Secure Messaging, Key management

Preparation for lecture:

Design your own protocol for mutual authentication and secure message transmission. You do not need to actually implement the protocol. However, you have to describe it on very detailed level (e.g., what mode of cipher is used, how many blocks are encrypted, what padding is used, when the protocol should abort due to incorrect values, …) and you can use only simple cryptographic primitives available on smart card (e.g., you can use DES cipher, SHA-1 hash function or random generator but not the OPSystem.getSecurityDomain() secure messaging object).

You protocol must provide:

- Mutual authentication between smart card and PC application based on pre-shared symmetric cryptography secret.
- Secure message exchange after authentication – all subsequent commands send to and from smart card after authentication must be confidential and integrity protected.

Few things you should keep in the mind:

- It is not a good idea to use long-term secrets to directly protect ordinary communication.
- Be aware of replay attack.
- What block cipher mode are you using, how the IV is, what type of padding is used.
- How the integrity is protected, is the apdu header and response protected as well?

You have 2 weeks to complete this task. Submit your description of the protocol (informal language, but detailed description what primitives were used, why and what threat are mitigated by the construction – should be around 1-2 A4 text) into IS before 9.12. and prepare short presentation (5 minute max.) for 10.12. (resp. 11.12.) lesson. Your design will be discussed and "attacked" by your classmates.

Global Platform

Publicly available specifications [GP03] for smart card managements covering issues of smart card life cycles, installation of applets, remote card management and secure communication between smart card and user application.

a. Card Manager and Security domain

The Card Manager is the card component responsible for all card administration and card system service functions:

Command Dispatch:
- Application selection
- (Optional) Logical channel management
- Command dispatching

Card Content Management
- Content verification
- Content loading

- Content installation
- Content removal

Security Management
- Security Domain locking
- Application locking
- Card locking
- Card termination
- Application privilege usage
- Security Domain privilege usage
- Tracing and event logging

The Issuer Security Domain is card component (more independent security domains can be present) that contains keys that the Card Issuer uses in support of cryptographic operations for the Card Issuer's Applications. These Security Domains are privileged applications established on a GlobalPlatform card to represent Application Providers who require a level of key separation from the Card Issuer.
Security domain is used when some card management operation is required after the card issued to card holder (e.g. upload and installation of applet, card locking, …). Keys carried by this Security domain are used to verify authenticity and integrity of request (Load File) and to provide confidentiality of transferred data. A Load File may contain one or more DAP (Data Authentication Pattern) Blocks that allow an entity other than the loading entity to verify the authenticity and the integrity of the Load File Data Block.

b. Smart card life cycles

The smart card passes various logical life cycle states between manufacture and final destruction. These life cycle states define which operations can be performed with the card. The following card Life Cycle States shall apply:

1. **OP_READY** –card is ready for uploading of key diversification data, any application and issuer specific structures.
2. **INITIALIZED** – card is fully prepared but not yet issued to card holder.
3. **SECURED** – card is issued to card holder. Card management is possible only throw Security domain in secure sense (installation of signed applets etc.).
4. **CARD_LOCKED** – card is locked due to some security policy and no data management can be performed. Card can be locked by Security domain and later unlocked as well (switch back to SECURED state).
5. **TERMINATED** – card is logically "destroyed" due to card expiration or detection of the severe security thread.

The card Life Cycle States OP_READY and INITIALIZED are intended for use during the Pre-Issuance phases of the card's life. The states SECURED, CARD_LOCKED and TERMINATED are intended for use during the Post-Issuance
phase of the card although it is possible to terminate the card at any point during its life.

c. GP APDU commands

Following APDU commands are defined to manage content of the smart card:

- **DELETE** – delete uniquely identifiable object (e.g. JavaCard applet)
- **STORE_DATA** – upload content of single data object
- **GET_DATA -** used to retrieve a single data object
- **SET_STATUS –** set Life Cycle status
- **GET_STATUS –** return Life Cycle status
- **INSTALL –** initiate installation, typically (JavaCard) applet
- **LOAD –** upload file from PC to smart card, e.g. JavaCard cap file
- **PUT_KEY –** update value of specified key

d. Secure Messaging – Secure channel protocol

Secure messaging stands for the process, which should lead to mutual authentication of the smart card and PC and optional creation of the authenticated encrypted tunnel between smart card and PC. Details of the process are described in Open Platform specifications [GP03].

Mutual authentication and computation of session keys consist from two phases (two APDU commands exchanged with smart card):

1. INITIALIZE UPDATE
2. EXTERNAL AUTHENTICATE

First phase (INITIALIZE UPDATE): PC application sends 8-bytes block of random data to smart card (called host random or host challenge). Smart card generates its own 8-bytes random block (card random/challenge). Using host and card challenge forms derivation data and encrypt them using card specific static encryption key (static ENC key - created during key diversification process). Result of this operation is the session key (session ENC key).

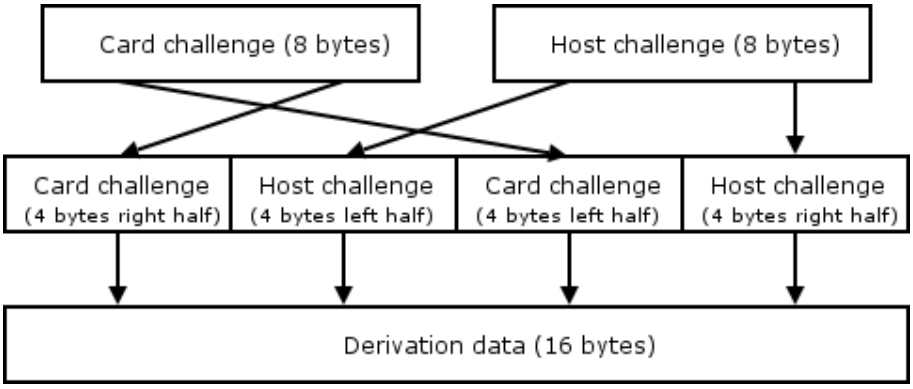

figure 1 Generation of the derivation data [GP03]

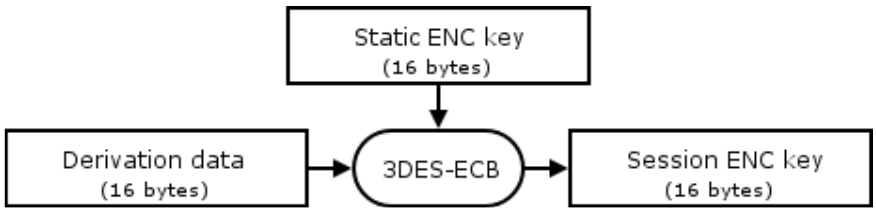

figure 2 Generation of session ENC key [GP03]

Session key is used for computation of the authentication *card cryptogram. Cryptogram* is computed using MAC operation. Input data are concatenated *card challenge* and *host challenge* padded by the block ('80 00 00 00 00 00 00 00'). Cryptogram is send together with card challenge back to PC.

Second phase (EXTERNAL AUTHENTICATE): PC application checks *card cryptogram* send by card and computes its own cryptogram (*host cryptogram*). Algorithm for the host cryptogram computation is again based on 3DES in CBC mode.

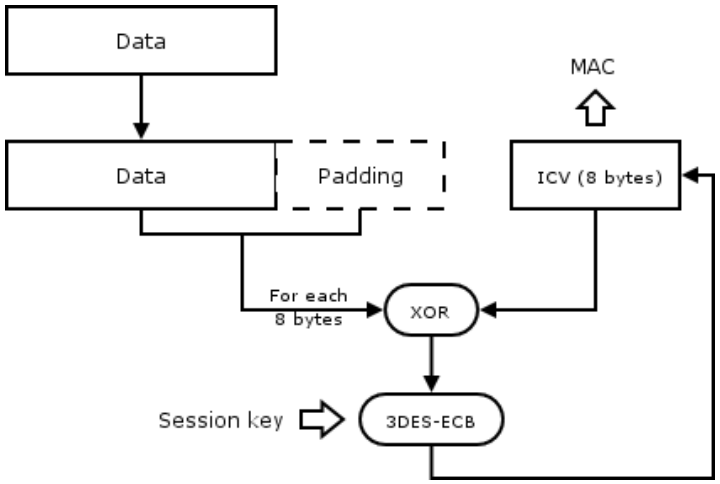

figure 3 Algorithm for MAC computation [GP03]

In case of the equivalence of the received and computed *card cryptograms*, *host cryptogram* is send back to card together with MAC of whole APDU command (EXTERNAL AUTHENTICATE).

Algorithm for host cryptogram is same with swapped host and card challenge.

Session encryption and MAC keys are then used to create confidential authenticated tunnel between smart card and PC application for all subsequent APDU commands.

Secure channel

Secure channel is vital component for the most of the security related applications that involves smart cards. Basically, it should provide establishment of robust channel with confidentiality and integrity between smart card (user applet, Card Manager) and second entity (typically PC application, server-side application). Various issues should be taken in to account.

Usage scenario and expected attackers

The general analysis of the system and appropriate properties of the secure channel should be based on the usage scenario and the abilities of the expected attackers. Following questions should be

1. What are the sensitive objects (keys, data, functions)?
2. What are these sensitive objects used for and what is the data flow of these objects?
3. What are the capabilities of the attackers (funding, tools, knowledge)?

4. What are the points where an attacker can observe the system (dump of exchanged messages, debugging, ...)?
5. Which parts of the system must be trusted to obtain required functionality (less the better)?

Confidentiality and integrity of command data

Data exchanged between smart card and PC can be intercepted and modified on binary level in various levels (USB, PC/SC). Therefore, exchanged data should be encrypted and integrity-protected by cryptographic manners.

- Both incoming and response apdu should be protected.
- Integrity should protect also apdu header, not only data part.
- Key for encryption and MAC computation should be different otherwise potential threat from data mismatch exists.
- Header semantics should be protected as well (see Resilience against traffic analysis).
- Long term secrets should be used only for session keys derivation and not for ordinary usage.
- Proper response in case of corruption – when corrupted command is detected, channel should be closed and relevant information should be stored into on-card security log.
- Vadenay's attack against padding should be taken in account – V. attack allows to decrypt data without knowledge of respective keys, if the information about data correctness from decryption oracle is available to an attacker.

Replay attack

Defense against the replay attack is an important measure, as an attacker can usually easily capture, block and replace commands between card and PC. Therefore, same apdu command (on encrypted level) received two times must be rejected and logged. The protection measure must be able to detect replay a) within actual session, b) between different parallel sessions and c) from previous sessions. Both incoming and response apdu should be protected. The replay attack can be prevented by the combination of the derivation of unique session keys and freshness nonce. Both card and PC must contribute into resulting derivation data, otherwise one party can force session keys to be same by setting same derivation data. Freshness nonce can be either sequence counter or (probably better) hash stream starting from derivation data and updated each time a new command is send (both incoming and response).

Atomicity of critical operations

The critical operations should be atomic both on code level (e.g., two variables are either changed both or none of them – use JCSystem.beginTransaction()) and on logical operations level (e.g., if command should either encrypt or decrypt mode supplied data with specified key than mode, key and data should be ideally send within same command). When using transactions, be aware of Rollback attack, where an attacker cut power inside the transaction – all changes are then restored back to original values including the security usage counters – which might be undesirable.

Robustness against side channel attacks (power analysis, fault attacks)

The side channel attacks prove to be a severe threat against the security of smart cards. The threat mitigation involves both the card manufacturer (cryptographic coprocessor) and

developer of applet. Typically, if the coprocessor is flawed and can be attacked by the assumed attacker (cryptographic keys can be extracted) than this version of the card should not be used (even when techniques that are able to limit attackers success still might exists – e.g., key derivation technique proposed by Kocher, limitation of number of packets encrypted by the same key etc.). Second layer that can be attacked is the applet code itself and data that are manipulated. Typically, single instructions of the processor can be observed and thus reverse engineering can be employed by an attacker to obtain parts of original code back.
To prevent fault attacks, operation might be executed twice and results should be compared. If the results mismatch than nothing should be given on output. CRT version of RSA should not be used as is highly vulnerable by the fault attack.
Measures against side channel attacks might be quite specific for given card – therefore the particular measures should not confuse original code – the clarity and correctness of the code before the measures are applied is very important! Write nice and correct code first and then add reasonable measures against side channel attacks.

Robustness against incorrect attempts

Depending on the usage scenario, data send to and received from the card might be corrupted either by transmission error or by an attacker. During mutual authentication, the challenge should change next time if the response cryptogram is not correct (do not allow multiple tries for same challenge). The secure channel should be closed when unexpected data are received.

Resilience against traffic analysis

Length of commands, frequency etc. can help an attacker a lot when building mental model of applet/protocol functionality. As the dump of the communication between valid card and application is relatively easy to obtain (USB dumper, PC/SC logger, virtual reader), transcription of the correct communication provides lot of information about process, even when the data itself are encrypted.
Following defense measures can be employed:

1. Internal semantics of the commands can be hidden into uniform structure with fixed length (maximum length of the command) and same header. The differentiation between logical functions inside applet are then given by nested internal header that is stored in encrypted part of apdu command (basic apdu header itself cannot be encrypted). An attacker without knowledge of encryption keys thus sees only the commands with same length and basic header.
2. Still an attacker can induce some knowledge from the order of the commands. E.g. he can assume that first two commands provide mutual authentication and third one the “secret” function he likes to attack. To counter this threat, dummy traffic must be introduced. Parameter in nested header will carry the information if the command is valid or dummy one and card will behave respectively. If an attacker is able to use power analysis to observe applet behavior, the dummy command should induce the same sequence of instructions as the valid one – but over the dummy internal values.

- HMAC or OMAC instead of hash or simple keyed hash
- Include command header into MAC
- Pre-share two keys (encryption, mac) or derive from master instead of one only

- Use pre-shared keys only to derive session keys. Session keys are used than to generate cryptograms tec.
- All parts of resulting session keys must be dependent on both contributions from PC and SC. One party cannot force resulting key into specific value.
- Replay attack – hash chain better than counter only
- Be aware of impossibility of verification, when random value is decrypted.
- MAC(ENC(data))
- Close channel on error
- use CBC rather than ECB
- be aware of cut attack in CBC
- be aware of block exchange in ECB mode
- do not use XOR for combination of values – use HMAC instead
- do not use symmetric protocol messages (PC->SC cannot be send SC->PC)

[GP03] GlobalPlatform, http://www.globalplatform.org/showpage.asp?code=specifications