Smart Cards

Towards a modern run-time platform

4. Card Management

Thorsten Kramp & Michael Kuyper
IBM Zurich Research Laboratory

Card Management

- The card issuer traditionally controls the card
  - *personalisation, applet loading/installation/removal, life cycle, …*
- But: How about multiple application providers?
  - *the card issuer stays in control but wants to share control of some of its card space with business partners*

    ☛ *business partners must be allowed to manage their own applications on the card issuer's cards in a (cryptographically) controlled way*

Overview

A. Global Platform

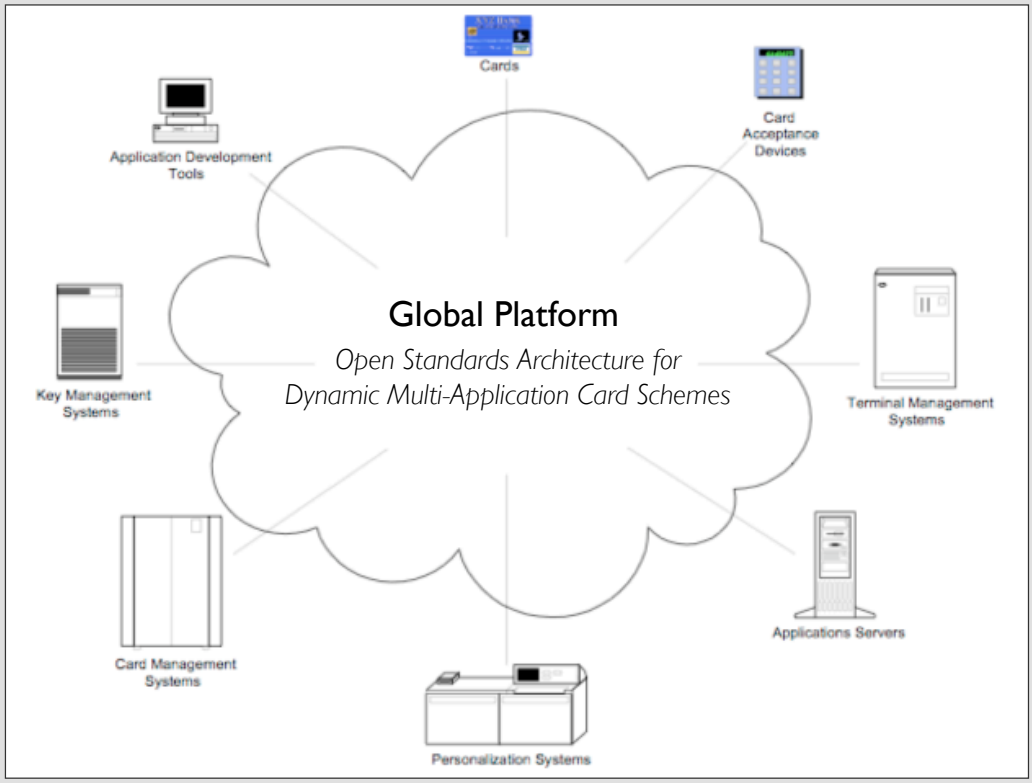
card manager, card and applet life cycles, security domains, delegated mgmt, secure comm., owner PIN

A. GP: Global Platform Standard

- consortium with over 50 leading organisations (payments, telecommunications, government, technology)
- specifies a common security and card management architecture
  - *hardware-neutral*
  - *vendor-neutral*
  - *application-independent*

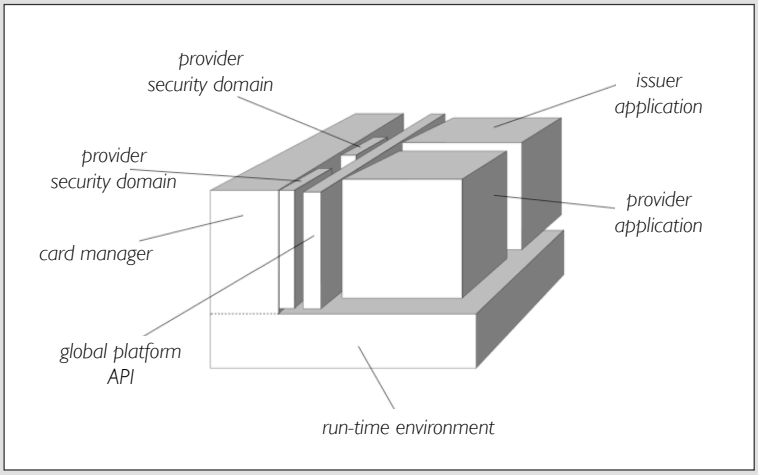
GLOBALPLATFORM

A. GP: Overview



A. GP: Components

- Card Manager
  - *on-card representative of the card issuer*
- Security Domains
  - *on-card representative of an application issuer*



A. GP: The Card Manager

- On-card representative of the card issuer
- Overall system and security controller
  - *behaves and functions as an application*
  - *external APDU interface for off-card management systems*
  - *on-card Java programming API for applications*
- Functionality
  - *APDU command dispatching and application selection (incl. itself)*
  - *card content management (e.g., application loading/installation/removal)*
  - *global PIN management*
  - *includes the card-issuer's security domain as a sub-component*

A. GP: Security Domains

- On-card representative of an application provider
- Key management application
  - *provides cryptographic services for all the applications owned by a particular application provider*
  - *established on behalf of the application provider by the card issuer*
- Load integrity via data authentication pattern (DAP) verification
  - *one or more DAP blocks in the load file*
  - *each DAP block is associated with one security domain*
  - *all DAP blocks are sequentially verified prior to completing the load process*
  - *DAP verification is mandatory if required by one or more security domains*

A. GP: Delegated Management

- allows a card issuer to provide an application provider the ability to perform specific content management
  - *delegated loading*
  - *delegated installation*
  - *delegated deletion*
- occurs through the use of the application provider security domain
  - *card manager security domain DAP blocks for load and install ensures that the card issuers has authorized the operation*
  - *card manager generates Load Receipt, Install Receipt, or Delete Receipt in response to be returned to the card issuer*

A. GP: Secure Channel

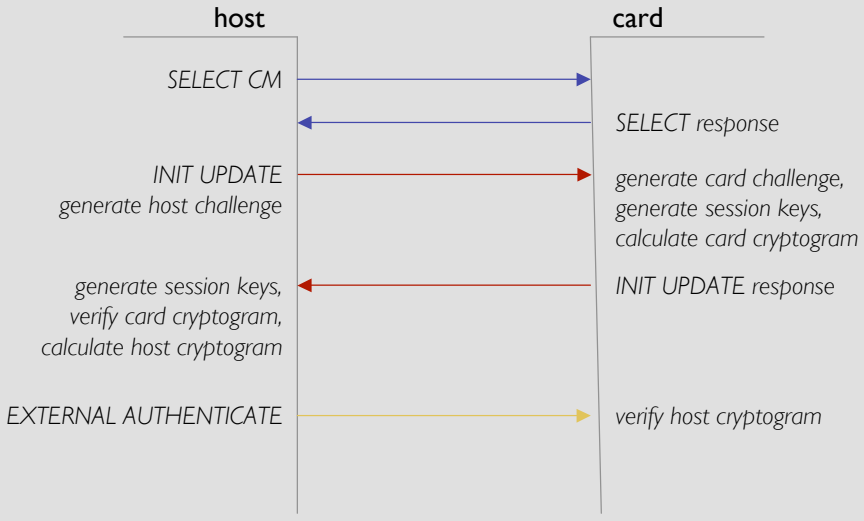
- provides secure communication between the card and an off-card entity (used by the card manager and security domains)
- 3 levels of security
  - ***mutual authentication:*** *the card and the off-card entity each prove that they have knowledge of the same secrets*
  - ***integrity and authentication:*** *the card ensures that the data being received from the off-card entity actually came from an authenticated off-card entity in the correct sequence and has not been altered*
  - ***confidentiality:*** *data being transmitted from the off-card entity to the card is not viewable by an unauthenticated entity*

A. GP: Secure Channel

- Mutual authentication
  1. *initially, the off-card entity passes a "host challenge" (random data unique to this session) to the card*
  2. *the card then generates its own "card challenge"*
  3. *using the host challenge, the card challenge and its internal static keys, the card creates new "session keys" and generates a "card cryptogram" using one of its newly created session keys*
  4. *the card cryptogram along with the card challenge and other data is transmitted back to the off-card entity*
  5. *the off-card entity should be able to generate the same card cryptogram and by performing a comparison, it is able to authenticate the card*
  6. *the off-card entity now uses a similar process to create a "host cryptogram" to be passed back to the card*
  7. *again, the card now should be able to generate the same host cryptogram and, by performing a comparison, it is able to authenticate the off-card entity*

A. GP: Secure Channel

- Mutual authentication flow (card manager)



A. GP: Secure Channel

- Integrity and authentication
  - *provides a means for the card to verify that commands*
    - are received from an authenticated off-card entity
    - have not been altered
    - have only been transmitted in sequence
  - *data integrity*
    - multiple chained DES operations (using a session key generated during the mutual authentication process) across the header and data field of an APDU command to generate a MAC (Message Authentication Code)
  - *sequence integrity*
    - using the MAC from the current command as the Initial Chaining Vector (ICV) for the subsequent command

A. GP: Secure Channel

- Confidentiality
  - *provides a means for data to be transmitted across an open network without the fear of the data being intercepted for the purpose of analysis*
  - *operation*
    - multiple chained DES operations (using a session key generated during the mutual authentication process) across the data field of the command message
    - no relationship between multiple messages (i.e., there is no chaining requirement across messages)
    - the data integrity MAC applies to the clear text data

A. GP: `visa.openplatform`

- `visa.openplatform.OPSystem`
  - *exposed subset of the card manager*
- `visa.openplatform.ProviderSecurityDomain`
  - *provides secure messaging for applets*

A. GP: `visa.openplatform`

- `visa.openplatform.OPSystem`
  - *exposed subset of the card manager*
- `visa.openplatform.P`
  - *provides secure messaging*

```
public class OPSystem {
   public static ProviderSecurityDomain
      getSecurityDomain();

   ...
};
```

A. GP: `visa.openplatform`

- `visa.openplatform.OPSystem`
  - *exposed subset of the card manager*
- `visa.openplatform.P`
  - *provides secure messaging*

```
public interface ProviderSecurityDomain {
   public byte openSecureChannel(APDU apdu);
   public void closeSecureChannel(byte channel);

   public void verifyExternalAuthenticate
      (byte channel, APDU apdu);
   public void unwrap(byte channel, APDU apdu);

   ...
};
```


A. GP: Secure Messaging Example

```
ProviderSecurityDomain domain;
byte channel;

public void process(APDU apdu) {
   ...
   byte[] buf = apdu.getBuffer();
   apdu.setIncomingAndReceive();

   switch (buf[ISO7816.OFFSET_INS]) {
    case VOP_INITIALIZE_UPDATE:
      domain = OPSystem.getSecurityDomain();
      channel = domain.openSecureChannel(apdu);
      apdu.setOutgoingAndSend(ISO7816.OFFSET_CDATA,buf[ISO7816.OFFSET_LC]);
      return;
    case VOP_EXTERNAL_AUTHENTICATE:
      if (domain == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
      domain.verifyExternalAuthenticate(channel,apdu);
      return;
    case PERSONALISE:
      if (domain == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
      domain.unwrap(channel,apdu);
      ...
      return;
   }

   ...
}
```

A. GP: Global PIN

- optional mechanism for card holder verification
- card manager provides a card global PIN service
  - *manages the PIN retry limit and PIN retry counter*
  - *performs PIN verification*
  - *a single PIN across all applications*

PIN Format

2 L	P P	P P	P/F	P/F	P/F	P/F	FF

L	*PIN length (4-12 digits)*
P	*digit in the range* `0..9`
P/F	*either* P P, P `F`, or `FF`

A. GP: `visa.openplatform`

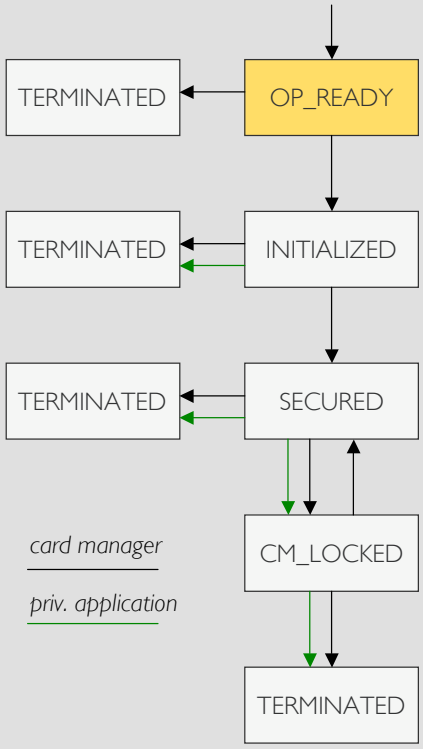
- `visa.openplatform.OPSystem`
  - *exposed subset of the card manager*

```
public class OPSystem {
   public static byte getTriesRemaining();
   public static boolean setPIN(APDU apdu, short ofs);
   public static boolean verifyPIN(APDU apdu, short ofs);

   ...
};
```

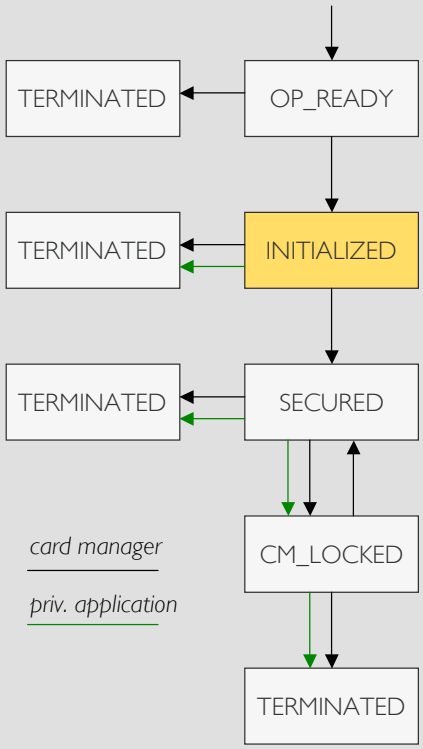
A. GP: Card Manager Life Cycle

- OP_READY
  - *run-time environment is ready for execution*
  - *card manager acts as selected application*
  - *applets in ROM are available*
  - *an initialization key is available within the CM*



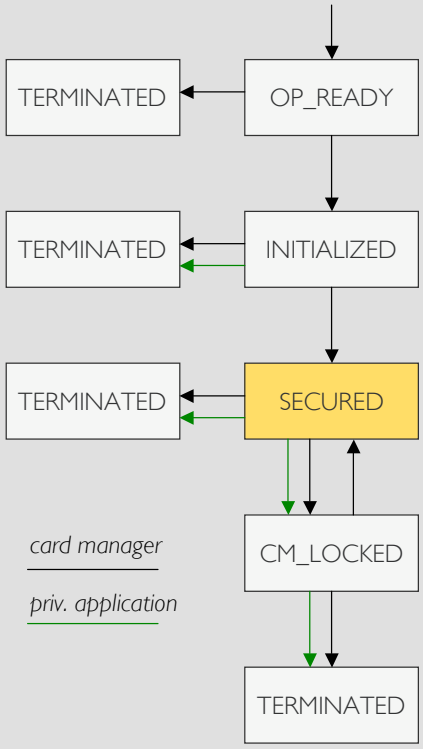
A. GP: Card Manager Life Cycle

- INITIALIZED
  - *administrative production state*
  - *exact definition is manufacturer, issuer and/or implementation dependent*
  - *some initial data has been populated (e.g., keys and card manager data) but the card is not yet ready to be issued to the cardholder*



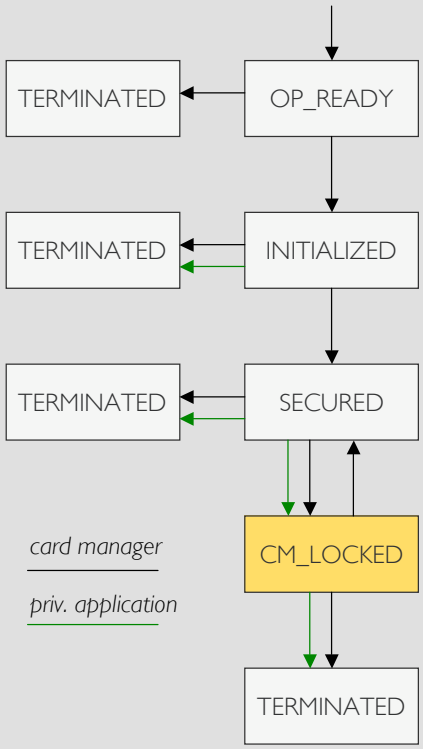
A. GP: Card Manager Life Cycle

- SECURED
  - *the card manager contains all necessary key sets and security elements for full functionality (e.g., card issuer initiated card content changes, post-issuance personalization of applications belonging to the card issuer)*
  - *active security domains contain all necessary key sets and security elements for full functionality (e.g., application provider initiated card content changes, post-issuance personalization of applications belonging to the application providers)*



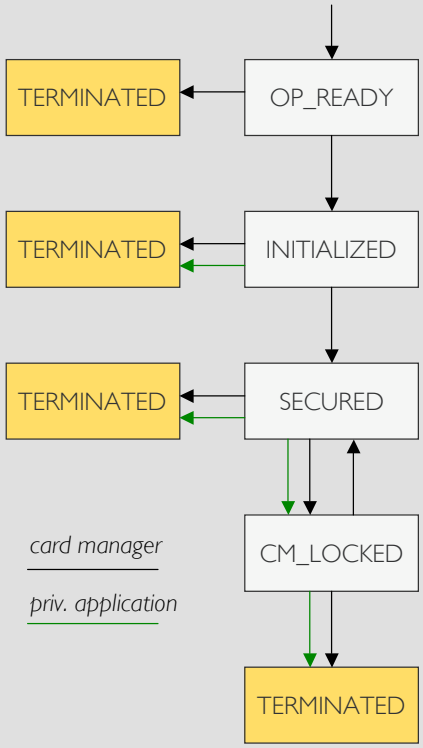
A. GP: Card Manager Life Cycle

- CM_LOCKED
  - *tell the card manager to temporarily disable all applications on the card except for the card manager (allows the card issuer to identify security threats)*
  - *the state transition to CM_LOCKED can be initiated by the card manager itself or a privileged application*
  - *the card issuer may reset the life cycle state to SECURED*



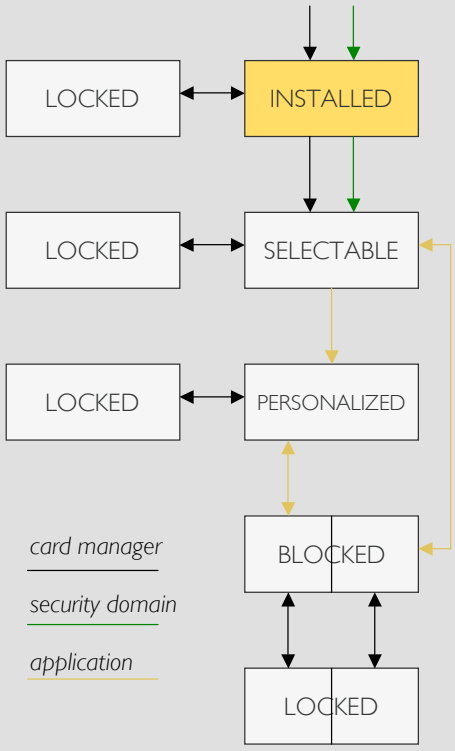
A. GP: Card Manager Life Cycle

- TERMINATED
  - *permanently disables all card functionality including the functionality of the card manager itself*
  - *mechanism for a privileged entity to logically 'destroy' the card for such reasons as the detection of a severe security threat or expiration of the card*
  - *the state transition to TERMINATED can be initiated by the card manager, the card issuer, or a privileged application*



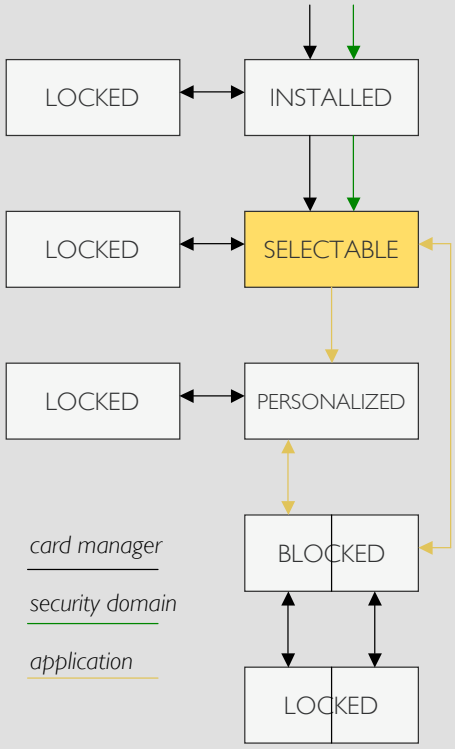
A. GP: Applet Life Cycle

- INSTALLED
  - *the application executable code has been properly linked and any necessary memory allocation has taken place so that the application can execute internally to the card*
  - *the application becomes an entry in the registry and is visible to authenticated off-card entities*



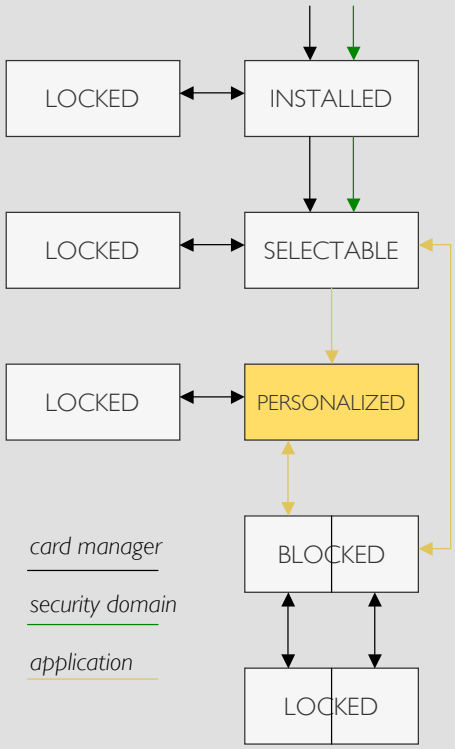
A. GP: Applet Life Cycle

- SELECTABLE
  - *the application is available to receive commands from outside the card*
  - *the transition to SELECTABLE can be combined with the application installation process so that INSTALLED is never visible to an off-card entity*
  - *while in this state the behaviour of the application is determined by the application itself*



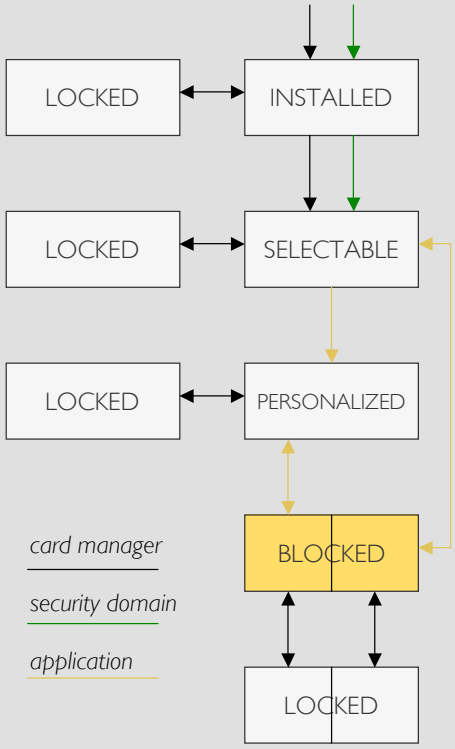
A. GP: Applet Life Cycle

- PERSONALIZED
  - *the application has all the necessary personalization data and keys for full runtime functionality*
  - *while in this state the behavior of the application is determined by the application itself*



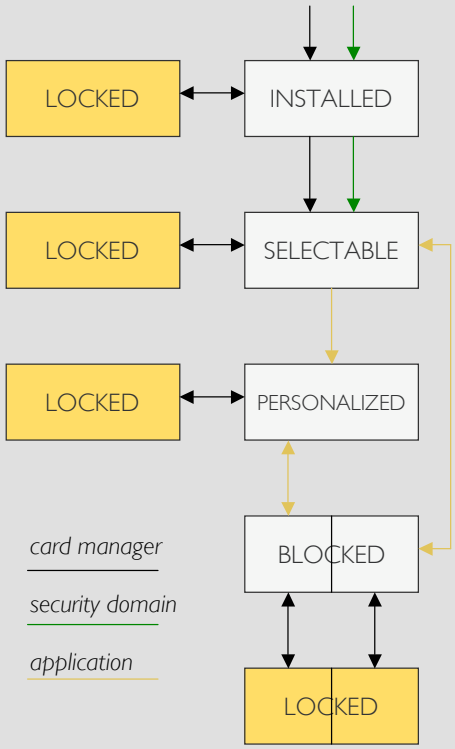
A. GP: Applet Life Cycle

- BLOCKED
  - *indicates that an application-specific security problem has been detected either from within the application or from outside the card*
  - *while in this state the behavior of the application is determined by the application itself*



A. GP: Applet Life Cycle

- LOCKED
  - *the card manager or card issuer uses the life cycle state LOCKED as a security management control to prevent the selection, and therefore the execution, of the application*
  - *the application can only become functional again if the card manager sets the application's life cycle back to the state that it held just prior to being set to the state LOCKED*



A. GP: `visa.openplatform`

- `visa.openplatform.OPSystem`
  - *exposed subset of the card manager*

```
public class OPSystem {
   public static byte getCardManagerState()
   public static boolean lockCardManager();
   public static boolean terminateCardManager();

   public static byte getCardContentState();
   public static boolean setCardContentState(byte state);

   ...
};
```


A. GP: Applet Life-Cycle Example

```
public void process(APDU apdu) {
   if (OPSystem.getCardContentState() == OPSystem.APPLET_BLOCKED)
     ISOException.throwit(SW_APPLET_BLOCKED);

   byte[] buf = apdu.getBuffer();

   if (OPSystem.getCardContentState() != OPSystem.APPLET_PERSONALIZED) {
      apdu.setIncomingAndReceive();

      switch (buf[ISO7816.OFFSET_INS]) {
       case VOP_INITIALIZE_UPDATE:
         ...
         return;
       case VOP_EXTERNAL_AUTHENTICATE:
         ...
         return;
       case PERSONALISE:
         ...
         OPSystem.setCardContentState(OPSystem.APPLET_PERSONALIZED);
         return;
      }

      ...
   }

   ... // normal processing
}
```

DEMO

A. GP: Roles & Responsibilities

- Card issuer responsibilities
  - *generating and loading the card manager keys (if not otherwise performed by the card manufacturer)*
  - *enforcing standards and policies for application providers governing all aspects of applications to be provided to the card issuer or operated on the card issuer's cards*
  - *working with application providers to load and initialize security domains*
  - *managing security relationships on the card*
  - *determining card manager policy with regards to card and application life cycle management and security parameters*
  - *managing application content both on a pre-issuance and post-issuance basis*
  - *cryptographically authorizing load files that are to be loaded through application provider security domains with delegated management privilege*

A. GP: Roles & Responsibilities

- Application provider responsibilities
  - *generating the keys for the application provider's security domain or obtaining security domain keys from a trusted third party*
  - *working with the card issuer to load generated keys into the security domain*
  - *providing applications that meet the card issuer's security standards/policies*
  - *providing load files according to defined application provider's security standards and policies*
  - *managing keys for security domains*
  - *obtaining pre-authorization from the card issuer to load and install applications onto the card issuer's card during post-issuance*
  - *returning load and install receipts to the card issuer following delegated management sessions*

A. GP: Roles & Responsibilities

- Run-time environment responsibilities
  - *providing a secure, consistent interface to all applications*
  - *performing secure memory management to ensure that each application's code and data is protected from unauthorized access from within the card*

Example: *JavaCard :-)*

A. GP: Roles & Responsibilities

- Card manager shall be able to
  - *initialize the card in a secure manner*
  - *communicate with off-card entities in accordance with the card issuer's security policies (a.k.a. secure messaging)*
  - *manage the secure loading, updating, and deletion of applications on-card in a post-issuance state*
  - *manage global card data including card and application life cycle states*
  - *perform optional internal velocity checks on the card global PIN to prevent card and application access violations*
  - *provide cryptographic protection for the card issuer's applications during their personalization phase*
  - *perform the verification of delegated management functions*

A. GP: Roles & Responsibilities

- Security domains shall be able to
  - *communicate with off-card entities in accordance with the card issuer's security policies (a.k.a. secure messaging)*
  - *provide cryptographic protection for the application provider's applications during their personalization (key loading) phase*
- With DAP or delegated management SDs are also responsible for
  - *performing the verification of the load files*
  - *performing the delegated management processes to securely load, update and delete applications post-issuance*
  - *returning load, install and delete Receipts during delegated management*

A. GP: Roles & Responsibilities

- Application responsibilities
  - *exposing only data and resources that are necessary for proper application function*
  - *performing internal velocity and other checking to determine if the application should block itself*