# Proposal for a specification of Demoney in Coq.

Jean Duprat

LIP, ENS-Lyon.

# Source.

- Demoney : A Demonstrative Electronic Purse — Card Specification —

  November 22, 2002

  Renaud Marlet, Cedric Mesnil,

  Trusted Logic.

# What is Demoney ?

- Basic electronic purse for smart cards.
  - simple but realistic :
    - fully addresses security issues regarding protocol attack;
    - the purse can be debited from a terminal in a store to pay a purchase,
    - credited at a bank terminal from cash or for a bank account.
  - specifications and a java Card implementation provided by Trusted Logic.

# Demoney variants.

- Demoney-stand-alone :
  - defined without any support from an underlying Open Platform system.
  - keys and Personal Identification Number explicitely handled by the application.
- Demoney-OP-2.0 and Demoney-OP-2.1 :
  - 2.0 : personalization keys and PIN handled by the OP version 2.0.
  - 2.1 : all security messages and PIN handled by the OP version 2.1.

# Demoney data.

Various kind of data stored in Demoney :

constant data,

persistent data,

session data.

# Data types.

- Numerical data :
  - nat (small), Z (values)
  - option nat or option Z if variant dependent data.
- Others :
  - byte (status)
  - list byte (information)
  - option (list byte)

# Structures.

- Variable :
  - name as equal as possible to the specification name.
- option :
  - to implement the variant cases.
- list :
  - to implement data of which length is the main verification argument.
- record :
  - to implement the logical group of data,
  - inductive structure with a name for each field.

# Groups of data.

- installation parameters :
  - provided at installation time;
- personalization parameters :
  - provided at installation time,
  - updated from an administration terminal;
- security parameters :
  - key sets;
- state variables :
  - given at personalization time;
- log records :
  - transaction trace.

Record **demoney** : Set := {

**ip** : installation_parameters;

**pp** : personalization_parameters;

**sp** : security_parameters;

**sv :** state_variables;

**lf** : list log_record

}.

# Installation parameters.

- number of records in the log file :
  - must be in the range of 1 - 50.
- PIN try limit :
  - only part of the Demoney Stand Alone variant,
  - in the range of 3 - 15.
- personalization key sets :
  - only part of the Demoney Stand Alone version.

```
Record
    installation_parameters :
    Set := {
    nb_of_records_in_log_file:
    nat;
    PIN_try_limit : option nat;
    personalization_key_set :
    option key_set
}.
```

# Personalization parameters

- purse identifier :
  - unspecified;
- purse currency :
  - unspecified,
  - updatable only if null currency;
- maximum transaction number :
  - rather large integer (<=65535);
- maximum balance :
  - positive integer;
- maximum debit amount :
  - positive integer;
- bank information :
  - optional, unspecified;
- AID of a purchase agent :
  - optional, unspecified.

Record
**personalization_parameters** : Set := {
**purse_identifier** : list byte;
**purse_currency** : list byte;
**maximum_transaction_nb** : Z;
**maximum_balance** : Z;
**maximum_debit_amount** : Z;
**bank_information** : option (list byte);
**AID_purchase_agent** : option (list byte)
}.

# Security parameters.

- debit key set :
  - used to secure debit transactions;
- credit key set :
  - used to secure credit transactions
- administration key set :
  - used to secure parameter updates as well as PIN unblocking
- PIN :
  - according to the version.

```
Record security_parameters : Set := {
    debit_key_set : key_set;
    credit_key_set : key_set;
    administration_key_set : key_set;
    PIN : list byte
}.
Record key_set : Set := {
    channel_encryption_key : list
    byte;
    message_authentication_key: list
    byte;
    data_encryption_key : list byte
}.
```

# State variables

- value given at personalization time.
- current transaction number :
  - initially 0;
- current balance :
  - initially 0;
- PIN presentation counter :
  - initially maximum number of tries;
- administrative status :
  - bit coded byte.

Record **state_variables** : Set := {
    **current_transaction_number**: Z;
    **current_balance** : Z;
    **PIN_presentation_counter** : Z;
    **administrative_status** : byte
}.

# Log file record.

- The log file is a cyclic file storing the newest records.
- purse transaction number :
  - integer;
- transaction currency :
  - unspecified;
- transaction amount :
  - signed integer;
- new balance :
  - positive integer;
- transaction context :
  - 4 unspecified informations.

Record **log_record** : Set := {
**purse_transaction_number**: Z;
**transaction_currency**: list byte;
**transaction_amount** : Z;
**new_balance** : Z;
**identifier_company** : list byte;
**POS_terminal_identifier** : list byte;
**POS_transaction_number** : list byte;
**date_time_stamp** : list byte
}.

# Format consistency.

static verification;

data sizes are always static.

numbers are interpreted.

# Key set format.

- key set :
  - 3 keys :
    - 16 byte format.

Inductive
**key_set_fmt_consistent** :
key_set -> Prop :=

| **ksfc** : forall l1 l2 l3 : list byte ,
length l1 = 16 ->
length l2 = 16 ->
length l3 = 16 ->
key_set_fmt_consistent
(Build_key_set l1 l2 l3).

# Numerical data format.

- using integer of Z,
- size format, signed or unsigned interpretation and interval constraints implemented by intervals.

```
Definition is_one_byte_unsigned_integer :=
    fun z:Z => (0 <= z <= 255)%Z.
Definition
    is_one_byte_positive_signed_integer
    :=
     fun z:Z => (0 <= z <= 127)%Z.
Definition is_two_byte_unsigned_integer :=
    fun z:Z => (0 <= z <= 65535)%Z.
Definition is_two_byte_signed_integer :=
    fun z:Z => (-32768 <= z <= 32767)%Z.
Definition
    is_two_byte_positive_signed_integer
    :=
    fun z:Z => (0 <= z <= 32767)%Z.
```

# Installation parameters format.

- Depends on the variant.
- number of records in the log file in the range of 1 - 50.
- PIN try limit in the range of 3 - 15.

```
Inductive ip_fmt_consistent : demoney_variant ->
      installation_parameters -> Prop :=
| ipfc_stand_alone : forall (n m:nat) (ks:key_set),
      1<=n<=50 ->
      3<=m<=15 ->
      key_set_fmt_consistent ks ->
      ip_fmt_consistent demoney_stand_alone
      (Build_installation_parameters n (Some m)
      (Some ks))
| ipfc_OP_2_0 : forall n:nat,
      1<=n<=50 ->
      ip_fmt_consistent demoney_OP_2_0
      (Build_installation_parameters n None None)
| ipfc_OP_2_1 : forall n:nat,
      1<=n<=50 ->
      ip_fmt_consistent demoney_OP_2_1
      (Build_installation_parameters n None None).
```

# Personalization parameter format.

- Depends on the optional fields.
- purse identifier :
  - 4 byte length;
- currency :
  - 2 byte length;
- maximum transaction number :
  - 2 byte unsigned integer;
- maximum balance :
  - 2 byte positive signed integer;
- maximum debit amount :
  - 2 byte positive signed integer;
- bank information :
  - optional, 16 byte length;
- AID of purchase agent :
  - optional, 5 - 16 byte length.

```
Inductive pp_fmt_consistent :
      personalization_parameters -> Prop :=
| ppfc00 : forall (pid cur : list byte) (maxt maxb maxd :
      Z),
      length pid = 4  ->
      length cur = 2  ->
      is_two_byte_unsigned_integer maxt ->
      is_two_byte_positive_signed_integer maxb ->
      is_two_byte_positive_signed_integer maxd ->
      pp_fmt_consistent
      (Build_personalization_parameters pid cur maxt
      maxb maxd  None None)
| ppfc10 : …
| ppfc01 : …
| ppfc11 : forall (pid cur bki aid : list byte) (maxt maxb
      maxd : Z),
      … ->
      length bki = 16  ->
      5 <= length aid <= 16  ->
      pp_fmt_consistent
      (Build_personalization_parameters pid cur maxt
      maxb maxd  (Some bki) (Some aid)).
```

# Security parameter format.

- Keys and PIN are private parameters.
- They are provided as ciphertext and cannot be retrieved.
- For the format, the length is only taken in account.

```
Inductive sp_fmt_consistent :
    security_parameters -> Prop :=
| spfc : forall (dek crk adk :
    key_set) (pin : list byte),
    key_set_fmt_consistent dek  ->
    key_set_fmt_consistent crk  ->
    key_set_fmt_consistent adk  ->
    length pin = 16  ->
    sp_fmt_consistent
    (Build_security_parameters dek
    crk adk pin).
```

# State variable format.

- transaction number :
  - 2 byte unsigned integer;
- Balance :
  - 2 byte positive signed integer;
- PIN presentation counter :
  - 1 byte positive signed integer;
- Administrative status :
  - 1 bit coded byte,
  - contains Demoney variant information.

```
Inductive sv_fmt_consistent : demoney_variant ->
      state_variables -> Prop :=
| svfc_sa_0 : forall (tn ba pip : Z) (hx : hexa),
      is_two_byte_unsigned_integer tn ->
      is_two_byte_positive_signed_integer ba ->
      is_one_byte_positive_signed_integer pip ->
      sv_fmt_consistent demoney_stand_alone
      (Build_state_variables tn ba pip (hx,h0))
| svfc_sa_8 : …
| svfc_op20_1 : …
| svfc_op20_9 : …
| svfc_op21_3 : …
| svfc_op21_11 : forall (tn ba pip : Z) (hx : hexa),
      is_two_byte_unsigned_integer tn ->
      is_two_byte_positive_signed_integer ba ->
      is_one_byte_positive_signed_integer pip ->
      sv_fmt_consistent demoney_OP_2_1
      (Build_state_variables tn ba pip (hx,hB)).
```

# Log record format.

- Purse transaction number :
  - 2 byte unsigned integer;
- Transaction currency :
  - 2 byte unspecified;
- Transaction amount :
  - 2 byte signed integer;
- New balance :
  - 2 byte positive signed integer;
- Transaction context :
  - 4 * 4 byte unspecified information.

```
Inductive lr_fmt_consistent :
    log_record -> Prop :=
| lrfc : forall (pt ta nb : Z) (tc idc pti ptn
    dts : list byte),
    is_two_byte_unsigned_integer pt ->
    length tc = 2  ->
    is_two_byte_signed_integer ta ->

    is_two_byte_positive_signed_intege
    r nb ->
    length idc = 4  ->
    length pti = 4  ->
    length ptn = 4  ->
    length dts = 4  ->
    lr_fmt_consistent (Build_log_record
    pt tc ta nb idc pti ptn dts).
```

# log file format.

- List of log records.
- The length of the list is statically verified in the Demoney format and dynamically verified in Demoney data consistency.

```
Inductive llr_fmt_consistent :
    (list log_record) -> Prop :=
| nilfc : llr_fmt_consistent nil
| consfc : forall (lr : log_record)
    (llr : list log_record),
    lr_fmt_consistent lr ->
    llr_fmt_consistent llr ->
    llr_fmt_consistent (lr :: llr).
```

# Demoney format.

- installation parameters depend on the variant;
- every fields have a consistent format;
- the length of the log file is between 1 to 50.

```
Inductive demoney_fmt_consistent :
    demoney_variant -> demoney -> Prop :=
| dfc_sa : forall (insp : installation_parameters)
    (persp : personalization_parameters)
    (secp : security_parameters)
    (stav : state_variables) (llr : list log_record),
    ip_fmt_consistent demoney_stand_alone insp ->
    pp_fmt_consistent persp ->
    sp_fmt_consistent secp ->
    sv_fmt_consistent demoney_stand_alone stav -
    >
    1 <= length llr <= 50  ->
    llr_fmt_consistent llr ->
    demoney_fmt_consistent demoney_stand_alone
    (Build_demoney insp persp secp stav llr)
| dfc_op20 : …
| dfc_op21 : …
```

# Data consistency.

## Dynamical verification.

# Length of the Log file.

- The number of records in the log file (installation parameter) is equal to the length of the list of log records.
- Note : at the beginning of the use, several places for records contain null records.
- The model can be changed, it depends on the implementation of the log file as a list.

Definition
**consistent_number_records**
:=
fun (insp : installation_parameters)
(llr : list log_record) =>
nb_of_records_in_log_file insp =
length llr.

# Number of transactions.

- The current transaction number (state variable) has to be lesser than the maximum transaction number (personalization parameter).

```
Definition
    consistent_transaction_number
    :=
fun (persp :personalization_parameters)
    (stav : state_variables) =>
(current_transaction_number stav <=
    maximum_transaction_nb
    persp)%Z.
```

# Balance.

- The balance (state variable) :
  - is positive
  - is lesser than the maximum (personalization parameter).

```
Definition consistent_balance
    :=
fun (persp :
    personalization_parameters)
    (stav : state_variables) =>
    (0 <=
    current_balance stav <=
    maximum_balance
    persp)%Z .
```

# Debit.

- The maximum debit amount is positive.

Definition **consistent_debit_amount** :=

fun persp : personalization_parameters =>

(0 <= maximum_debit_amount persp)%Z .

# Data consistency.

- Logical and of the previous properties.

```
Inductive demoney_data_consistent :
    demoney -> Prop :=
Iddc : forall
    (insp : installation_parameters)
    (persp : personalization_parameters)
    (secp : security_parameters)
    (stav : state_variables)
    (llr : list log_record),
    consistent_number_records insp llr ->
    consistent_transaction_number persp
    stav ->
    consistent_balance persp stav ->
    consistent_debit_amount persp ->
    demoney_data_consistent
    (Build_demoney insp persp secp stav llr).
```

# Future works.

# Access levels.

- Public :
  - the initial access level
  - each time the application is newly selected.
- Debit :
  - allowing a terminal to unload the purse.
- Credit :
  - allowing the purse to be loaded from cash or a bank account.
- Credit-ident :
  - PIN identification + credit operations.
- Admin :
  - updating personalization parameters.

# Commands.

- Store-data.
- Select.
- Initialize-update.
- External authenticate.
- Put key.
- PIN change / unblock.
- Verify PIN.
- Initialize transaction.
- Complete transaction.
- Get data.
- Put data.
- Read record.

# Specification about commands.

- authorizations (level access).
- data modifications.
- data consistency maintenance.

# Secure channels ?

- specifications ?