A Generic Proxy for Secure Smart Card-Enabled Web Applications

Guenther Starnberger, Lorenz Froihofer, and Karl M. Goeschka

Vienna University of Technology
Institute of Information Systems
Argentinierstrasse 8/184-1
1040 Vienna, Austria
{guenther.starnberger, lorenz.froihofer, karl.goeschka}@tuwien.ac.at

Abstract. Smart cards are commonly used for tasks with high security requirements such as digital signatures or online banking. However, systems that Web-enable smart cards often reduce the security and usability characteristics of the original application, e.g., by forcing users to execute privileged code on the local terminal (computer) or by insufficient protection against malware. In this paper we contribute with techniques to generally Web-enable smart cards and to address the risks of malicious attacks. In particular, our contributions are: (i) A single generic proxy to allow a multitude of authorized Web applications to communicate with *existing* smart cards and (ii) two security extensions to mitigate the effects of malware. Overall, we can mitigate the security risks of Web-based smart card transactions and—at the same time—increase the usability for users.

Key words: Smart cards, Web applications, Digital signatures, Security

1 Introduction

Despite ongoing efforts to Web-enable smart cards [1] there is still a *media discontinuity* when using smart cards in combination with Web applications, as smart cards typically require a *native* helper application as proxy to communicate with the Web browser. One reason is that the Web security model is fundamentally different from the smart card security model, leading to potential security issues even for simple questions such as: "Is a particular Web application allowed to access a particular smart card?".

Ongoing research to Web-enable smart cards typically either requires computational capabilities at smart cards higher than the capabilities provided by today's smart cards or requires users to install software customized to particular types of Web applications [2]. In contrast, our generic mapping proxy enables access from arbitrary Web applications to arbitrary smart cards, while using access control to protect smart cards from malicious Web applications, without requiring any on-card software modifications.

However, guarding only against malicious Web applications is not sufficient, if the local computer is potentially controlled by malware. Consequently, we

enhance our mapping approach to provide end-to-end security between a user and a smart card, but this enhancement requires the possibility to adapt on-card software. In particular, we allow the user to (i) either use the TPM (Trusted Platform Module) inside her computer or, (ii) alternatively, use a trusted secure device to secure communication with the smart card.

Summarized, the contributions of our paper are:

- A smart card Web communication protocol that provides a secure way for Web applications to interact with existing smart cards. Unlike state-of-the-art technologies, our approach allows any Web application to interact with any given smart card where communication is allowed based on our authorization and access control mechanisms.
- A first extension to our protocol that uses the Trusted Computing facilities part of recent PC (Personal Computer) hardware. This allows us to mitigate the effects of malware on the local computer, but requires modification of the on-card software.
- A second extension to our protocol that uses QR-TANs (Quick Response – Transaction Authentication Number) [3] instead of a TPM. Thus, the security is provided by an external security device instead of a PC.

Section 2 discusses related work before Sect. 3 presents the architecture and trust model of our application. Sect. 4 introduces our generic Web mapping. Sect. 5 extends our mapping approach with TPM-based attestation, while Sect. 6 provides alternative security measures based on QR-TAN authentication. Finally, Sect. 7 concludes the paper and provides an outlook on future work.

2 Related Work

In this section we discuss related work to Web-enable smart cards as well as to improve client-side security.

Web-enabling smart cards. Itoi et al. [4] describe an approach for secure Internet smart cards that allows users to access remote smart cards over the Internet. In contrast, we provide the client-side part of a Web application running in a Web browser with access to smart cards at the local computer. Thus, the security assumptions and implementation details differ fundamentally. An expired IETF Internet draft for SmartTP by P. Urien [5] specifies a unique software stack applicable to different types of smart cards, but—unlike our approach—requires software support from the smart card. Hence, it is not applicable to legacy cards. The TLS-Tandem approach [6] seems to use smart cards for access control to a Web server, while we aim at Web-enabling smart cards to mitigate man-in-the-middle attacks. Further details on approaches to provide smart cards with network access can be found in [1].

Improving client-side security. Lu et al. [7] increase security with respect to online identity theft by placing confidential information inside the smart card from where it can be transferred to a remote authenticated server. This reduces

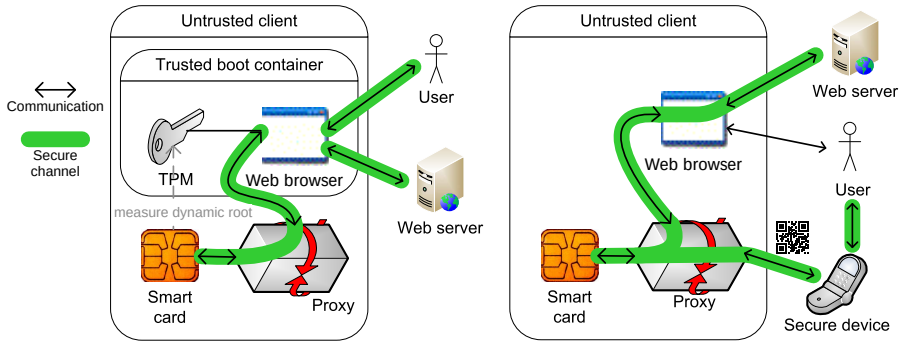
the risk of confidential information being captured by malware at a user's computer. However, it does not guarantee that data entered on the computer are not changed on the way to the server, which is the focus of our two security extensions. Bottoni and Dini [8] use a secure device to secure transactions between a user and a merchant. This is conceptually similar to our QR-TAN approach [3]. However, in this work we secure transactions between the user and the smart card itself. While our techniques rely on a trusted device or execution platform, Aussel et. al [9] include security-hardened monitors into applications running on untrusted platforms and use USB (Universal Serial Bus) smart cards to verify the log data provided by the monitors. Consequently, this approach could complement our techniques if no trusted execution environment is available, providing less security than a dedicated secure hardware device, of course.

Conclusion. Related work and existing implementations prove that connectivity of smart cards is a well researched topic. However, *real* Internet smart cards [10] able to communicate directly using the IP protocol are not yet widely available on the market. Furthermore, all existing evaluated solutions require custom on-card software for communication with the terminal. In comparison to existing work, our approach strives to partition trust requirements between Web applications and different smart cards and, additionally, features advanced security capabilities that allow to mitigate attacks due to insecure terminals. While Internet smart cards do not require our proxy application for Internet access, they do not provide equivalent capabilities for request filtering on the terminal. However, combining Internet smart cards with our two security mechanisms discussed in Sections 5 and 6 would allow to improve these cards' security in regard to man-in-the-middle attacks.

3 Architecture and Trust Model

This section presents our overall system architecture and the different security constraints. Due to the different trust requirements of the different entities, the problem we solve can be seen as a type of *multilateral security* [11] problem. For example, the user and the Web server both trust the smart card, but neither does the user trust executable code provided by the Web server, nor does the Web server trust executable code provided by the user. And while the user may place considerable trust into her own hardware, this hardware may not be trustworthy enough for the Web application in regard to non-repudiability requirements.

An overview of our architecture is given in Figure 1, which illustrates the major components and communication paths, but not the sequence of interactions detailed later. Figure 1(a) shows the architecture when used in combination with TPM and Figure 1(b) shows the architecture when used in combination with QR-TAN. The black arrows indicate direct communication paths between two entities while the highlighted broader lines in the background depict the secure channels in our system. If a secure channel spans several black arrows, this means that the intermediate entities are untrusted and data are passed through that entities by means of encryption or digital signatures. The trust relations



(a) TPM (b) QR-TAN

Fig. 1. System components

between the constituents described in the following paragraphs are depicted in Figure 2. Arrows labelled "high" indicate that a component is highly trusted, while "partial" indicates a lower trust relationship.

Web application and Web server. The Web application is an entity that wants to interact with the smart card; for example a banking site that requires a digital signature before conducting a transaction. The user trusts the Web application for communication with the smart card. However, the user does not trust the Web application with unrestricted access to her computer—e.g., to execute binary code obtained from the Web application. The server-side part of the Web application is running on the Web server, while the client-side part of the Web application is implemented in JavaScript and running on the Web browser. The term "Web application" refers to the combination of these two components.

Web browser. The Web browser is the entity used to interact with the smart card. It hosts the client-side part of the Web application and interacts with the server-side part of the Web application and the smart card. The user needs to trust the Web browser for the type of executed transaction. For low security transactions such as reading a stored-value counter, a normal Web browser can be used. For high security transactions, the trust in the Web browser can either be increased by executing the Web browser inside a *trusted environment* (see Sect. 5), or the trust requirements in the Web browser can be decreased by outsourcing part of the transaction to a trusted secure device (see Sect. 6).

Proxy. The proxy is our application responsible for mapping requests from a Web browser to a smart card. Combined with our generic mapping approach (Sect. 4), only a single generic proxy provided by a trusted vendor is required to be installed in order to allow access to smart cards from a multitude of authorized Web applications using state-of-the-art Web technologies. The trust requirements in the proxy are two-fold: From a user's perspective, the proxy is running on a semi-trusted platform as the proxy is started on her local operating system. Thus, some security features—such as controlling which type of APDUs

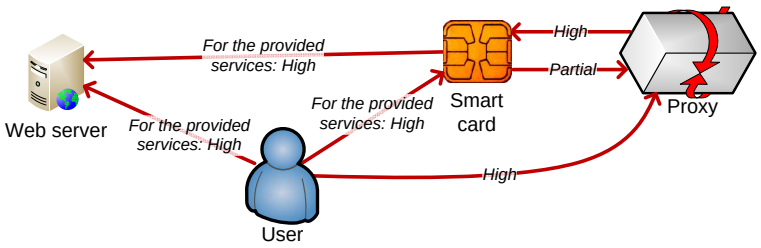


Fig. 2. Trust relations

(Application Protocol Data Unit) can be transmitted to smart cards are taken care of by the proxy. However, from a smart card issuers perspective the proxy is not necessarily trusted, as malware could control the computer. Thus, the smart card issuer can mandate additional security measures such as the authentication over a TPM (Sect. 5) or QR-TANs [3] (Sect. 6).

Smart card. A smart card is issued by an entity such as a local bank and is responsible for signing sensible data and/or for executing sensitive transactions. User and Web application have trust in the smart card's correctness. However, the user does not have a direct input and output path to the smart card. Thus, malware can manipulate the user's communication with the smart card.

4 Our Generic Proxy and Mapping Approach

Our generic proxy Web-enables smart cards without installation of custom on-card software and hence is applicable to a large range of existing smart cards. In sections 5 and 6, we enhance our approach to provide even better security in particular against malware for cases where the on-card software can be adapted.

The developed mapping technique uses a proxy to provide Web applications with access to smart cards and—in addition—protect the smart card from malicious Web applications. In contrast to existing techniques, our *generic mapping* allows a single executable to be used for a diverse set of Web applications and smart cards, not requiring the user to trust and execute code obtained from different Web sites. We validated our concepts with prototype implementations, showing that (i) the proxy concept is feasible in practice, (ii) correctly serves as a filter between Web applications and smart cards, and (iii) allows Web access to smart cards using state-of-the-art Web technologies.

Our system uses a mapping configuration to map abstract method calls to APDUs; an example is shown in Figure 3. This configuration defines how invoked methods with their arguments are mapped to APDUs and how results are mapped back to a structure. Furthermore, it includes a list of trusted origins, defining Web sites that may use the mapping, and a list of ATRs (*Answer To Reset*—an ATR *identifies* a smart card), identifying accessible cards. An AID (*Application Identifier*) identifies the respective smart card application. The mapping is cryptographically signed by the card issuer with a key certified by a Privacy CA (Privacy Certificate Authority).

```
<mapping>
  <smartcard atr="3b134028351180" aid="A000000062030l0C0202" />
  <method name="login">
    <request>
      <args><arg name="pin" type="STRING" /></args>
      <apdu_mapping is="D4"><argument name="pin" /></apdu_mapping>
    </request>
    <response />
  </method>
</mapping>
```

Fig. 3. Request mapping example

Mapping procedure. The following steps detail how a Web application can call a particular method defined in the mapping file. The interaction between the different components is shown in Figure 4.

1. The Web browser obtains a mapping definition and transmits the mapping to the proxy via an RPC (Remote Procedure Call) call by using JavaScript.
2. The proxy verifies that the origin of the client-side Web application part running in the Web browser matches the origin defined in the mapping file—e.g., by providing a secret to the Web application over a callback—and verifies the signature of the mapping.
3. The proxy verifies that there is a card in the reader and that the ATR of the card matches the ATR of the mapping. If the ATR is different, or if during the remaining process the ATR changes (e.g., because the card is replaced), the proxy will reset.
4. The proxy either asks the smart card if the public key used to sign the mapping should be trusted, or—alternatively—only verifies if the public key has been signed by a trusted certificate authority. If one of the options succeeds, the process is continued. Otherwise, the process is aborted.
5. The proxy receives RPC requests from the Web application's JavaScript code running inside the browser and converts them to APDUs according to the mapping. These APDUs are subsequently transmitted to the smart card. After the response APDU is received from the smart card it is converted and sent back to the application running in the Web browser.

Summarized, to protect smart cards from malicious Web applications we first identify the type of smart card connected to the PC. We proceed by verifying if the smart card provider has authorized the mapping file[1] provided by the Web application for this particular type of smart card. If this verification succeeds, this mapping is then used to restrict which Web applications can access the smart card and to restrict the type of APDUs that the Web application may transmit to the smart card.

[1] The authorization of a Web application's mapping file is an administrative task in contrast to the development and installation of on-card software, which would require re-distribution of smart cards.

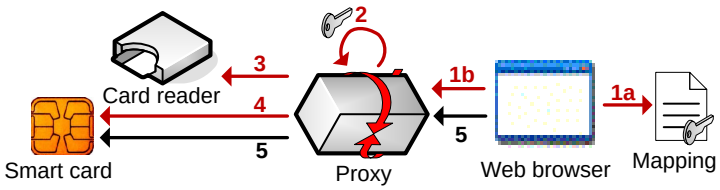


Fig. 4. Mapping procedure

5 Smart Card-Based TPM Attestation

This and the next section show extensions for establishment of a secure channel between the smart card and either (i) the Web browser running in a secure environment, or (ii) a mobile device trusted by the user. Both extensions require the support of on-card software.

The first approach discussed in this section uses an end-to-end security protocol between the smart card and the Web browser that provides authentication, integrity and confidentiality between the two endpoints. Our protocol works on a layer between APDU transmission and APDU interpretation. Conceptually, it can be compared to TLS. However, instead of desktop computers it targets smart cards and uses the remote attestation features of TPM that allow a remote party to verify if a computer is running a particular software configuration. The main requirements of our protocol are: (i) *Authentication*: Each party must be able to securely authenticate the other party. (ii) *Integrity*: Each party must be able to verify that the transmitted data has not been manipulated. (iii) *Confidentiality* (optionally): End-to-end encryption between the parties must be possible.

The first two items are required to prevent manipulation of transmitted data: Without authentication of the remote party, an attacker could directly establish a connection to one of the parties. Furthermore, without integrity of individual data items, an attacker could use a man-in-the-middle attack to manipulate these data items while in transit. The third item is optional: Without encryption, a man-in-the-middle is able to read transmitted information, but she is not able to manipulate this information. By using encryption, we can prevent an attacker from learning information about ongoing transactions.

In the following sections we first introduce the TPM functionality we use for remote attestation. Afterwards, we continue with a description of our secure channel that provides authentication, integrity and confidentiality. While a secure channel is already part of the Global Platform specification (`http://www.globalplatform.org/`), the specification assumes that there is a shared secret key between smart card and accessor. However, as we want to enable access to smart cards from different Web sites, a shared secret between the smart card and each individual Web site is infeasible.

5.1 Secure Computer Model

For the endpoint of our end-to-end security protocol on the local computer (see Figure 1(a)) we assume a computer model that allows to create a *secure runtime*

partition in which software is executed that cannot be accessed or manipulated by the user's (default) operating system. This secure partition hosts a browser instance used for communication with the smart card. Furthermore, the secure partition allows for remote attestation—allowing a remote entity to securely identify the executed software. To provide compatibility across different types of trusted environments, our model does not assume any further features. In particular, we do *not* assume that it is possible to open a secure channel to I/O devices such as smart card readers. This is in accordance with the current state of trusted environments, where applications can open secure channels only to some types of I/O devices such as monitors and keyboards [12, 13].

There are different technologies that allow for the creation of such a secure partition. One technology is Intel's *Trusted Execution Technology* (Intel TXT) that complements the functionality of a TPM by allowing a secure hypervisor to provide *virtual environments* that are protected from access by malicious applications. For remote attestation, a TPM can be used. The Xen hypervisor provides a *vTPM* [14] implementation that provides *virtual* TPM chips [15] to the executed instances. These virtual TPMs use features of the host's hardware TPM for the secure implementation of their different functions.

5.2 Establishing a Shared Secret for HMAC and Encryption

As basis for encryption and authentication we use a shared secret between smart card and Web browser running in a trusted environment. To establish this secret we use an authenticated Diffie-Hellman (DH) key exchange [16] as depicted in Figure 5. The variables g, p, A, B in the figure are Diffie-Hellman parameters. For authentication, we sign the parameter set sent by each of the parties with digital signatures that prevent man-in-the-middle attacks. On the smart card we use an asymmetric key that is certified by the smart card manufacturer, while on the TPM we use the AIK (Attestation Identity Key) for authentication.

Instead of using Diffie-Hellman, it would also be possible to generate the symmetric key on one of the endpoints and use asymmetric encryption to transfer this key to the other endpoint. However, with such a method the long-term security of the communication would depend on the security of the particular asymmetric key. If the asymmetric key would be broken, each symmetric key encrypted with this key in the past would be compromised. With Diffie-Hellman on the other hand, an attacker needs to crack each session key individually.

5.3 Mutual Authentication and Integrity

Mutual authentication allows each endpoint of a conversation to authenticate the identity of the opposing endpoint. Authentication and integrity are intertwined concepts: When endpoint authentication is used without data integrity, an adversary can exchange data while in transit. Likewise, if data integrity is used without endpoint authentication, an endpoint knows that the data have not been modified, but does not know the identity of the remote endpoint.

In this section we provide our approach for authentication and integrity. There are two endpoints (see Figure 1(a)): The smart card and the Web browser

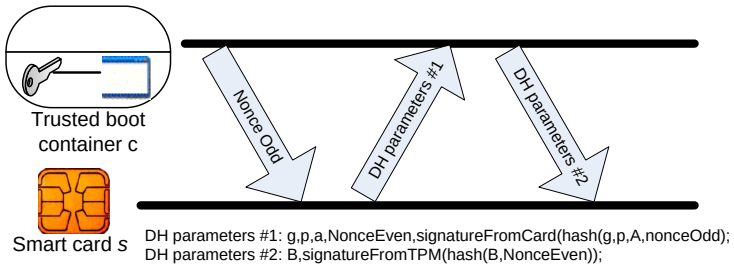


Fig. 5. Exchange of Diffie-Hellman parameters for secure channel. A nonce is used to prevent replay attacks. Each endpoint transfers cryptographically signed DH parameters to the other endpoint. These parameters are then used to establish the key for the secure channel.

running in a trusted environment. Authentication uses authenticated Diffie-Hellman key exchange, while integrity uses HMACs (Keyed-Hash Message Authentication Codes).

Each smart card stores a custom key pair generated on initialization and digitally signed by the smart card issuer. When transmitting Diffie-Hellman parameters, the smart card signs them with its private key and appends the public key together with the certificate of the issuer to the data structure as shown in Figure 5.

On the PC, the `TPM_Quote` command of the TPM is used to sign the content of particular PCRs (Platform Configuration Registers) that contain measurements of the executed software used to identify a particular software configuration. When PCRs are updated, the TPM combines the existing value with the new value, thus software running on the computer is not able to set the registers to arbitrary values. By cryptographically signing the values within the registers, the TPM chip can attest the state of the system to a remote system. This functionality is also called *remote attestation*.

The certification of the TPM's key is more complex than in the case of the smart card. Figure 6 shows the certification and attestation procedure, illustrating which entity certifies which other entity to build up a chain of trust that allows the smart card to verify a correct software execution environment. The TPM contains two different types of keys: The unmodifiable EK (Endorsement Key) generated during production and certified by the TPM's manufacturer and a modifiable AIK used for attestation. A Privacy CA (Privacy Certificate Authority) with knowledge about EK and AIK is responsible for certifying the AIK [12,13]. During attestation, the AIK signs a set of values that identifies the executed software. The smart card compares these values with a set of reference values identifying a particular browser appliance and signed by a trusted party (e.g., the smart card issuer).

5.4 APDU Encryption and Authentication

For the encryption and authentication of APDUs we use a protocol similar to smart card *secure messaging* defined in ISO/IEC 7816-4. The main reason why

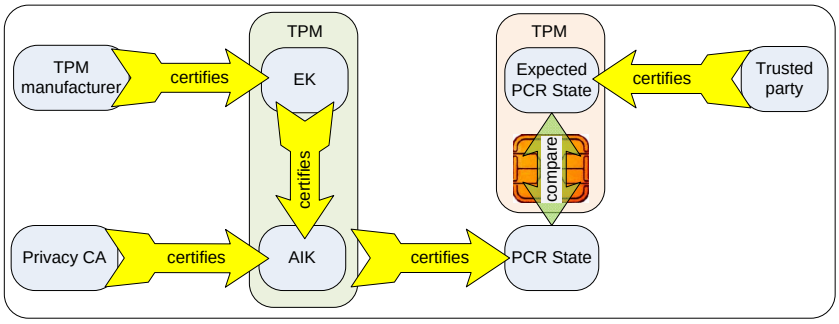


Fig. 6. Certification and attestation procedure

we cannot directly use secure messaging is that secure messaging requires a shared key between smart card and terminal. However, in our application scenario, this is not feasible as we want to enable secure communication with the smart card from a wide range of Web applications.

In theory, it would be possible to use the secret we established in Section 5.2 as the basis of a ISO/IEC 7816-4 compliant communication established directly by the smart card's operating system. However, as common smart card operating systems do not allow for access to the particular layer by client applications, this option is not possible. Instead, we re-implement comparable functionality inside the application layer, transmitting secured data as payload in APDUs. Therefore, APDU encryption and authentication allows us to establish a secure channel between smart card and Web browser.

APDUs are encrypted and authenticated by calculating *encrypt(session_key, hmac(session_key, orig_apdu + counter) + orig_apdu + counter)* using a symmetric encryption protocol such as AES (Advanced Encryption Standard). If no encryption is required, using an HMAC without encryption is possible. The HMAC serves to detect manipulation attempts of the APDU. The counter allows to detect replay attacks: In the beginning, a counter value derived from the shared key is used. As the session proceeds, each endpoint increases the counter value by one for each request and each response. Furthermore, each endpoint can detect if the received counter value matches the expected counter value. As a side effect, the counter also acts as a type of *initialization vector* (IV), as two equal APDUs encrypt to two different cipher texts.

5.5 Security Discussion

One issue with the certification of the PCR state by the smart card is that if the Privacy CA only certifies that the AIK belongs to *any* valid TPM implementation, it would be sufficient for an adversary to obtain the private key of *any* certified TPM to apply signatures. To cause a security problem the adversary would need to (i) break the user's environment so that the browser does not run inside a secure environment with the effect that malware has access to the software and (ii) to sign the TPM's side of the transaction with a certified key from another broken TPM implementation.

One option for the mitigation of such an issue would be for the Privacy CA to not only certify that the key belongs to any TPM implementation, but to further include a user identifier in the certificate. This certificate can then be used by the smart card to verify that the TPM belongs to an authorized user. Another option is to remember the first TPM key used for authentication and to only allow this particular key for future transactions. While this does not help against malware on a freshly installed PC, it protects the user against later attacks. However, to use another PC, a user would first need to obtain a certificate from the card issuer that instructs the smart card to reset the stored key.

When a secure channel is used, there is an important difference in the behavior of the intermediate proxy: In unencrypted communication, the proxy is responsible for filtering the requests, i.e., to only allow requests *whitelisted* in the mapping to pass. However, when encryption is used, such a filtering is not possible as the proxy does not have access to the plain text. Thus, the proxy cannot verify if a particular encrypted APDU is allowed by the mapping file. As the proxy cannot filter requests sent to smart cards in that case, smart cards need to be developed with the assumption that potentially *any* Web site can send requests. While a majority of smart cards is already designed to withstand external attacks, the optimal mitigation strategies in our scenario are different. Traditionally, a smart card does, e.g., deactivate itself, if large amounts of failed authentication attempts are detected. However, if any Web application can communicate with the smart card, a Web application could abuse such a behavior for a DoS (Denial of Service) attack: By deliberately causing failed authentication attempts, any Web application could disable the smart card.

As mitigation strategy, smart cards should not take any *destructive* actions in case of failed authentication attempts or other types of security alerts that can be caused by external Web applications. For example, instead of deactivating the smart card in case of multiple failed authentication attempts the smart card could just increase the minimum interval required between each authentication attempt. As an alternative, the on-card application responsible for the secure connection can filter requests according to the information in the mapping—and thereby accomplish the filtering task of the proxy.

6 Authentication with QR-TAN

This section presents the second option to increase the security of the proxy, by extending our system with QR-TANs [3]. QR-TANs are a transaction authentication technique that uses a secure device to allow users to securely confirm electronic transactions on remote systems. A user scans a two dimensional barcode containing information about a transaction with a secure device. The secure device allows the user to verify the transaction. To approve the transaction, the user transmits a TAN (Transaction Authentication Number) dynamically generated by the secure device to the remote system. In comparison to our original QR-TAN approach [3], our modifications allow the use of QR-TANs without the interaction of a server.

Conceptually, it is sufficient to replace the RTC (Remote Trusted Computer) in the original approach with a smart card. However, due to the different capabil-

ities of smart cards and servers, modifications to the original approach allow for better integration. In particular, our modifications address the following issues:

1. On smart cards, the generation of textual authentication requests intended for humans is more complicated than on servers. Especially as the smart card's memory restricts the amount of stored localizations and as it is rather complex to update the messages once the card has been issued.
2. The smart card should have the capability to decide if external transaction authentication is required. For example, in banking applications a smart card may allow daily transactions of up to a particular total value without authentication, only requiring authentication above that value.
3. Usage of QR-TAN should be transparent to applications using the proxy. Thus, only the smart card, the proxy, and the secure device should contain QR-TAN specific code.

To enable these properties, we extend our mapping description to contain information about QR-TAN authentication. In particular, we introduce a new `<auth />` section that describes which status words in the APDU response indicate that QR-TAN authentication is required and how human readable text is generated from the structure returned by the smart card. For authentication the following steps depicted in Figure 7 are used:

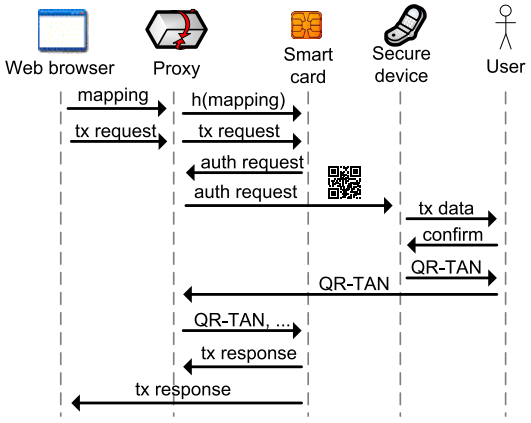


Fig. 7. QR-TAN authentication steps.

1. On initialization the browser provides the mapping to the proxy. The proxy transmits a hash and a certificate of the used mapping file signed by a trusted third party to the smart card. The smart card verifies that the certificate allows use of the given mapping file.
2. The Web browser sends its transaction request to the smart card.
3. The smart card responds with a particular status word indicating that QR-TAN authentication is required for this type of transaction. The data of the response contains a structure with information about the transaction, a nonce, and the secure hash of the used mapping file.

4. The proxy reads the response by the smart card and converts it to a QR code that is subsequently displayed to and scanned by the user's secure device.
5. The secure device first checks if it has stored the mapping file indicated in the QR code. If not, it prompts the user to install the mapping file—e.g., by scanning a compressed QR code containing the mapping file. Otherwise, it generates a human readable text according to the information in the mapping file and asks the user for confirmation.
6. If the user confirms, the secure device generates a QR-TAN over the data structure of the original request (in step 2), the nonce, and the hash of the mapping file and presents this QR-TAN to the user.
7. When the user provides this QR-TAN to the proxy, it generates an APDU with this information and sends it together with the nonce and the hash of the mapping file to the smart card. The smart card validates if the QR-TAN request for the particular nonce matches the hash of the information within the APDU.
8. In case of success, the smart card returns the response of the original APDU request issued in step 2.

The conversion of the transaction data to a human readable text can be done on either one of the two endpoints of the QR-TAN authentication: Inside the smart card or inside the trusted secure device. It is not possible to do this conversion on any device between these two endpoints as this would prevent from successful end-to-end authentication. In our approach we perform this conversion inside the secure device. While generating the text directly within the smart card would be conceptually simpler, it would require the smart card to store potentially large amounts of textual data—e.g., if multiple localizations are required. Furthermore, it is not easily possible to adapt the text once the smart card has been issued.

As the secure device uses a mapping file to convert the structure with information about the transaction to a textual format, it must ensure that the mapping file is also authenticated. Otherwise, an attacker would be able to send a manipulated mapping file to the secure device, causing the device to show incorrect transaction information to the user. It is not sufficient for the secure device to only validate if the mapping file has been signed by a trusted party: As the secure device cannot securely obtain the ATR (Answer To Reset) of the smart card, it cannot ensure that the mapping file belongs to a particular smart card. Instead, we include a hash of the mapping file in the structure that is hashed by the secure device, allowing the smart card to ensure that the QR-TAN belongs to a particular mapping. While the smart card does not need to know the content of the mapping file, it needs to know the digital signature to decide if it can trust the mapping. Thus, the proxy can send the signature of a mapping to the smart card via APDUs.

By integrating our QR-TAN approach directly with the proxy, the proxy is responsible for displaying the QR code and for forwarding the QR-TAN back to the smart card. Thus, the whole process can be transparent to applications using the mapping, as the only difference between authentication and non-authentication is the additional delay of the authentication process.

7 Conclusion and Outlook

We presented a secure approach for Web-based smart card communication. Our overall contributions are: (i) a secure technique using a single generic proxy to allow a multitude of authorized Web applications to communicate with *existing* smart cards and (ii) techniques for *new* smart cards that allow for secure end-to-end communication between a user and a smart card. In particular, our security extensions cover the usage of (i) a TPM and (ii) QR-TANs to secure communication with smart cards.

Especially with citizen cards recently introduced in several countries and with high security requirements in online banking, a secure solution for Web-enabled smart cards is required. Compared to related approaches, our system works with existing smart cards without requiring changes to on-card software. Thereby, we can increase the security of the user's system, by not requiring the user to install privileged software distributed by Web sites that require access to a smart card. Furthermore, in cases where it is feasible to adapt on-card software, we can increase the security over the state-of-the-art even further, as we can use the TPM or QR-TANs to secure transactions that would otherwise be affected by malware on the terminal.

Overall, we see that Web to smart card communication techniques are an area where further research is required. In particular, researching the possibilities to use state-of-the-art Web protocols for secure mashups [17] may provide viable results, allowing smart cards to use standard Web protocols to identify and authorize Web applications. However, with increasing usage of Web technologies in smart cards also new kinds of attacks against smart cards are viable, as malicious applications can now target the Web browser to gain access to the smart card. Therefore, end-to-end security techniques are required to allow smart cards to mitigate the risk of such attacks. Additionally, new approaches [18] in automatic generation of network protocol gateways can allow for the more efficient generation of Web to smart card mapping files.

Concluding, our research can serve as basis for a newer, more secure generation of smart card to Web communication. By combining the security features of smart cards with the features provided by TPMs and QR-TANs, we can mitigate the effects of the terminal problem [19] as the smart card is able to assert that the transaction data has not been manipulated. While future smart card generations may require modifications to the specific techniques introduced in this paper, the overall approach will still be applicable. Furthermore, recent developments such as the Trusted Execution Module (TEM) [20] allow for the implementation of more powerful request mapping approaches on smart cards.

Acknowledgments. The authors would like to thank Markus Wilthaner for the proof-of-concept prototype implementation of the work described in this paper. This work has been partially funded by the Austrian Federal Ministry of Transport, Innovation and Technology under the FIT-IT project TRADE (Trustworthy Adaptive Quality Balancing through Temporal Decoupling, contract 816143, `http://www.dedisys.org/trade/`).

References

1. Lu, H.K.: Network smart card review and analysis. Computer Networks **51**(9) (2007) 2234–2248
2. Leitold, H., Hollosi, A., Posch, R.: Security architecture of the austrian citizen card concept. In: ACSAC, IEEE Computer Society (2002) 391–402
3. Starnberger, G., Froihofer, L., Goeschka, K.M.: QR-TAN: Secure mobile transaction authentication. In: Availability, Reliability and Security, 2009. ARES '09. International Conference on, Fukuoka (March 2009) 578–583
4. Itoi, N., Fukuzawa, T., Honeyman, P.: Secure internet smartcards. In Attali, I., Jensen, T.P., eds.: Java Card Workshop. Volume 2041 of Lecture Notes in Computer Science., Springer (2000) 73–89
5. Urien, P.: Smarttp smart transfer protocol. Internet Draft (June 2001)
6. Urien, P.: TLS-tandem: A smart card for WEB applications. In: 6th IEEE Consumer Communications and Networking Conf. CCNC 2009. (January 2009) 1–2
7. Lu, H.K., Ali, A.: Prevent online identity theft - using network smart cards for secure online transactions. In Zhang, K., Zheng, Y., eds.: ISC. Volume 3225 of Lecture Notes in Computer Science., Springer (2004) 342–353
8. Bottoni, A., Dini, G.: Improving authentication of remote card transactions with mobile personal trusted devices. Computer Communications, Elsevier **30**(8) (2007) 1697–1712
9. Aussel, J.D., d'Annoville, J., Castillo, L., Durand, S., Fabre, T., Lu, K., Ali, A.: Smart cards and remote entrusting. In: Future of Trust in Computing, Vieweg+Teubner (2009) 38–45
10. Márquez, J.T., Izquierdo, A., Sierra, J.M.: Advances in network smart cards authentication. Computer Networks **51**(9) (2007) 2249–2261
11. Rannenberg, K.: Multilateral security a concept and examples for balanced security. In: NSPW '00: Proceedings of the 2000 workshop on New security paradigms, New York, NY, USA, ACM (2000) 151–162
12. Müller, T.: Trusted Computing Systeme. Xpert.press. Springer (2008)
13. Challener, D., Yoder, K., Catherman, R., Safford, D., Van Doorn, L.: A practical guide to trusted computing. IBM Press (2007)
14. Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th USENIX Security Symposium, USENIX (August 2006) 305–320
15. England, P., Löser, J.: Para-virtualized tpm sharing. In Lipp, P., Sadeghi, A.R., Koch, K.M., eds.: TRUST. Volume 4968 of LNCS., Springer (2008) 119–132
16. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Transactions on Information Theory **IT-22**(6) (November 1976) 644–654
17. Hammer-Lahav, E., Cook, B.: The oauth core protocol. Internet Draft draft-hammer-oauth-02 (March 2009)
18. Bromberg, Y.D., Réveillère, L., Lawall, J.L., Muller, G.: Automatic generation of network protocol gateways. In: Middleware 2009. Volume 5896 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2009) 21–41
19. Gobioff, H., Smith, S., Tygar, J.D., Yee, B.: Smart cards in hostile environments. In: WOEC'96: Proc. of the 2nd USENIX Workshop on Electronic Commerce, Berkeley, CA, USA, USENIX Association (1996) 3–3
20. Costan, V., Sarmenta, L.F.G., van Dijk, M., Devadas, S.: The trusted execution module: Commodity general-purpose trusted computing. In Grimaud, G., Standaert, F.X., eds.: CARDIS. Volume 5189 of Lecture Notes in Computer Science., Springer (2008) 133–148