flexibility is especially important to make a viable business model possible, because it avoids dependency on a single institution. A VTIssuer himself could for instance care about the binding of VTs issued by him. For the sake of completeness, it should be mentioned that the drawback of having many VTRegistrars is the bigger overhead when moving VTs from one DP to another. For each VT being transferred the corresponding VTRegistrar has to be contacted to unbind it from the first DP and to rebind it subsequently to the second one. This could impair the flexibility of the scheme in case VTs are frequently moved.

4.5 Who Trusts Whom?

To better understand the dependencies between the single roles their trust relationships are examined in the following.

VTRegistrar: The only thing VTRegistrar has to rely on is that the DPs behave correctly, i.e. that they do not divulge private keys and that they execute operations only according to the protocols presented above. In other words, VTRegistrar has to trust DPProvider to only issue DPs that work as specified and to provide him with the correct list of valid public keys (PubList). In all the other actors VTRegistrar does not need to trust.

VTHolder: Like VTRegistrar, VTHolder has to trust DPProvider. If his DP functions correctly and keeps its secrets safe, VTHolder does not have to trust VTRegistrar, because the latter does not learn any sensitive information in the course of binding and unbinding VTs. Conversely, assuming the trustworthiness of VTRegistrar, VTHolder cannot be fooled into using a manipulated DP, because he would notice it as soon as he tries to use the chip for the first binding. This renders the distribution channel for DPs uncritical.

VTIssuer/VTVerifier: VTIssuer and VTVerifier have to trust VTRegistrar to ensure the uniqueness of VTs, i.e. that binding confirmations are only issued for VTs not already bound. In particular, they have not to rely on the genuineness of a given DP, because VTHolder would not be able to get a binding confirmation from VTRegistrar if his hardware token were not working properly.

Remark 5. Someone who learns S_g would be able to request bindings that he may copy. This is because he could generate a key pair $(S_{\mathrm{VT}}, P_{\mathrm{VT}})$ outside a DP, use S_g to get a binding confirmation from VTRegistrar, and thus know the private key S_{VT} that is necessary for duplication. However, he would not be able to generate a second binding for a given VT using a different pair $(S_{\mathrm{VT}}, P_{\mathrm{VT}})$, nor could he use any previously bound VT if he lacks the corresponding DP. This is why the consequences of compromising a DP's private key are the same whether it is shared within a group of DPs or not.

Remark 6. The scheme described above assumes a running public key infrastructure (PKI). This PKI is, however, only needed in order to manage PubList and the public keys of VTRegistrars and VTIssuers. While PubList has to be accessible to VTRegistrars, the only actor interested in getting authentic public keys of VTRegistrars and VTIssuers is VTVerifier. VTHolders and their DPs have no public keys to be managed by this PKI. This makes the required PKI much simpler than one needed by single ID cards able to generate digital signatures, for instance.

4.6 Discussion

The solution proposed in this paper can be seen as a pragmatic compromise between the inflexible single ID cards, on one hand, and the cumbersome multi-application smart cards, on the other hand. While single ID cards represent the simplest solution and have therefore many serious limitations, multi-application smart cards provide the most flexible system, but still need a lot of work to reach their vision. The main difference between the new approach and single ID cards is that the former employs an independent virtual token for each back-end system. The virtual tokens can be used without any online access to a central database and they can be transferred from one hardware token to another without being duplicable. In contrast to multi-application smart cards, a DP merely stores data on the chip, but no application-specific code is executed on it. This makes implementation easier and avoids many security problems. [17] states that "...for many applications, using a smart card securely means understanding it not as a 'trusted' computation platform, but as a data storage device with limited computational abilities".

Nevertheless, there are also applications not covered by the new scheme. These are use cases requiring some application-specific code to be executed on card (e.g. digital signature cards, digital cash cards with a purse-to-purse functionality such as Mondex [10], etc.). Hence, the approach presented above is not meant as a substitute for the other solutions but rather as a complement. Note that it is for example possible to combine a digital signature card with the scheme presented in this paper to cover an even larger set of applications. Summarizing, one could say that the presented approach, although not covering all use cases, is able to avoid undue complexity and still serve a big number of applications adequately.

5 Conclusions

A new scheme for a multifunctional hardware token was described in this paper. It is based on the observation that many applications (e.g. driving licenses, student ID cards, credit cards, loyalty cards, pay TV cards, subway tickets, etc.) do not necessarily need the execution of application-specific code on card nor the ability to generate digital signatures. The role of each actor in the scheme, which can be assumed by any person or organization, was clearly defined and the interactions taking place between the different actors were specified. The architecture of a microchip needed to host different virtual tokens was roughly sketched. Unlike attribute certificates, virtual tokens are bound to a single hardware token, which prevents them from being duplicated. Not only does the scheme enable off-line use and post-issuance loading, but virtual tokens can also be transferred among different hardware tokens making the solution very flexible. Moreover, the scheme inherently provides for authenticity, integrity, and privacy. Comparing it to other approaches it was shown that while certainly not providing a universal solution, it may help considerably towards a secure and practical multifunctional smart card.

References

1. Anderson, Ross and Kuhn, Markus: Tamper Resistance - a Cautionary Note. In: Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland,

CA, USA (1996), 1–11.

2. Bucci, Marco; Germani, Lucia; Luzzi, Raimondo; Trifiletti, Alessandro and Varanonuovo, Mario: A High Speed Oscillator-Based Truly Random Number Source for Cryptographic Applications on a Smart Card IC . In: IEEE Transactions on Computers, No. 4, Vol. 52 (2003), 403–409.
3. Chadwick, David W.: The X.509 Privilege Management Infrastructure. In: Proceedings of the NATO Advanced Networking Workshop on Advanced Security Technologies in Networking, Bled, Slovenia, 2003.
4. Chan, Siu-cheung Charles: Infrastructure of Multi-Application Smart Card (in the concerns of access control). `http://home.hkstar.com/~alanchan/papers/multiApplicationSmartCard/`, download 2005-02-28 (1997).
5. Chen, Zhiqun: Java Card Technology for Smart Cards: Architecture and Programmer's Guide. Addison-Wesley Professional, Amsterdam (2000).
6. Éluard, Marc; Jensen, Thomas and Denney, Ewen: An Operational Semantics of the Java Card Firewall. In: Proceeding of Smart Card Programming and Security (ESMART), Lecture Notes in Computer Science, 2140, Springer-Verlag, Berlin Heidelberg New York (2001), 95–110.
7. European Union (ed.): Information Technology Security Evaluation Criteria (ITSEC). `http://www.bsi.de/zertifiz/itkrit/itsec-en.pdf`, download 2005-02-28 (1992).
8. Grimaud, Gilles and Vandewalle, Jean-Jacques: Introducing research issues for next generation Java-based smart card platforms. In: Proceedings of the Smart Objects Conference (SOC), Grenoble, France (2003), 138–141.
9. MAOSCO, Ltd: MULTOS. `http://www.multos.com`, download 2004-06-07 (2004).
10. MasterCard International: Mondex. `http://www.mondex.com`, download 2004-06-07 (2004).
11. Menezes, Alfred J.; van Oorschot, Paul C. and Vanstone, Scott A.: Handbook of Applied Cryptography. CRC Press, Boca Raton et al. (1997).
12. Niwano, Eikazu; Hatanaka, Masayuki; Hashimoto, Junko and Yamamoto, Shuichiro: Early Experience of a Dynamic Application Downloading Platform for Multi-Application Smart Cards. In: Proceedings of the Fifth Joint Conference on Knowledge-Based Software Engineering (JCKBSE) Maribor, Slovenia (2002).
13. Posegga, Joachim and Vogt, Harald: Byte Code Verification for Java Smart Cards Based on Model Checking. In: Proceedings of the Fifth European Symposium on Research in Computer Security (ESORICS), Louvain-la-Neuve, Belgium, Lecture Notes in Computer Science, 1485, Springer-Verlag, Berlin Heidelberg New York (1998), 175–190.
14. Rankl, Wolfgang and Effing, Wolfgang: Handbuch der Chipkarten: Aufbau - Funktionsweise - Einsatz von Smart Cards. Hanser Verlag, Munich et al. (2002).
15. Reiter, Michael K. and Rubin, Aviel D.: Crowds: Anonymity for Web Transactions. ACM Transactions on Information and System Security, No. 1, Vol. 1 (1998), 66–92.
16. Schneier, Bruce: Applied cryptography: protocols, algorithms and source code in C. John Wiley & Sons, Inc., New York et al. (1996).
17. Schneier, Bruce and Shostack, Adam: Breaking Up Is Hard To Do: Modeling Security Threats for Smart Cards. In: Proceedings of the USENIX Workshop on Smart Card Technology, USENIX Press (1999), 175–185.
18. Siveroni, Igor; Jensen, Thomas and Éluard, Marc: A Formal Specification of the Java Card Firewall. Nordic Workshop on Secure IT-Systems (2001).
19. Sun Microsystems, Inc.: Java Card Technology. `http://java.sun.com/products/javacard/`, download 2004-06-07 (2004).