# CIDFINMA Z Schema Description

developed by Serge (Siarhei Vinahradau, vinahradau@yahoo.de)

## Contents

## *References*

CIDFINMA_spec_Z

FINMA-Rundschreiben „Operationelle Risiken – Banken“ (published in 20.11.2008, last updated in 2017). https://www.finma.ch/de/~/media/finma/dokumente/rundschreiben-archiv/finma-rs-200821---30-06-2017.pdf

## *Data Classification*

DATACATEGORY
CIDCATEGORIES

According to FINMA 10*, categories of client identifying data (CID) include:

- direct (e.g. customer name): DIRECT;
- indirect (e.g. passport number): INDIRECT;
- potentially indirect (combination of data): POTENTIALLYINDIRECT.

In our specification we also use:

- NONCID (for data not classified as CID).
- PROTECTED (for protected, e.g. anonymized CID data, in which case it's not CID anymore).

DIRECT, INDIRECT and POTENTIALLYINDIRECT is included into the CIDCATEGORIES collection accordingly.

```
DATACATEGORY ::= DIRECT | INDIRECT | POTENTIALLYDIRECT | PROTECTED | NONCID
CIDCATEGORIES == {DIRECT, INDIRECT, POTENTIALLYDIRECT}
```

## *Implementation of Data Classification*

METADATA
CONTENT
ENTITY
DOMAIN

InitDomain
AssignDataOwner
ClassifyDataCategory
ImplementDataClassification
RecycleData

In our schema, METADATA is an abstraction for data descriptors or attributes, such as CUSTOMERNAME, CUSTOMERADDRESS or ISVIPCUSTOMER (in reality, it can be a technical data descriptor, e.g. a database schema, table and column or an XML attribute path).

We describe the link between METADATA and DATACATEGORY, understood as CID data classification, via a partial function in DOMAIN schema. Here the function imposes its own constraint: a domain element of dataClassification, which is of type METADATA, can be linked to at most one data category.

dataClassification: METADATA ⇸ DATACATEGORY

Data owners (ENTITY in our schema) are responsible for the whole data lifecycle, including the CID data classification (FINMA 14*).

dataOwner: METADATA ⇸ ENTITY

The following constraint in DOMAIN schema states that domain of dataClassification function is a subset of dataOwner, i.e. the data classification happens after the data ownership assignment:

dom dataClassification ⊆ dom dataOwner

Operations AssignDataOwner and ClassifyDataCategory populate the functions, and ImplementDataClassification is the sum of the two operations above. RecycleData does the opposite, removing the links.

Data setup example:

*dataOwner == {(CUSTOMERNAME, ENTITY1), (ISVIPCUSTOMER, ENTITY1)}*
*dataClassification == {(CUSTOMERNAME, DIRECT),(ISVIPCUSTOMER, NONCID)}*

## *Data Storage*

COUNTRY
CONTENT
NODEID
NODE
CIDSTORINGNODESAUDITLOG

AddNodeData

NODE is an abstraction for systems and applications (FINMA 15*).

Our NODE schema has its own data category classification function:

nodeDataCategories: METADATA ⇸ DATACATEGORY

If CID data is stored outside Switzerland, it has to be protected, e.g. anonymized (FINMA 20*). The NODE schema has a corresponding constraint:

nodeCountry = SWITZERLAND ∨ (∀ c : ran nodeDataCategories ⦁ c ∉ CIDCATEGORIES)

As already mentioned, in cases of CID data protection (e.g. anonymization), data loses its CID status (in our schema, PROTECTED is not a member of CIDCATEGORIES). As a consequence, nodeDataCategories may differ from the global dataClassification: the same data may be classified by the data owner as CID, but stored as protected (non-CID) on a node.

Data contents (or data values) are described as nodeDataContents function. Our schema expects that for all stored data contents, their data categories are known (NODE schema):

nodeDataContents: METADATA ⇸ CONTENT
dom nodeDataContents ⊆ dom nodeDataCategories

The bank has to know where CID data is stored (FINMA 15*). In our schema, cidStoringNodesIds is a power set of node identifiers, and the following constraint expects identifiers of all nodes with stored CID data to be recorded in cidStoringNodesIds:

cidStoringNodesIds: ℙ NODEID
∀ cidDataCategory : ran nodeDataCategories ⦁ cidDataCategory ∈ CIDCATEGORIES ⇒
nodeId ∈ cidStoringNodesIds

Data storage logic is described in AddNodeData operation:

- Case 1: non-CID; store data;
- Case 2: CID and node in Switzerland; store data; create a node identifier entry in cidStoringNodesIds;
- Case 3: CID and node not in Switzerland ; store protected (anonymized) version of the data (XXXXX is stored instead of the data content coming from nodeDataContentInput?).

Case 3:

nodeDataContents′ = nodeDataContents ⊕ {nodeMetadataInput? ↦ XXXXX}
∧
nodeDataCategories′ = nodeDataCategories ⊕ {nodeMetadataInput? ↦ PROTECTED})

Stored data example, node in Switzerland:

*nodeDataCategories=={(CUSTOMERNAME, DIRECT), (ISVIPCUSTOMER, NONCID)}*
*nodeDataContents == {(CUSTOMERNAME, MUSTERMANN),*
*(ISVIPCUSTOMER, YES)}*

Stored data example, node outside Switzerland:

*nodeDataCategories=={(CUSTOMERNAME, PROTECTED),*
*(ISVIPCUSTOMER, NONCID)}*
*nodeDataContents=={(CUSTOMERNAME, XXXXX), (ISVIPCUSTOMER, YES)}*

## *Roles, Users and Access Rights*

ROLE
USER

AddRole
AddUserAccessRights

To regulate access to CID data, Finma expects banks to have a role or function based authorisation system (FINMA 22*). To describe the authorization system, our schema introduces the abstractions of ROLE and USER. Relations are defined as follows:

roles: ROLE ↔ METADATA
userAccessRigths: USER ↔ ROLE

For the data structures above, we choose relations, not functions, since a role can be linked to multiple METADATA instances, and USER can have multiple roles.

Thus, USER is not assigned access to METADATA instances directly, but rather through ROLE. This approach is an example of RBAC Access Control Model (Role Based Centralized) and provides additional flexibility in access rights maintenance.

To populate roles and userAccessRigths relations, AddRole and AddUserAccessRights operations use mathematical unions of the existing relations and new pairs created from the inputs:

roles′ = roles ∪ {(role?, metadata?)}
userAccessRigths′ = userAccessRigths ∪ {(user?, role?)}

Authorization data example:
*roles=={(ROLEGUIUSER, CUSTOMERNAME), (ROLEGUIUSER, ISVIPCUSTOMER)}*
*userAccessRigths=={(USER1, ROLEGUIUSER)}*

## *Data Access*

`AccessNode`

Assuming the necessary access rights are available, a user can access particular data stored on a node. This logic is described in `AccessNode` operation.

- Case 1: user tries to access CID data from outside Switzerland: a protected (anonymized) version of the data is transmitted (instead of the real data content, `XXXXX` is stored in the `contentOutput!` Variable);
- Case 2: otherwise (non-CID or user in Switzerland); transmit the unmodified data to the user (unchanged data content is stored in `contentOutput!` Variable).

Thus, requirement in FINMA 20* is fulfilled.

Data access output example, CID data for a user in Switzerland:
*contentOutput!=={MUSTERMANN}*

Data access output example, CID data for a user outside Switzerland:
*contentOutput!=={XXXXX}*

## *Bulk Data Access*

CIDBULKLOG

AccessBulk

Bulk access is a process of accessing big quantities of data (FINMA 40*). For CID bulk access, the FINMA regulation expects a special procedure, e.g. instances of bulk access being documented in log files. As an abstraction for bulk access logs, our schema defines a relation between users and nodes. Here, a user may be a part of multiple log entries (pairs), hence a relation, not a function.

$$cidBulkAccess: USER \leftrightarrow NODEID$$

The bulk access logic is described in AccessBulk operation. Only 2 cases are defined here:

- Case 1: user in Switzerland accessing a node with stored CID data: all node data is stored in contentOutput!, and a cidBulkAccess log entry is created;
- Case 2: otherwise, all node data is stored in contentOutput!, and no cidBulkAccess log entry is created.

Mathematically, the absence of CID data on a node is described in AccessBulk operation in the following predicate: no data categories stored on a node belong to the set of CID data categories (their intersection is empty).

$$ran\ nodeDataCategories \cap CIDCATEGORIES = \varnothing$$

For the bulk access to be enabled, our schema does not expect AddRole to be called, since AccessBulk should provide access to all data on a node, regardless of the metadata descriptors. Thus, the authorization logic in AccessBulk does not check the metadata descriptor of the data to be transmitted. Here in AccessBulk operation, checking ROLEBULKCID and ROLEBULK roles of the user is sufficient, as an opposite to AccessNode operation, where the metadata descriptors of the stored data are explicitly checked.

Bulk data access example, CID data and non-CID data, user inside Switzerland:
*contentOutput!=={MUSTERMANN, YES}*

Bulk data access example, protected data and non-CID data, user inside or outside Switzerland:
*contentOutput!=={XXXXX, YES}*