A Model for Provably Secure Software Design

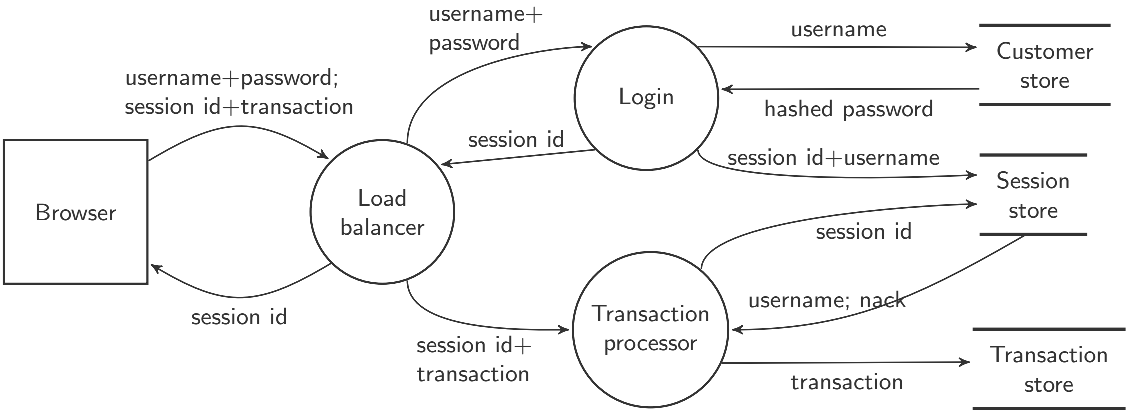
Alexander van den Berghe[1], Koen Yskout[1]
Riccardo Scandariato[2], Wouter Joosen[1]
[1]imec-DistriNet, KU Leuven, Belgium
[2]Software Engineering Division, Chalmers and Göteborg University, Sweden

DistriNet

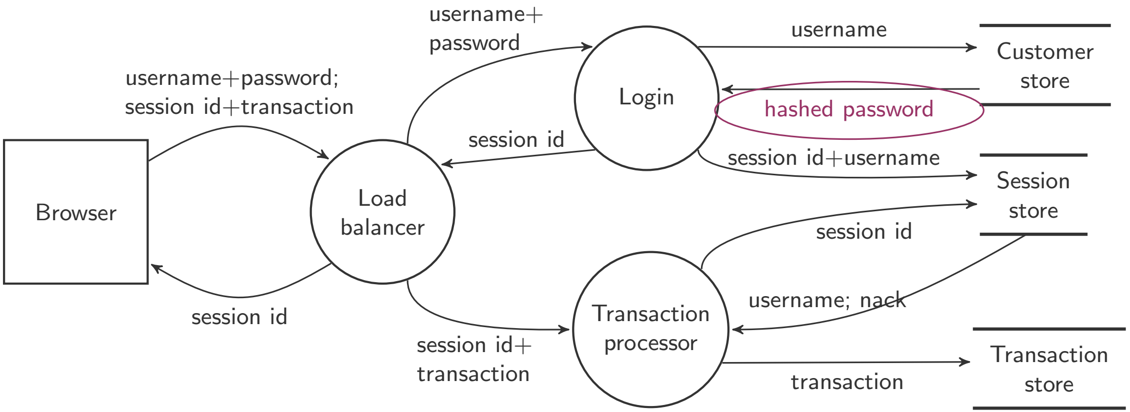
FormaliSE 2017
27 May 2017

Setting the scene

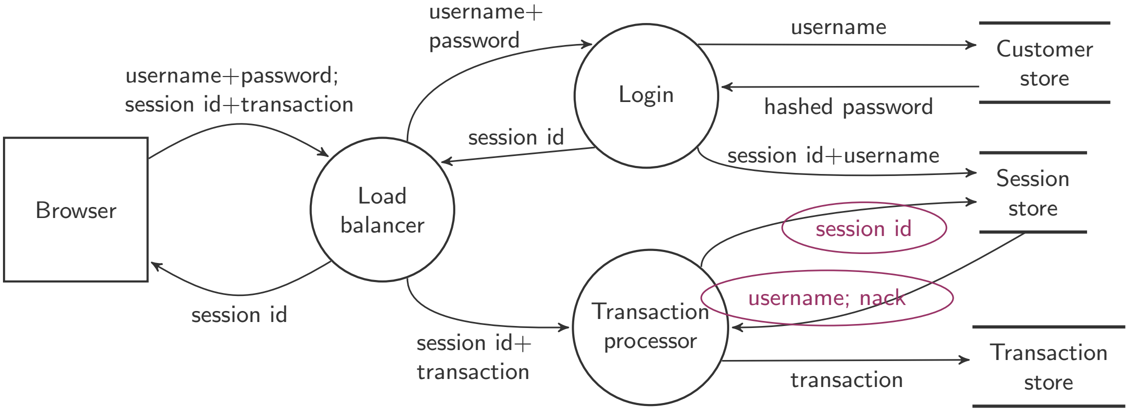
Possible DFD for a banking system



Possible DFD for a banking system

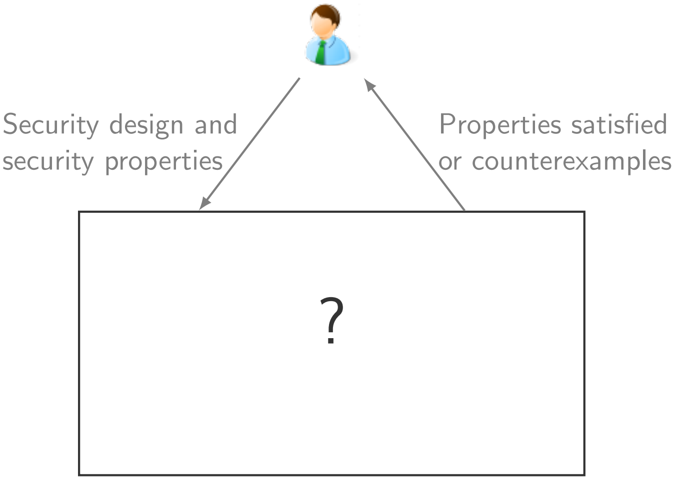


Possible DFD for a banking system

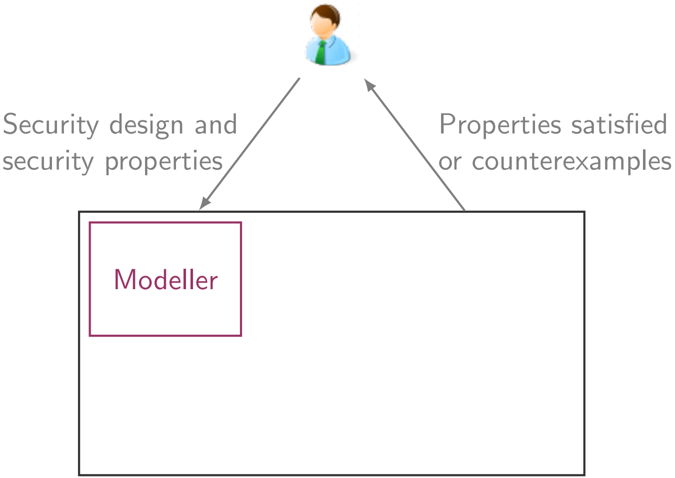


Our vision to improve on this

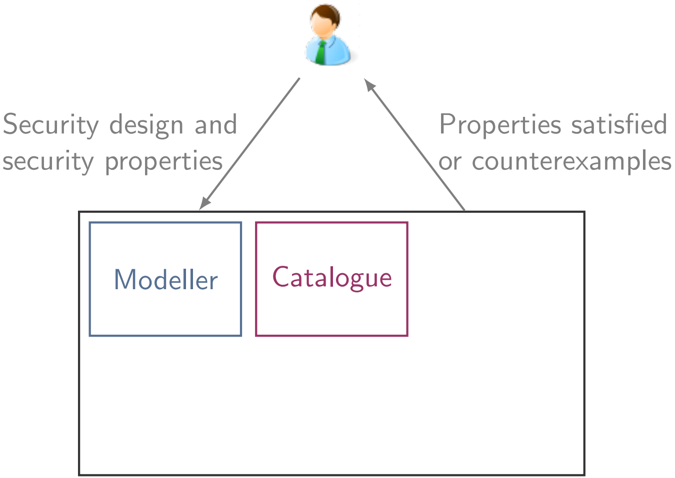
Our end goal


Security design and
security properties
Properties satisfied
or counterexamples
?

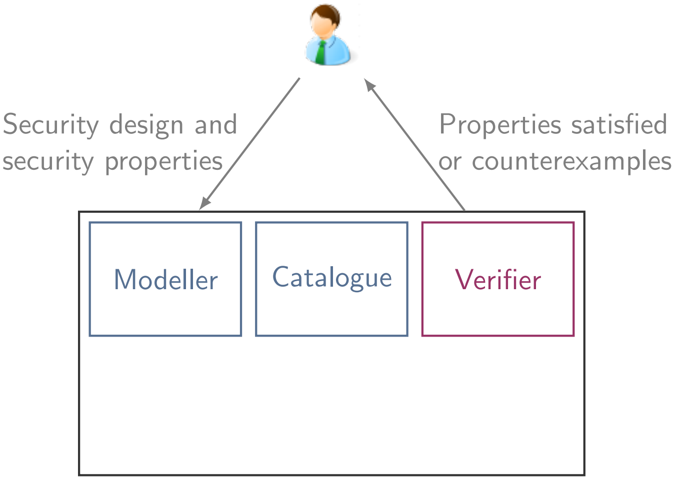
Model on familiar abstraction level



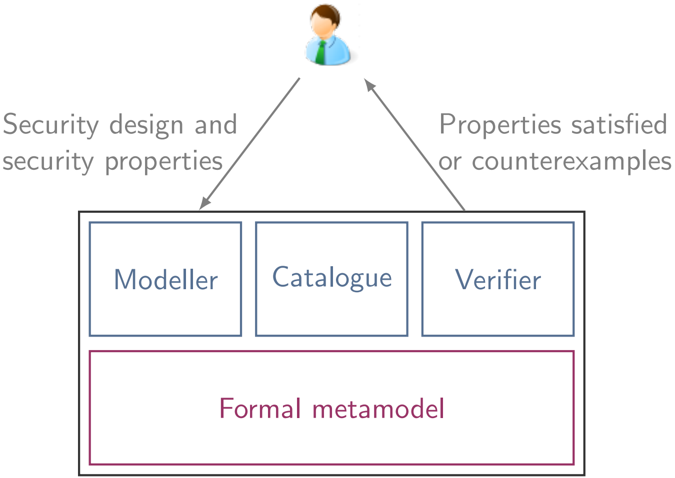
Reuse well-known security solutions



Automate property verififaction



All of this based on a formal foundation



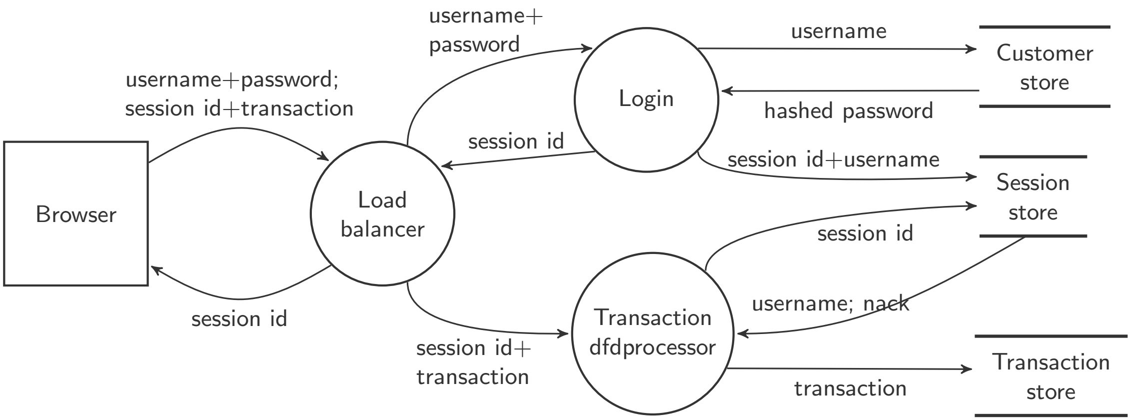
A precise model for security design

Bird's eye view of our model

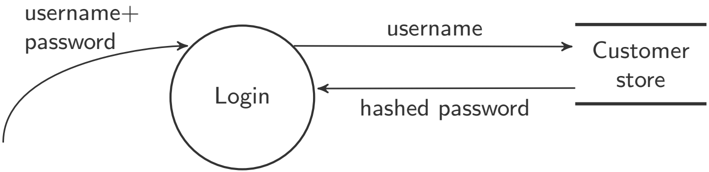
Data operated on by **processes** that can be connected to each other to form **networks**

Formalised using the Coq Proof Assistant

Recall the banking system DFD



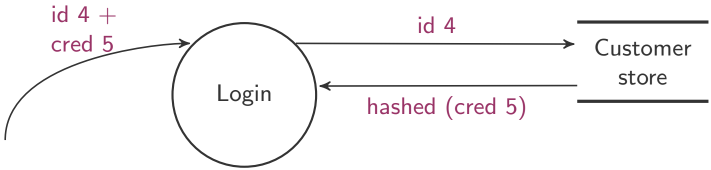
Let us focus on the login process



Behaviour to model:

Compare hash value of received password to the one stored

The data types in our model



Security specific data types

cryptographic key, **identity**, **credential**, **session identifier**, **signature**

Transformed data

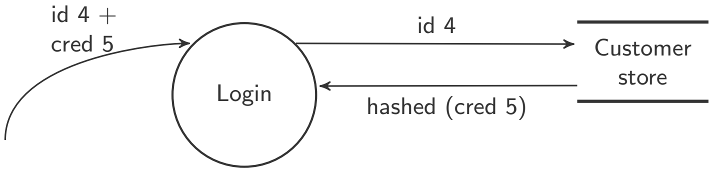
hashed, **encrypted**

Abstract non-security data type

plain

Collections to construct complex data structures

Pre-defined, off the shelf processes as building blocks

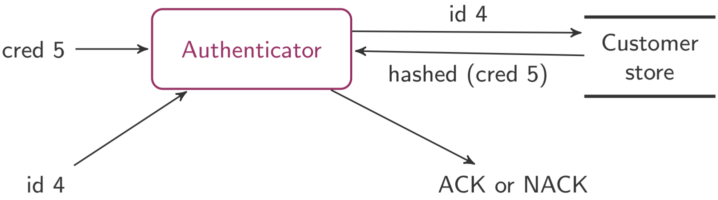


Each encapsulating well-defined, possibly non-deterministic behaviour by

a state machine; and

sets of input and output queues

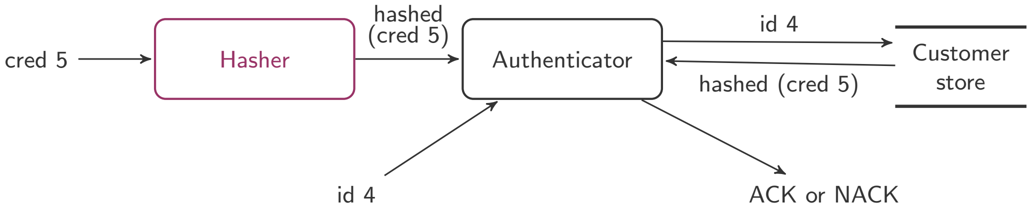
Introducing the Authenticator process



Behaviour:

Verifies whether some provided identity and credential match with a looked-up version

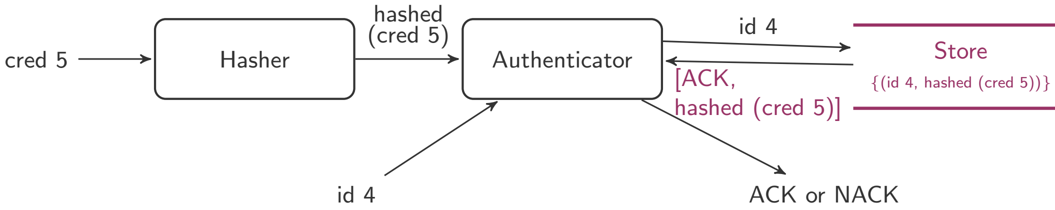
Explicitly calculating the hash value



Behaviour:

Calculates a hash value of its input data

Replace the Customer Store by our Store process



Behaviour:

Stores data as key-value pair

Security design as a "network" of processes

Network $\triangleq$ a set of processes connected by channels

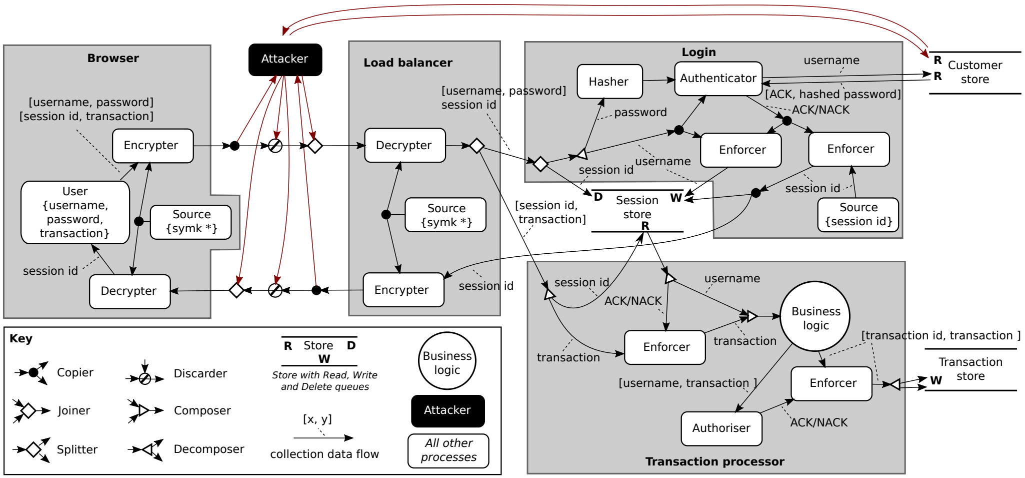
Transition relation between 2 networks:

1. local state transition for each process; and
2. propagate (some) process outputs along connected channel

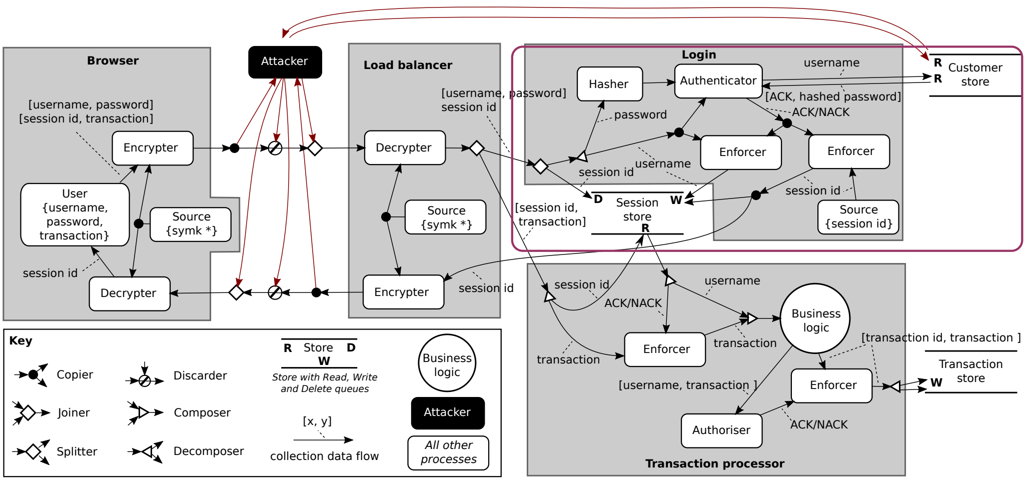
$\Rightarrow$ Can construct an infinite sequence of successive networks

Apply to the whole banking DFD

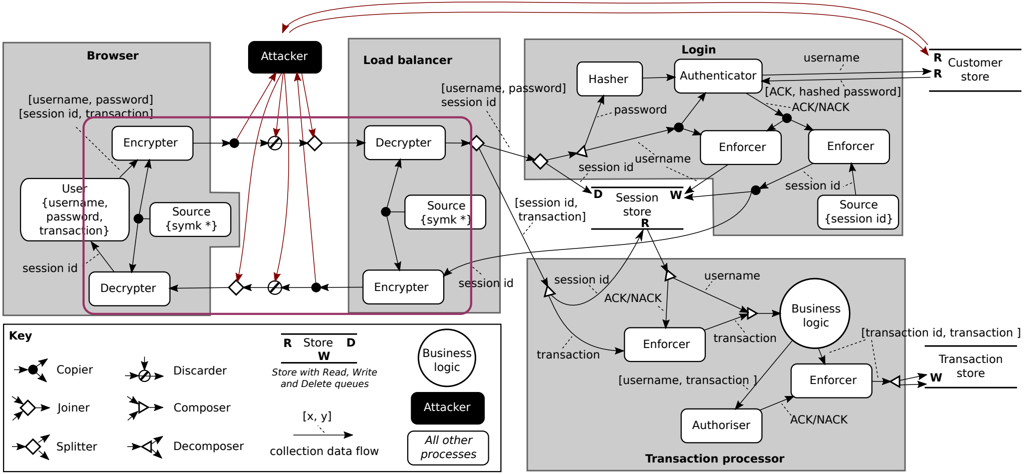
The banking system using our model



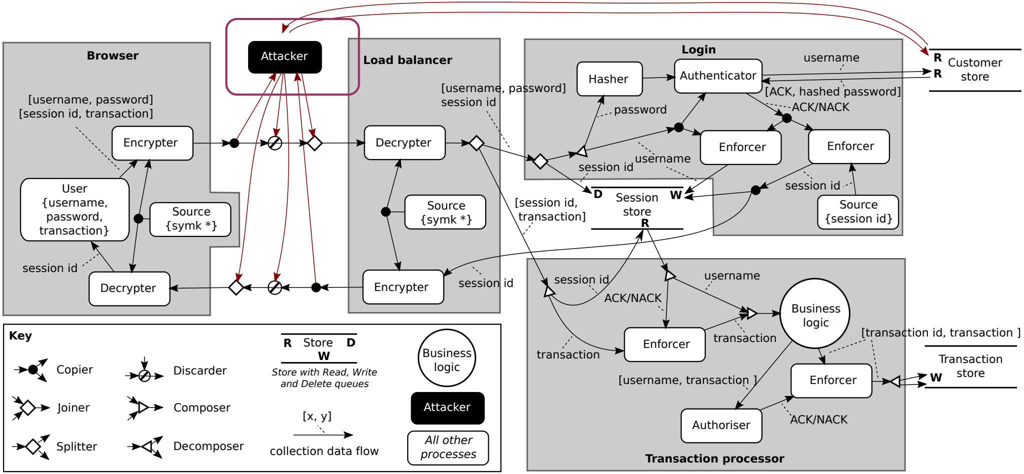
Username/password authentication with sessions



Simplified HTTPS



Incorporated attacker model



Reasoning about security

Proving data origin authentication for transactions

Formalised using Linear-time Temporal Logic (LTL)

$$\Box(in\_input\ tx\ bl \implies (\blacklozenge\ in\_output\ tx\ user))$$

Hypothesis

transaction tx arrived as input for the business logic

Goal

transaction tx must have been sent by user earlier

Intuition of proof

start at business logic and "step backwards" process by process

Some resulting assmptions

that became explicit while proving

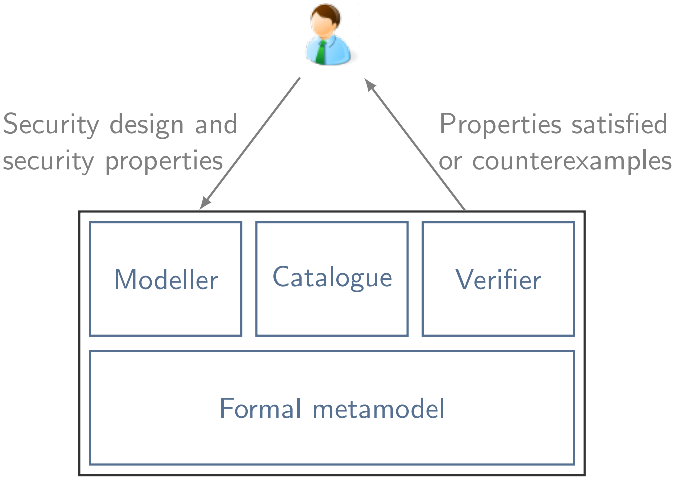
Attacker cannot guess user's password (i.e. brute force)
Reasonable if good password policy is enforced.

Attacker cannot decrypt data without correct key
Reasonable if strong encryption is used.

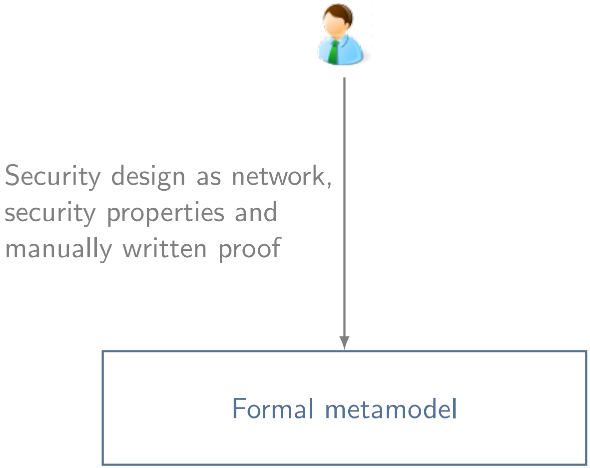
$\Rightarrow$ Should be verified against whole design (incl. documentation)

Conclusion

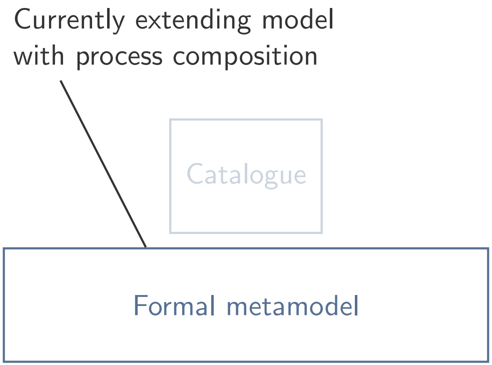
Recall our vision



Current state of affairs



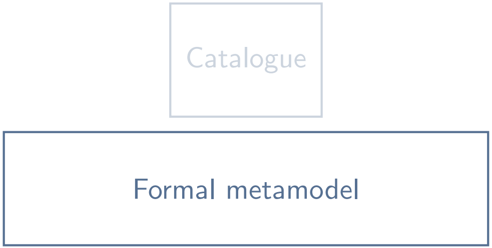
Initial steps towards catalogue



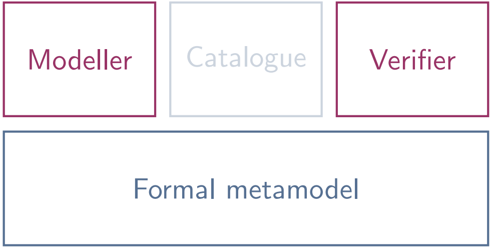
Assessing the model as foundation

Performed a user study with $\pm$100 master students to assess understandability of model (elements)

Sneak peek: students indicated they found model (elements) easy to understand



Further down the road



A Model for Provably Secure Software Design

Alexander van den Berghe[1], Koen Yskout[1]
Riccardo Scandariato[2], Wouter Joosen[1]
[1]imec-DistriNet, KU Leuven, Belgium
[2]Software Engineering Division, Chalmers and Göteborg University, Sweden

DistriNet

FormaliSE 2017
27 May 2017

Code samples

Data

```
Inductive Data : Type :=
| plain: nat → nat → Data
| key: CryptoKey → Data
| id: Identity → Data
| cred: Credential → Data
| sid: SessionId → Data
| sig: Data → CryptoKey → Data
| enc: Data → CryptoKey → Data
| hashed: Data → Data
| collection: nat → list Data → Data
```

Hasher process

```
Inductive HState :=
| h_idle: HState
| h_hashing: Data → HState.

Record State := mk_hstate {
  hstate: HState; iostate: IOState }.

Inductive HTrans : State → State → Prop :=
| h_read: ∀ (d : Data) (io io' : IOState),
    (Some d, io') = read_input io IN_DATA →
    HTrans (mk_hstate h_idle io)
           (mk_hstate (h_hashing d) io')
| h_write: ∀ (d : Data) (io io' : IOState),
    io' = write_output io OUT_DATA (hashed d) →
    HTrans (mk_hstate (h_hashing d) io)
           (mk_hstate h_idle io').
```

Network

```
Record Channel_End := mk_end {
  processID: ProcessID;
  queue_name: QueueName
}.

Record Channel := mk_chan {
  source: Channel_End;
  target: Channel_End
}.

Record Network := nw {
  processes: list Process;
  channels: list Channel
}.
```

Network transition relation

```
Inductive N_step : Network → Network → Prop :=
| n_step: ∀ ps ps' cs cs_prop,
  step_all ps ps' →
  incl cs_prop cs →
  N_step (nw ps cs)
         (nw (propagate_all cs_prop ps') cs).
```

Confidential

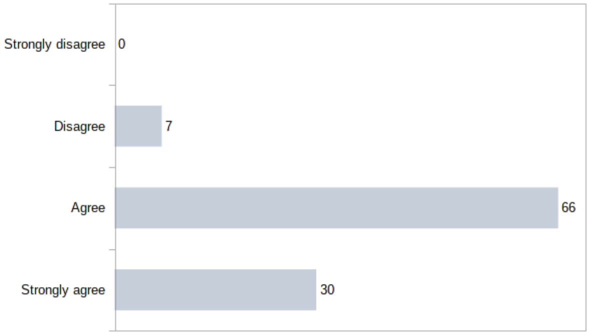
```
Definition confidential (d : Data) (n : NetworkWF) :=
  ∀ s, path s n → s@0 ⊨ [] (no_attacker_knows d).
```

Data origin authentication

```
Definition data_origin_authentication (f : Data → Prop)
  (rcv snd : ProcessID) (qr qs : QueueName) (n : NetworkWF) :=
  ∀ s d, path s n → f d →
         s@0 ⊨ [] (implies (contained_in_input d qr rcv)
                           (◆(contained_in_output d qs snd))).
```

Some initial data from user study

The semantics of the model kind elements (processes, channels, networks) are straightforward to understand.



Overview of available processes

Security processes

Process	Description
Hasher	Calculates a hash value of its input data.
Encrypter	Encrypts input data with a provided cryptographic key.
Decrypter	Decrypts input data with a provided cryptographic key.
Authenticator	Verifies whether an identity and credential match with a looked-up version.
Enforcer	Enforces input data to be cleared before passing on.
Authoriser	Encapsulates an authorisation policy by non-deterministically allowing or denying requests
Generator	Generates a digital signature given a data element and a cryptographic key.
Verifier	Verifies whether a data element and signature match.

External processes

Process	Description
User	Non-malicious user interacting with the system.
Attacker	Malicious user interacting with the system.
Source	Produces data satisfying a pre-defined property.
Sink	Consumes its input.

Auxiliary processes

Process	Description
Business	Encapsulates the non-security related functionality of the system under design.
Store	Stores data as key-value pairs.
Comparator	Compares two data elements using a decidable function.
Collector	Collects the first data element of its n first input queues into a collection.
Disperser	Disperses a collection into its contained elements.
Dropper	Non-deterministically chooses to forward or discard its input data.
Discarder	Discards its input data if directed to by another process.
Joiner	Outputs data from a non-deterministically selected input queue.
Copier	Copies its input data to each of its output queues.
Fork	Outputs input data to a non-deterministically selected output queue.
Latch	Remembers its last received input data and continues to output it.