# Using model checkers to elicit security metrics

Thomas Heyman, Koen Yskout, Christophe Huygens and Wouter Joosen

DistriNet, K.U.Leuven, Belgium

# Recap

# MetriCon1.0

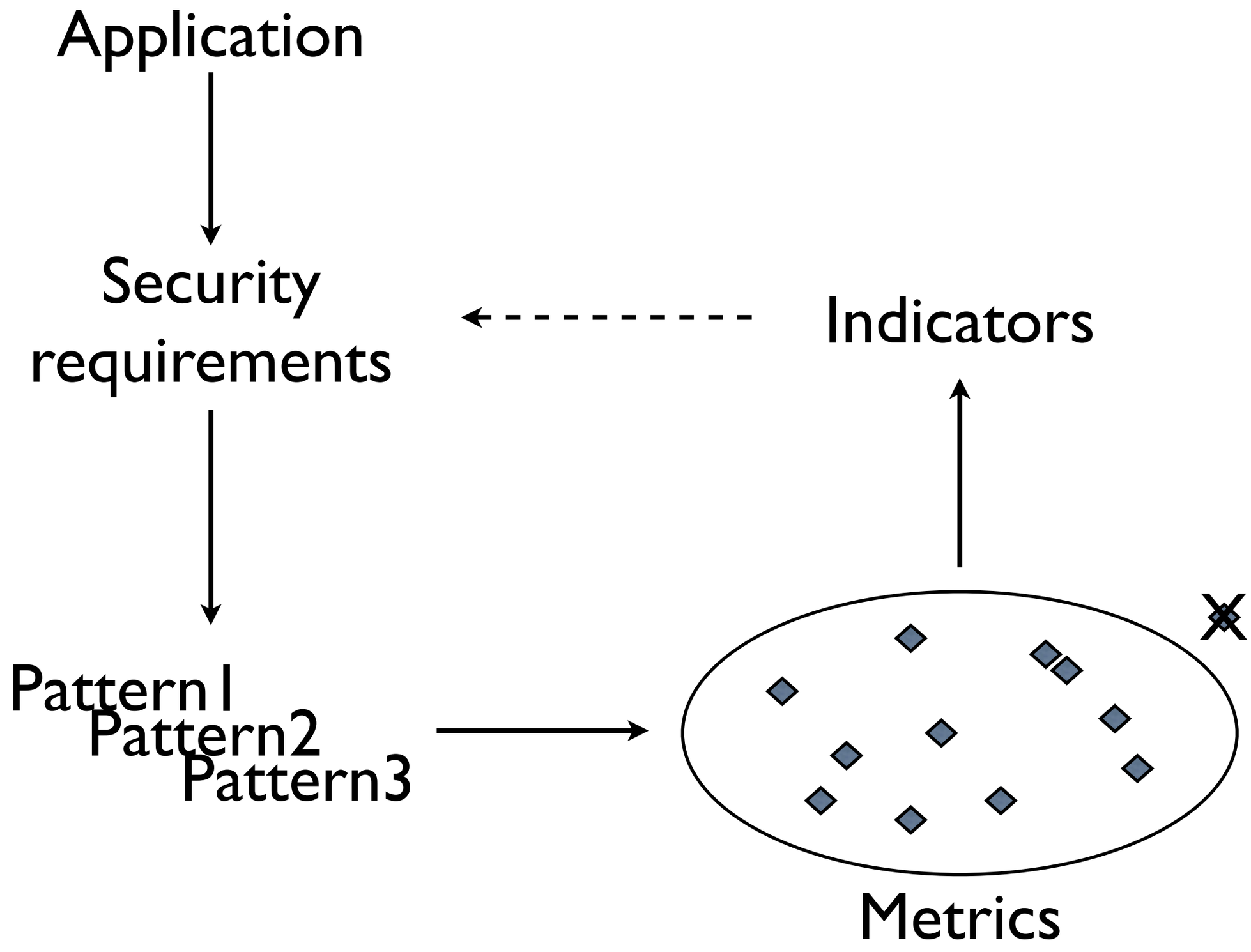
# Assigning reusable code-level metrics to security patterns

# MetriCon2.0

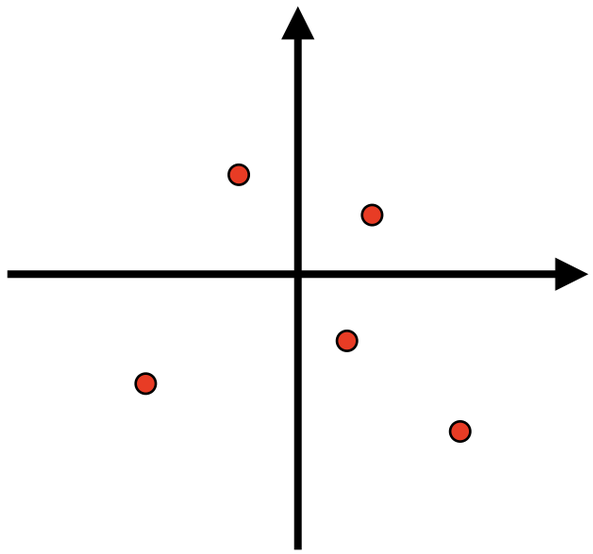
## Combining low-level measurements to high-level indicators

# MetriCon3.0

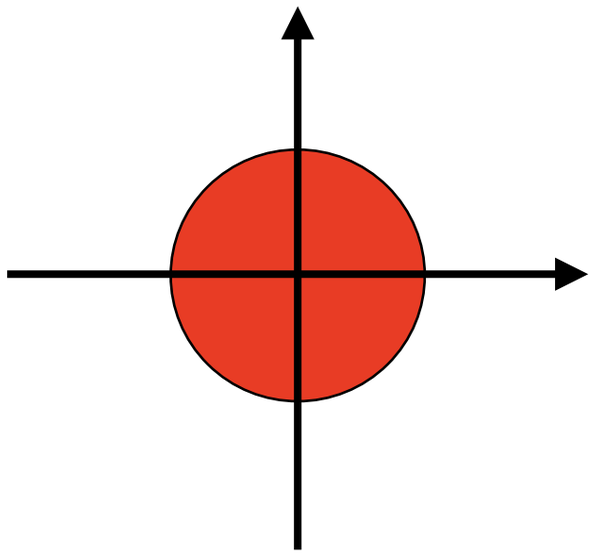
## How to systematically elicit implementation-level metrics


Application
Security requirements
Indicators
Pattern1
Pattern2
Pattern3
Metrics

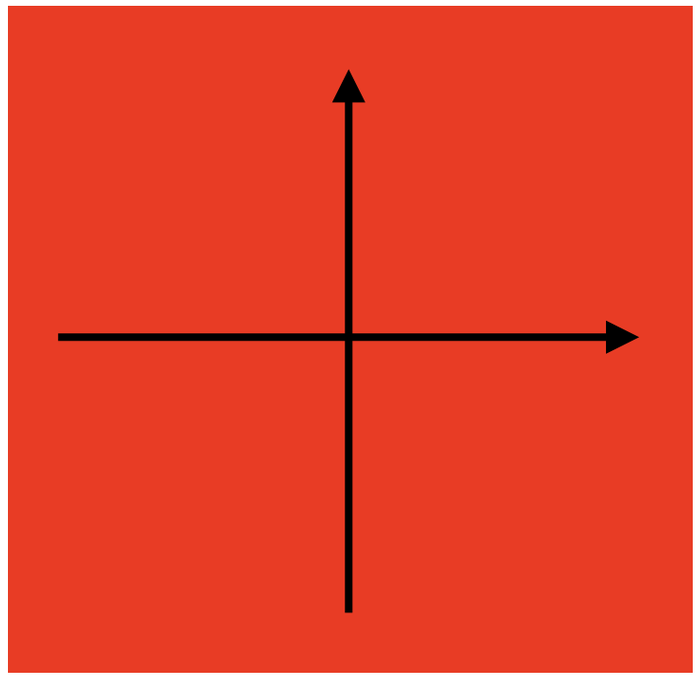
# Model checking ?



Testing



Model checking



Proof of correctness

# Our approach

Model

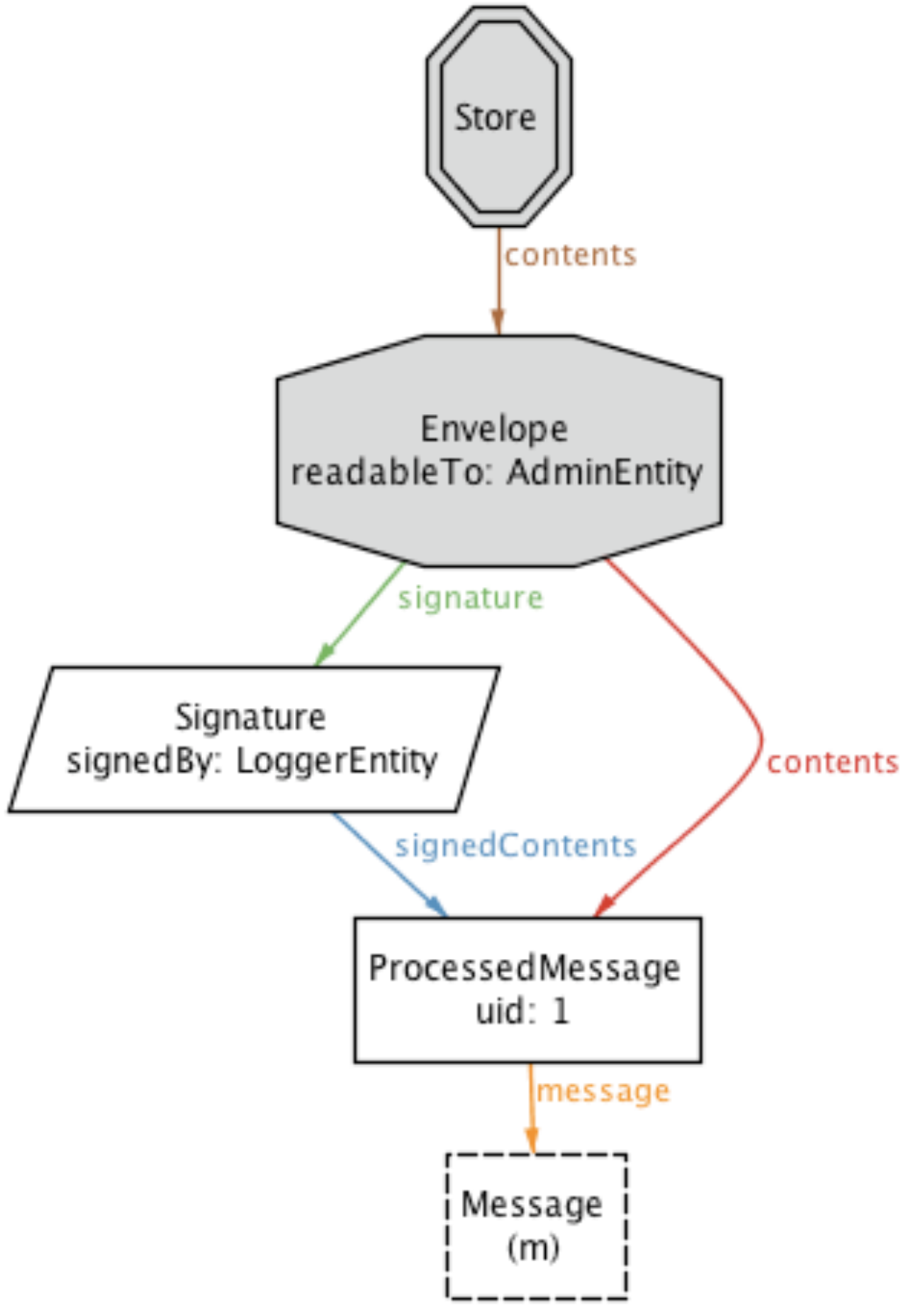
Isolate assumptions

Assess risk

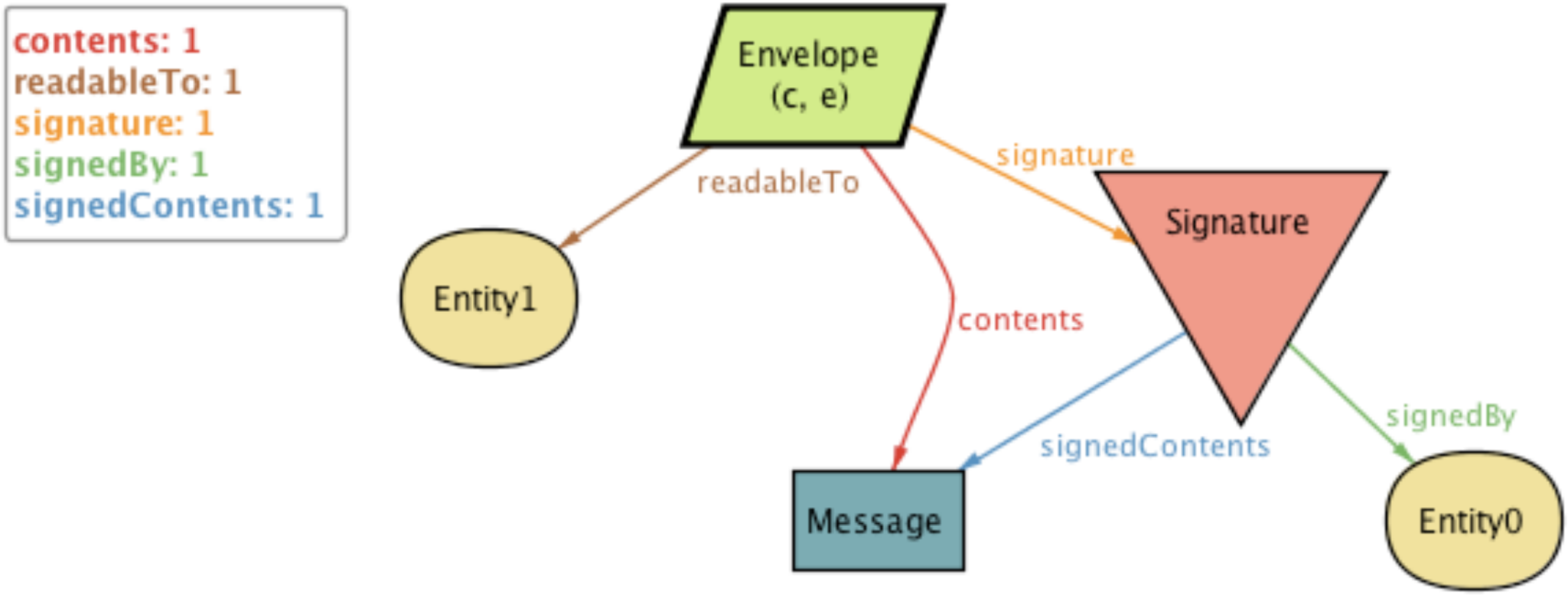
Accept, monitor, refine ?

# Case study:
# the **Secure Logger**

contents: 1
contents: 1
message: 1
signature: 1
signedContents: 1

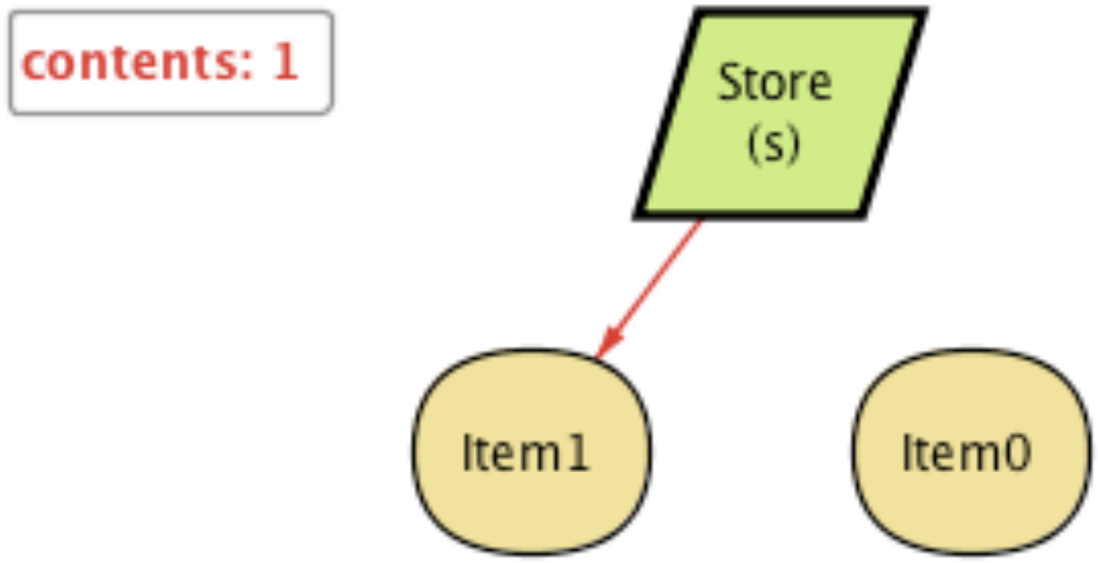


# Realistic crypto




```
pred Envelope.read(e: Entity) {
    e in this.readableTo
}
```

# Realistic storage




pred Store.create(newItem: Item, t: Time) { ...

**this.contents.(t.next) =**

**this.contents.t + newItem**

... }

Application

Security requirements

Indicators

Secure logger

**Naive logger**
*(protect next UID)*

**Crypto**
*(protect key)*

**Storage**
*(caching, space)*

Metrics

# Meta-metric

Percentage of assumptions covered

# Conclusion

# Questions?

thomas.heyman@cs.kuleuven.be