An (Interactive) Introduction to

SecFog

Stefano Forti

Department of Computer Science, University of Pisa, Italy

Reading: A. Brogi, S. Forti, G.-L. Ferrari, Secure Apps in the Fog: Anything to Declare?, 2018.

Security in the Fog

- The Fog will **increase** the number of **security enforcement points**.
- The Fog will **share** with the Cloud many **security threats** (e.g., virtualisation-related).
- The Fog will be **exposed to new**, peculiar **threats** due to physical accessibility of Fog nodes.

Security is one of the pillars of OpenFog Reference Architecture.

Considered Problem

The problem is known as *Component Deployment Problem* (CDP).

> Given
>
> - a **multi-component application** A with **requirements** R
> - a distributed **Fog infrastructure** I (nodes and links)
> - a set of **objective metrics** M
>
> determine **eligible application deployments** that meet all requirements in R and optimise metrics in M.

How difficult is CDP?

To prove P is NP-hard, we take another problem P' that is known to be NP-hard and we reduce P' to P in polynomial time, i.e. $P' \rightarrow_p P$.

This proves that P is at least as hard as P'. Hence, finding a poly-time algorithm that solves P would lead to solve P' in poly-time and prove that $P = NP$. (You can give it a try 😎)

Consider the **Subgraph Isomorphism Problem (SIP)** as our P':

> Given two graphs G and H, a solution to the **Subgraph Isomorphism Problem (SIP)** answers the question: is there a subgraph G' in G such that H can be mapped one-to-one, nodes and links, to G'?

CDP is NP-hard (in 4 steps)

> Given two graphs G and H, a solution to the **Subgraph Isomorphism Problem (SIP)** answers the question: is there a subgraph G' in G such that H can be mapped one-to-one, nodes and links, to G'?

How to reduce this to CDP in poly-time?

CDP is NP-hard (in 4 steps)

> Given two graphs G and H, a solution to the **Subgraph Isomorphism Problem (SIP)** answers the question: is there a subgraph G' in G such that H can be mapped one-to-one, nodes and links, to G'?

Step 1 Change all vertices v of G into Fog nodes in I with available resources set to 1.

Step 2 Change all edges in (u, v) of G into node-to-node links in I with available bandwidth set to 1.

Step 3 Change all v of H into components of A with resource requirements set to 1.

Step 4 Change all edges (u, v) of H into component-component interactions in A with bandwidth requirement set to 1.

With Steps 1-4 we can do (SIP) $\rightarrow_p$ (CDP). Hence CDP $\in$ NP-hard. ■

* *For all details of the proof see:* A. Brogi, S.Forti, QoS-aware Deployment of IoT Applications Through the Fog, 2017.

FogTorchΠ

1. solves CDP via **backtracking** search (exp-🕐, alas!)
2. estimates QoS-assurance by varying links QoS in 🎰 **Monte Carlo** simulations
3. estimates 📈 **Fog resource consumption** and 💶 **monthly deployment cost**

👮 **What about application deployment security?**

Our contribution

A **declarative methodology** to assess the **security level** of multi-component application deployments in Fog scenarios.

Security Capabilities

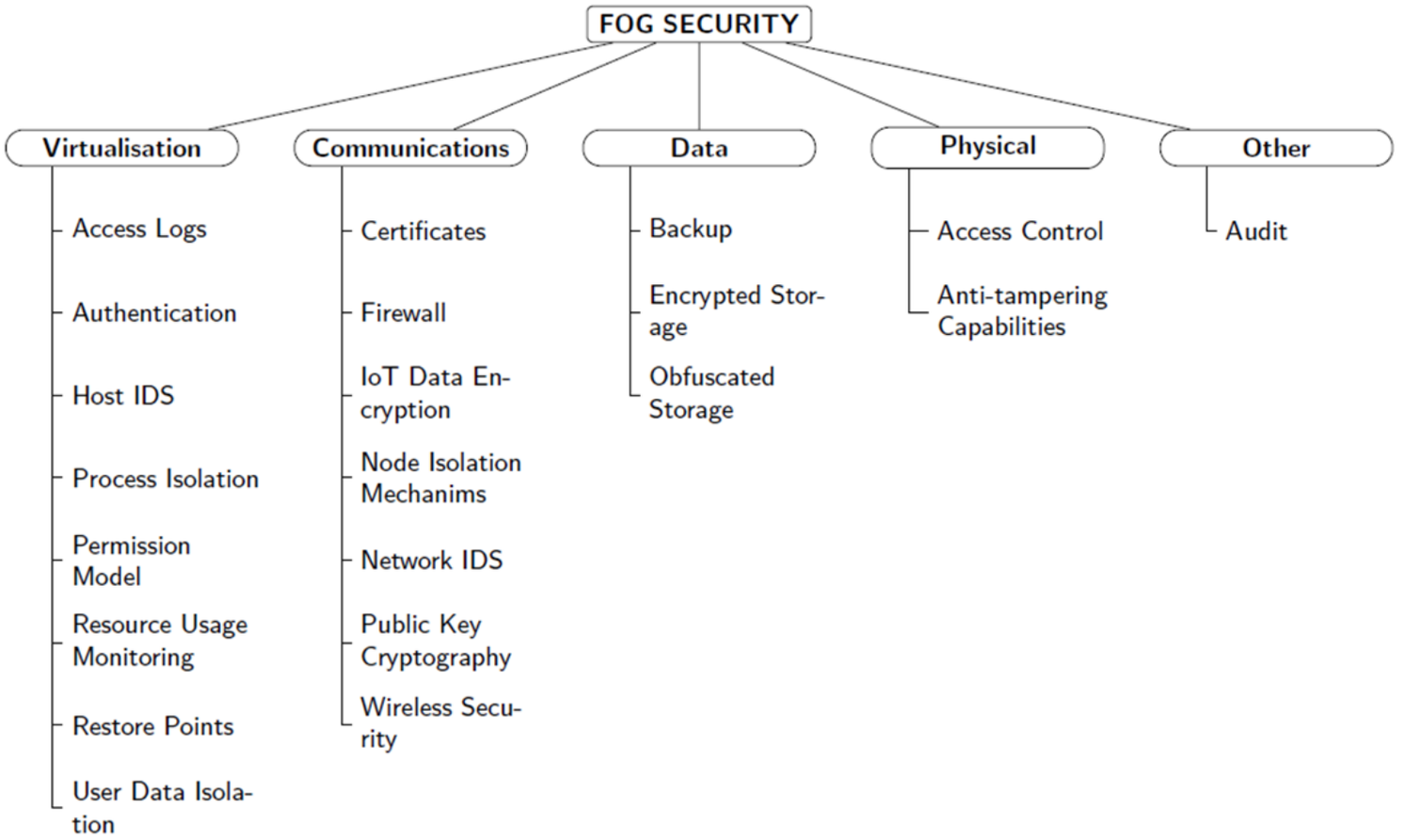
SecFog needs a **vocabulary** to specify **security capabilities** available/required in Fog scenarios.

Security control frameworks for the Cloud exists ☁️ (e.g., ISO/IEC 19086, EU Cloud SLA Standardisation Guidelines).

This is not the case for the Fog 🌁 😢

A Taxonomy

We hence proposed *our* taxonomy (based on recent surveys 📚)

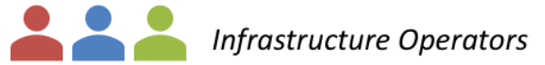


Big Picture

Infrastructure Operators

How does SecFog work?

Big Picture

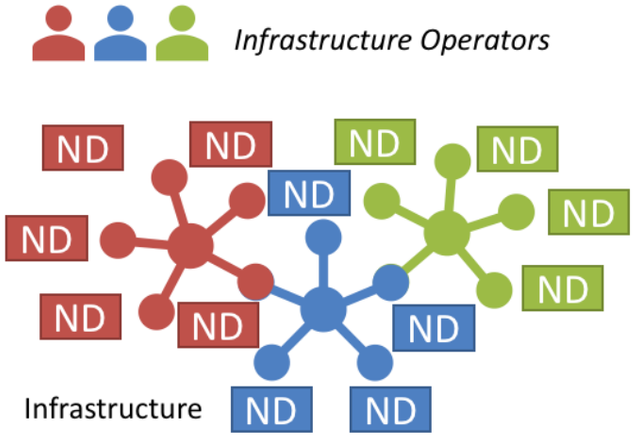


Infrastructure

Different (small, medium, large) **operators** manage the Fog infrastructure.

Indeed, Fog deployments will span various service providers.

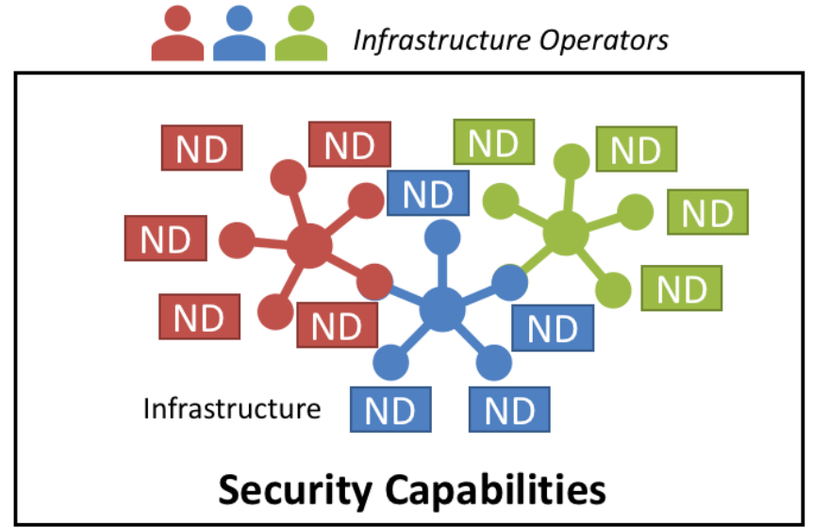
Big Picture



SecFog

We assume that each node will self-describe its capabilities (and their *reliability*) through a **Node Descriptor** (ND) using the taxonomy vocabulary.

Big Picture

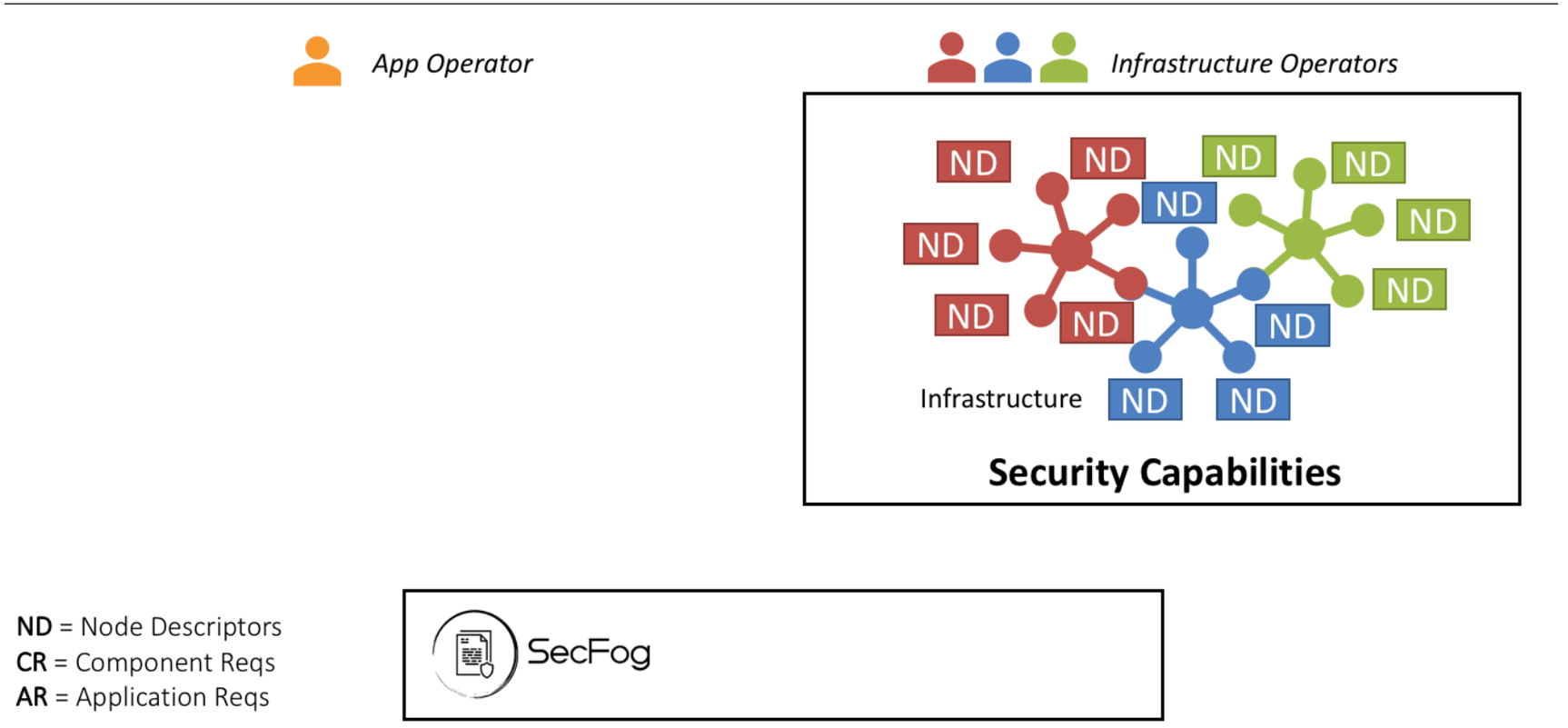


ND = Node Descriptors
CR = Component Reqs
AR = Application Reqs

SecFog

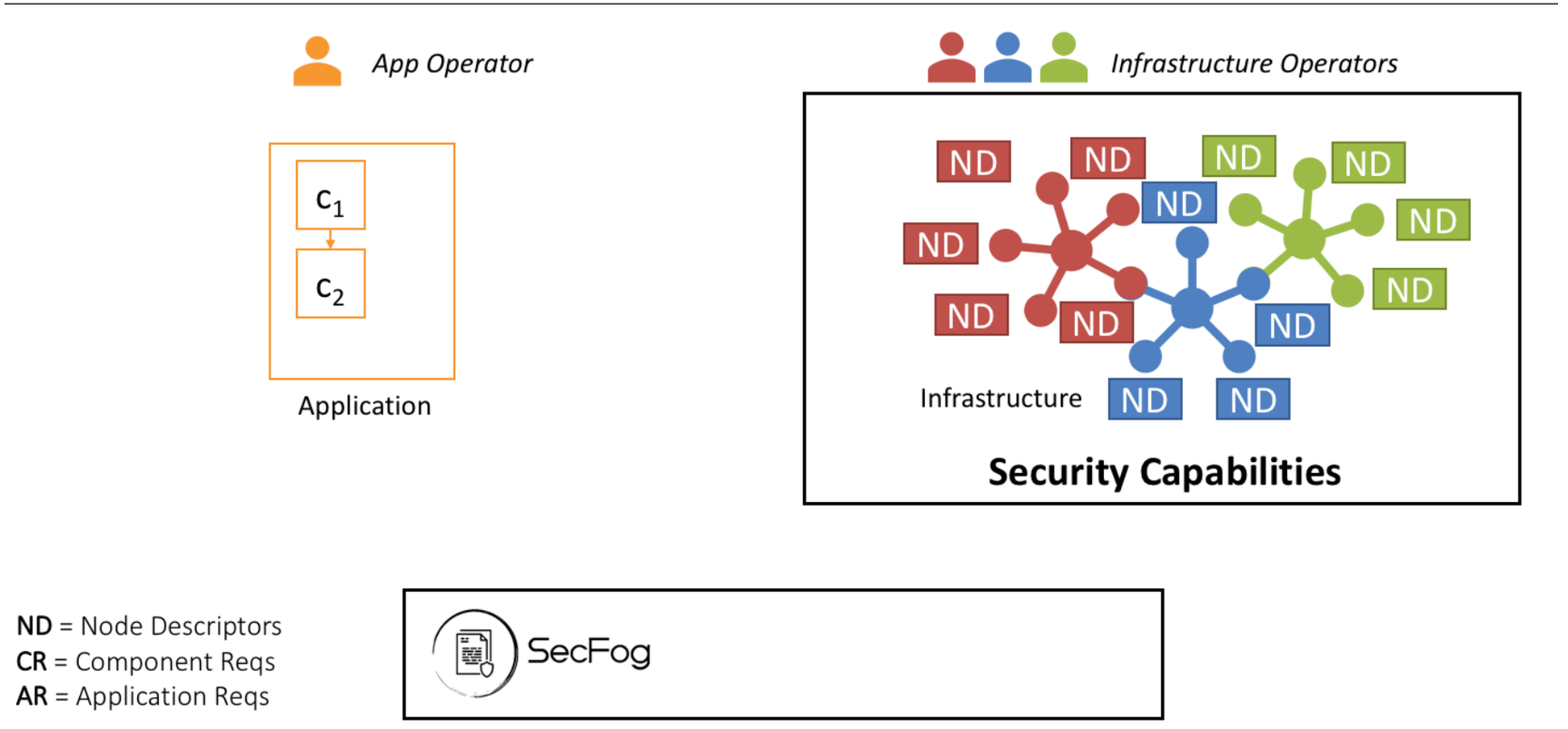
In this way, we get a complete description of the available **Security Capabilities**.

Big Picture



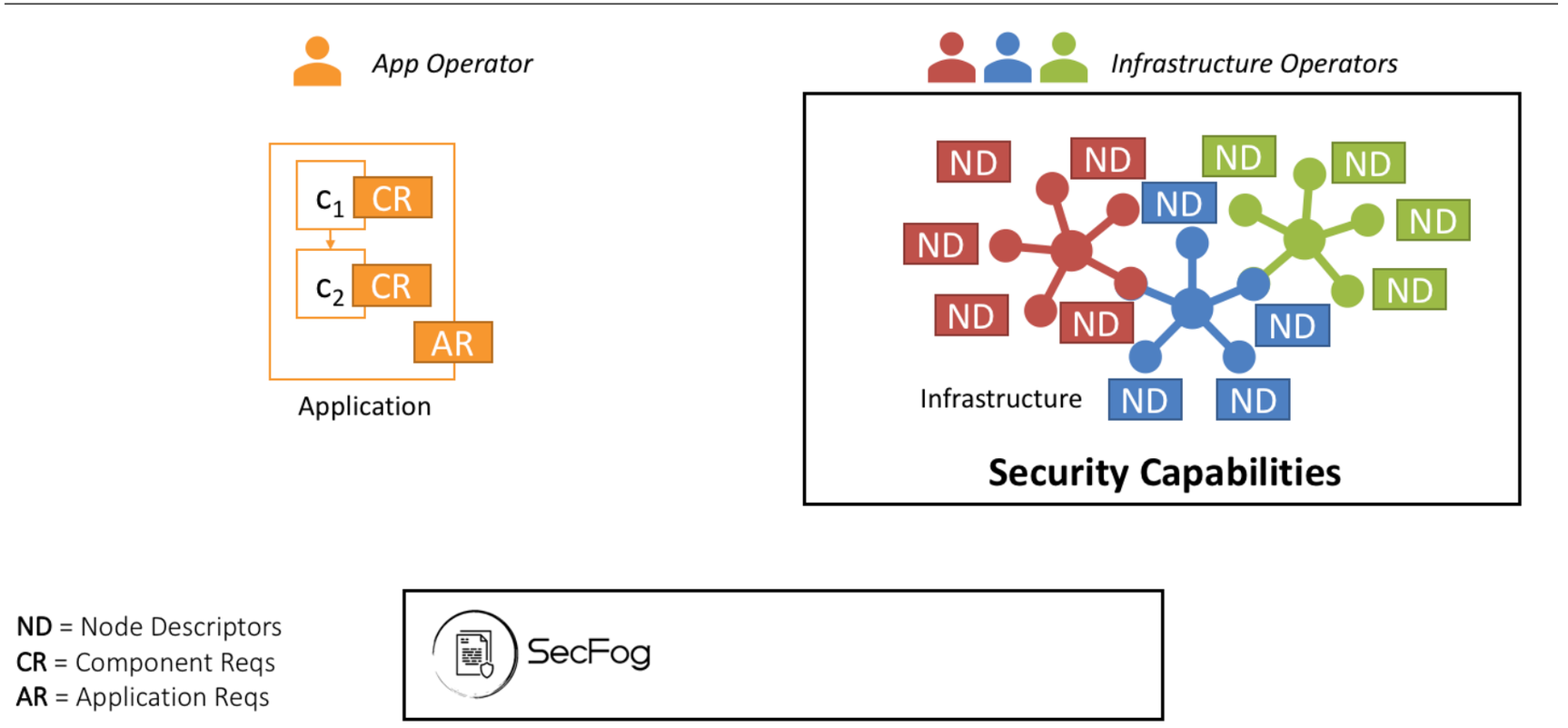
On the other hand, we consider the **App Operator** that will describe their app.

Big Picture



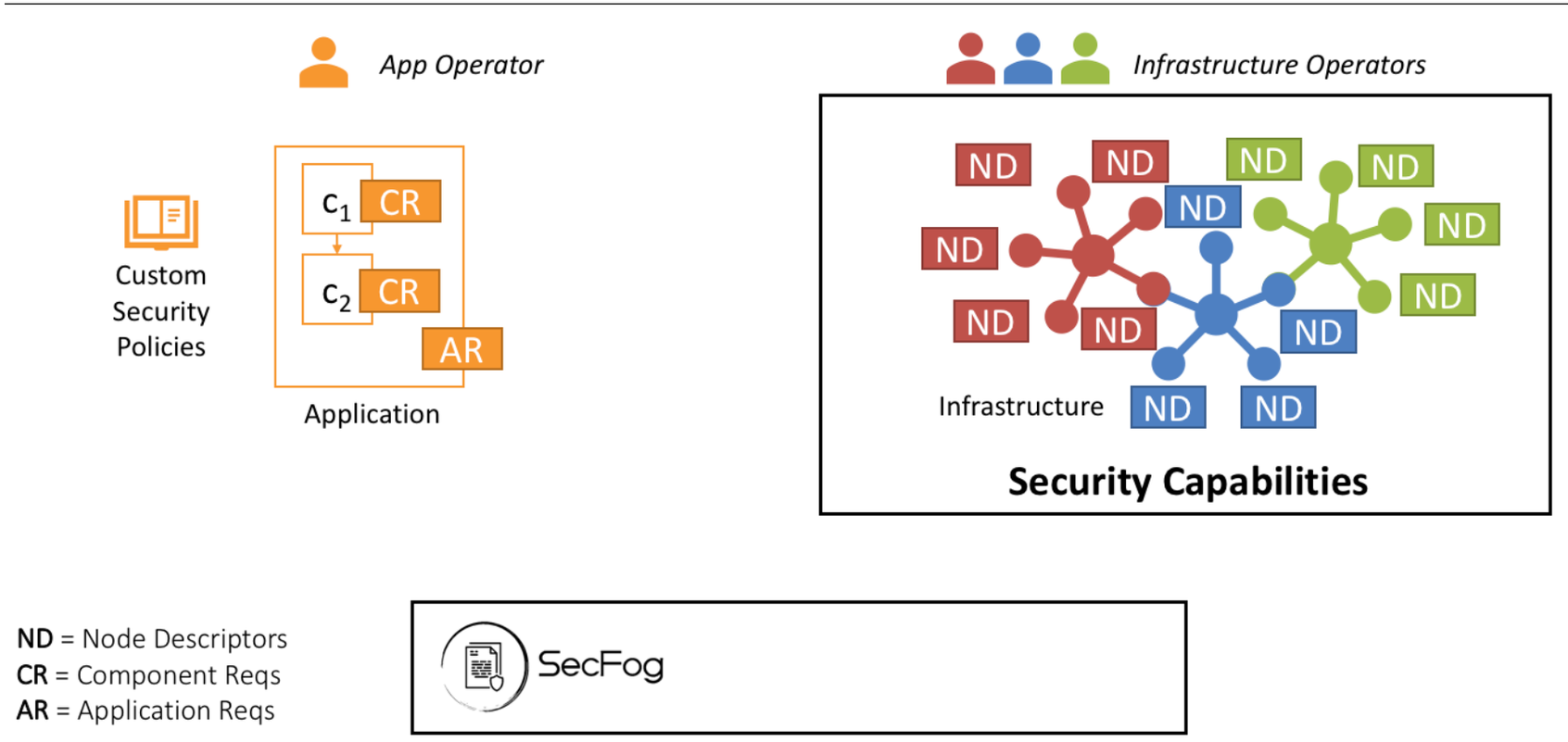
This can be done by specifying the **app topology**...

Big Picture



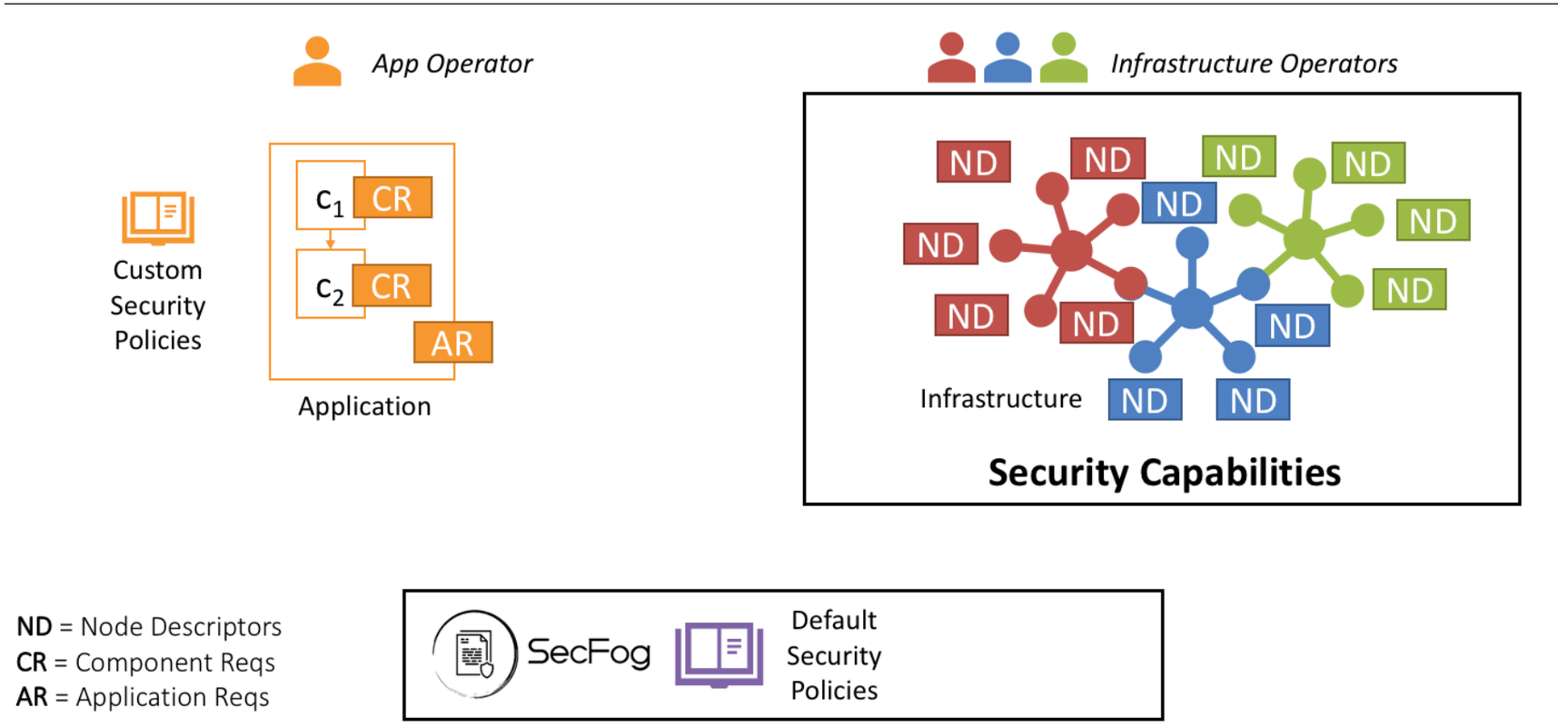
... and by annotating it with security requirements both at the **component** (CR) and **application** (AR) level, again relying on the taxonomy dictionary.

Big Picture



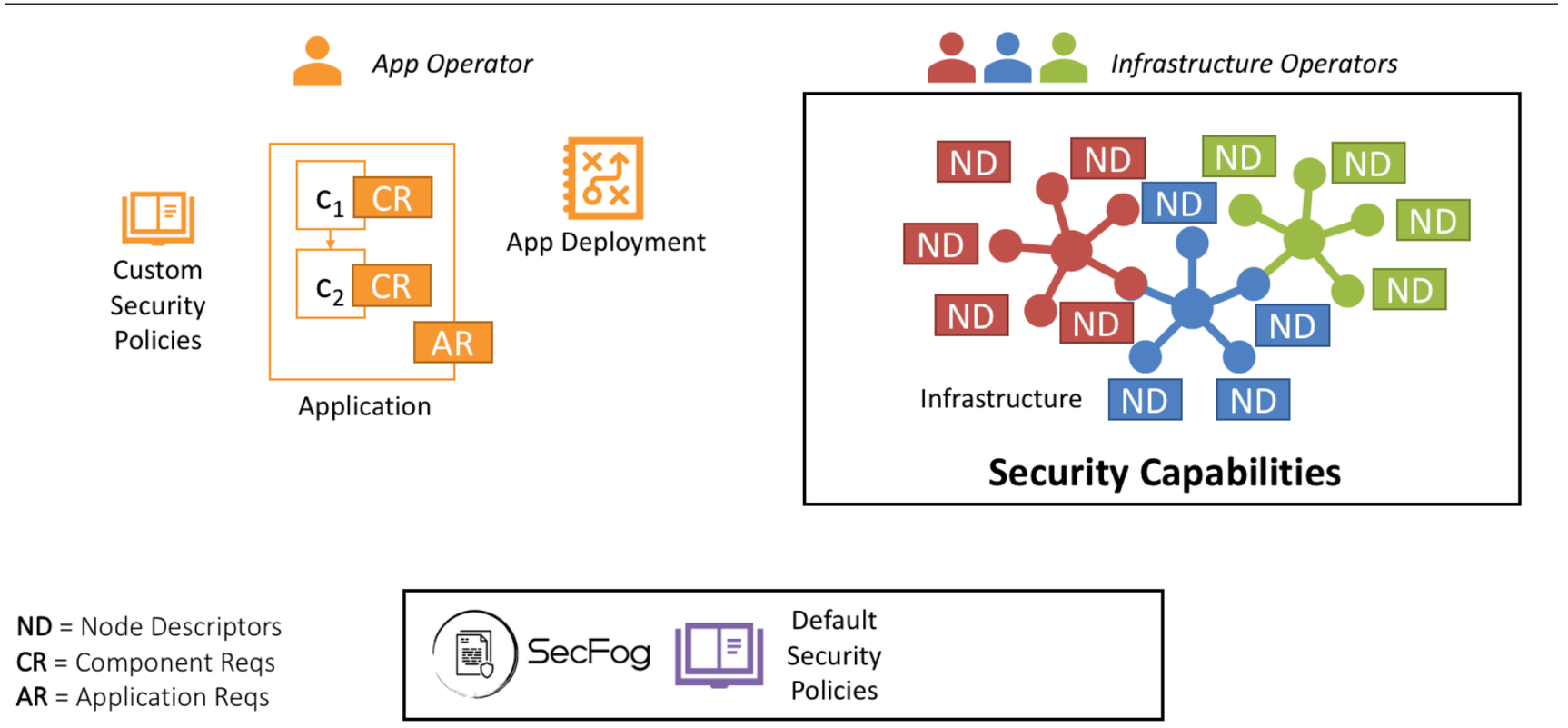
AR and CR can be expressed in term of **Custom Security Policies**.

Big Picture



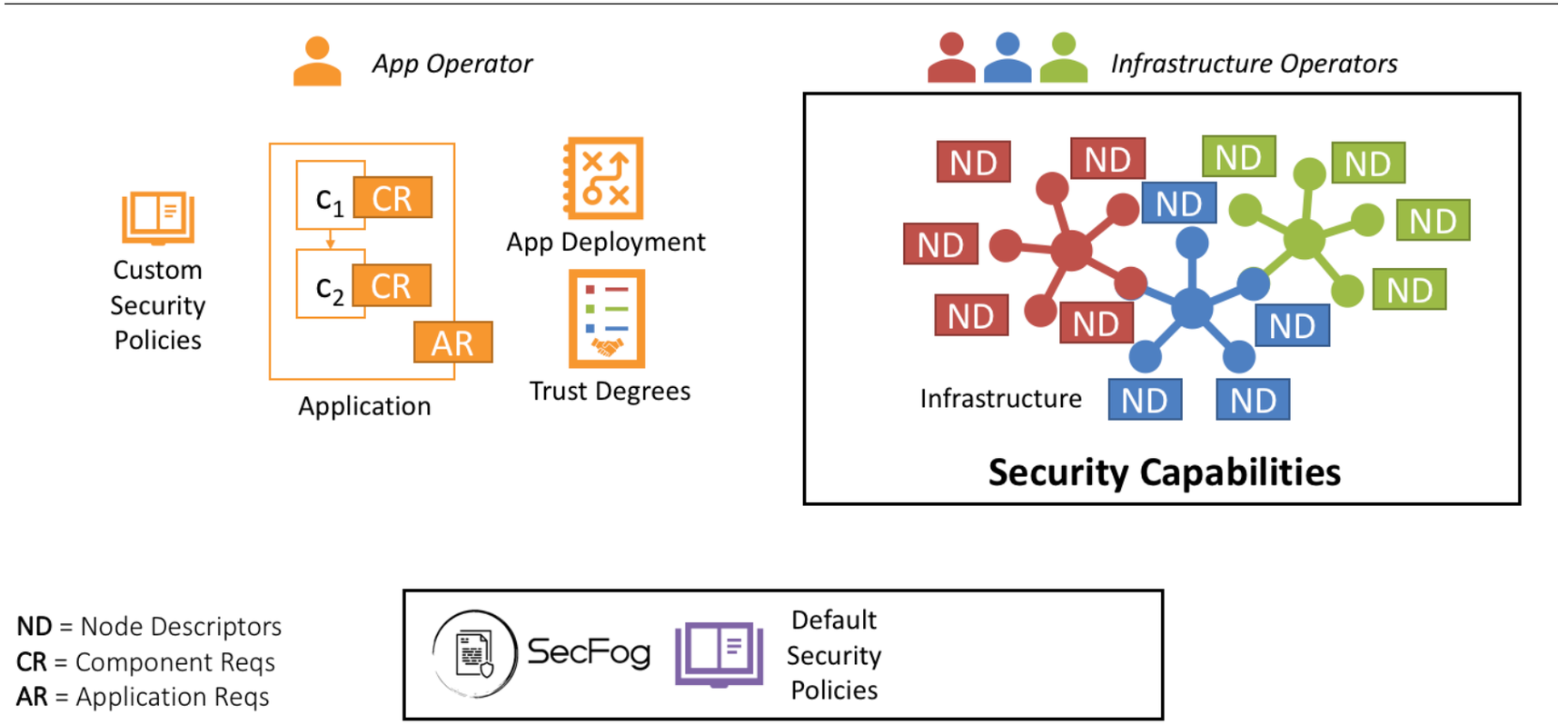
Custom Security Policies are expressed in terms of the **Default Security Policies** of SecFog.

Big Picture



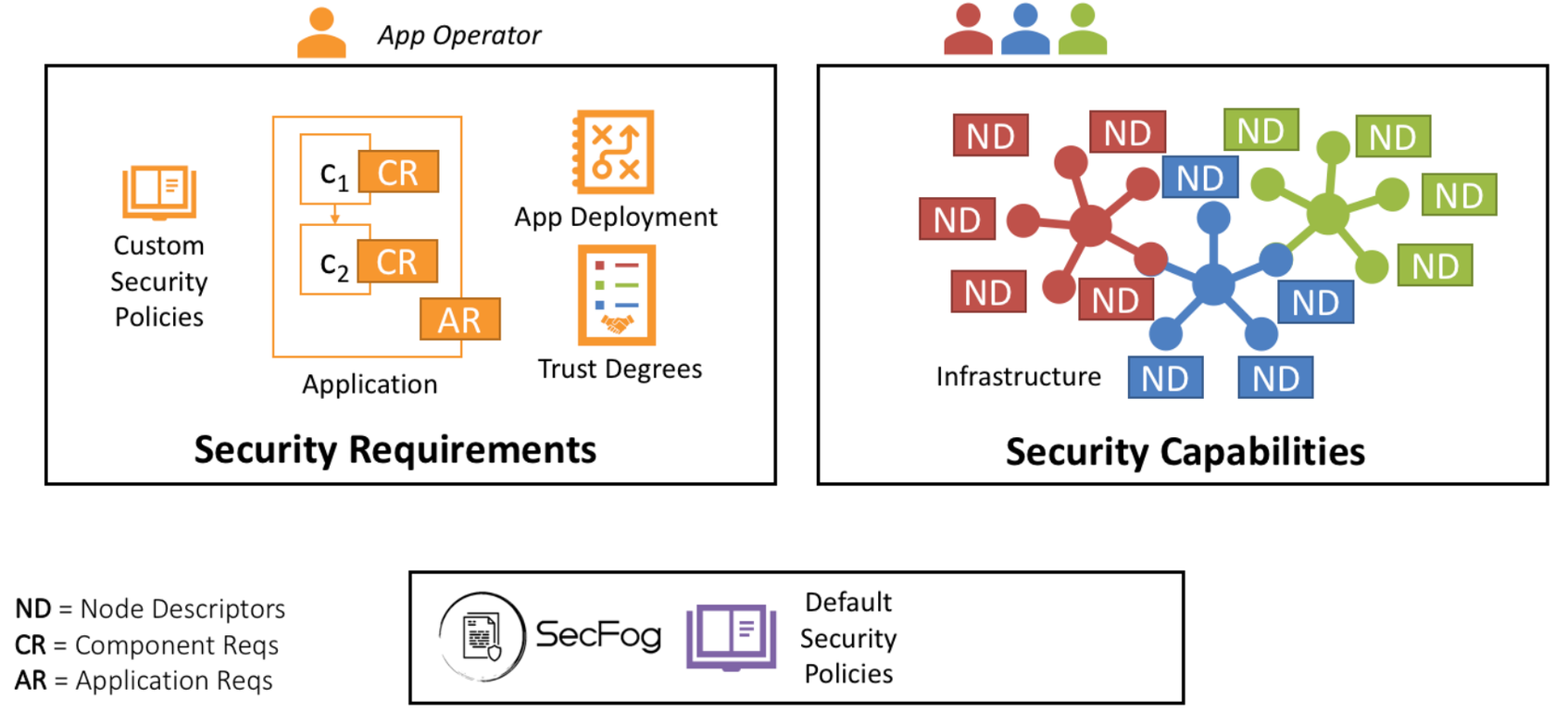
The App Operator can also specify complete or partial **App Deployments** of their application.

Big Picture



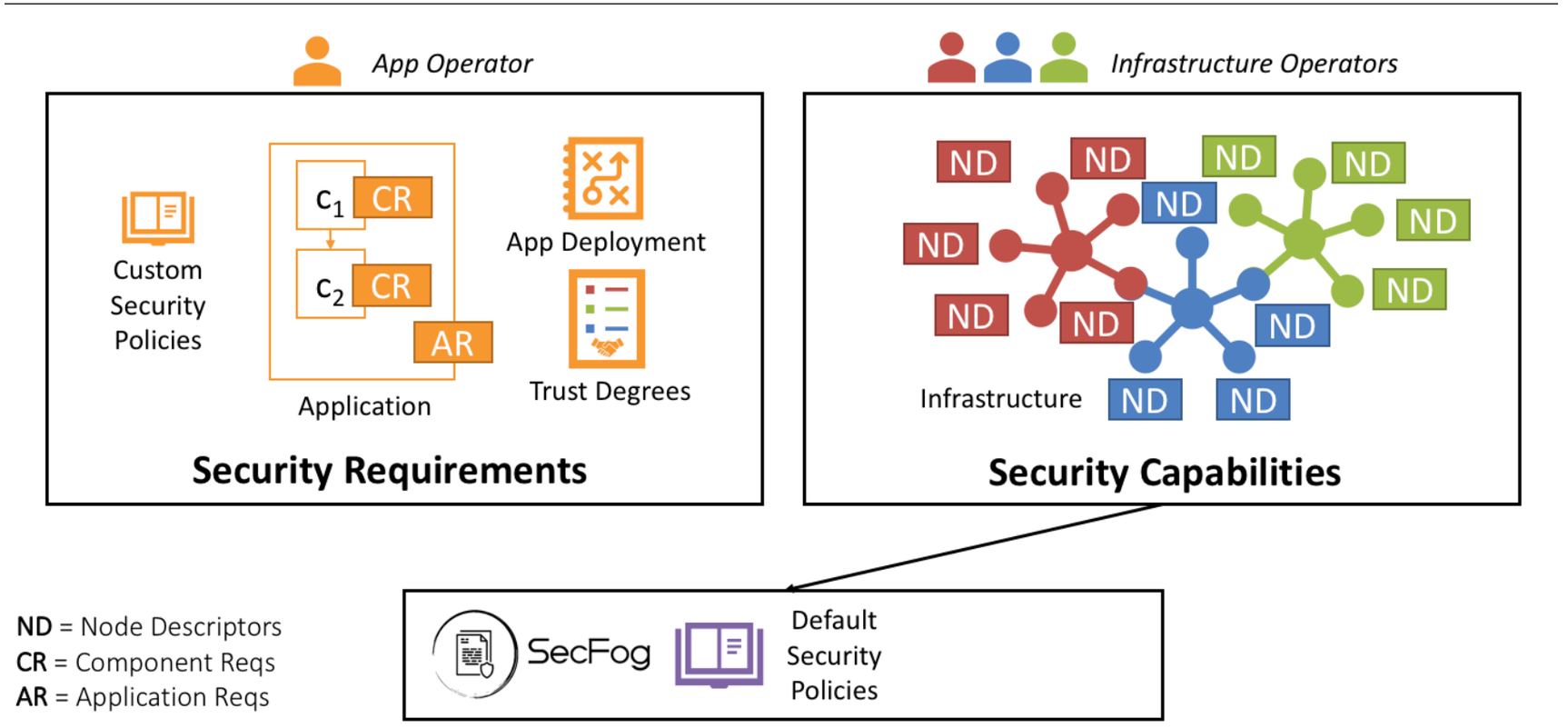
Finally, the App Operator can specify the **Trust Degrees** towards different Infrastructure Operators.

Big Picture



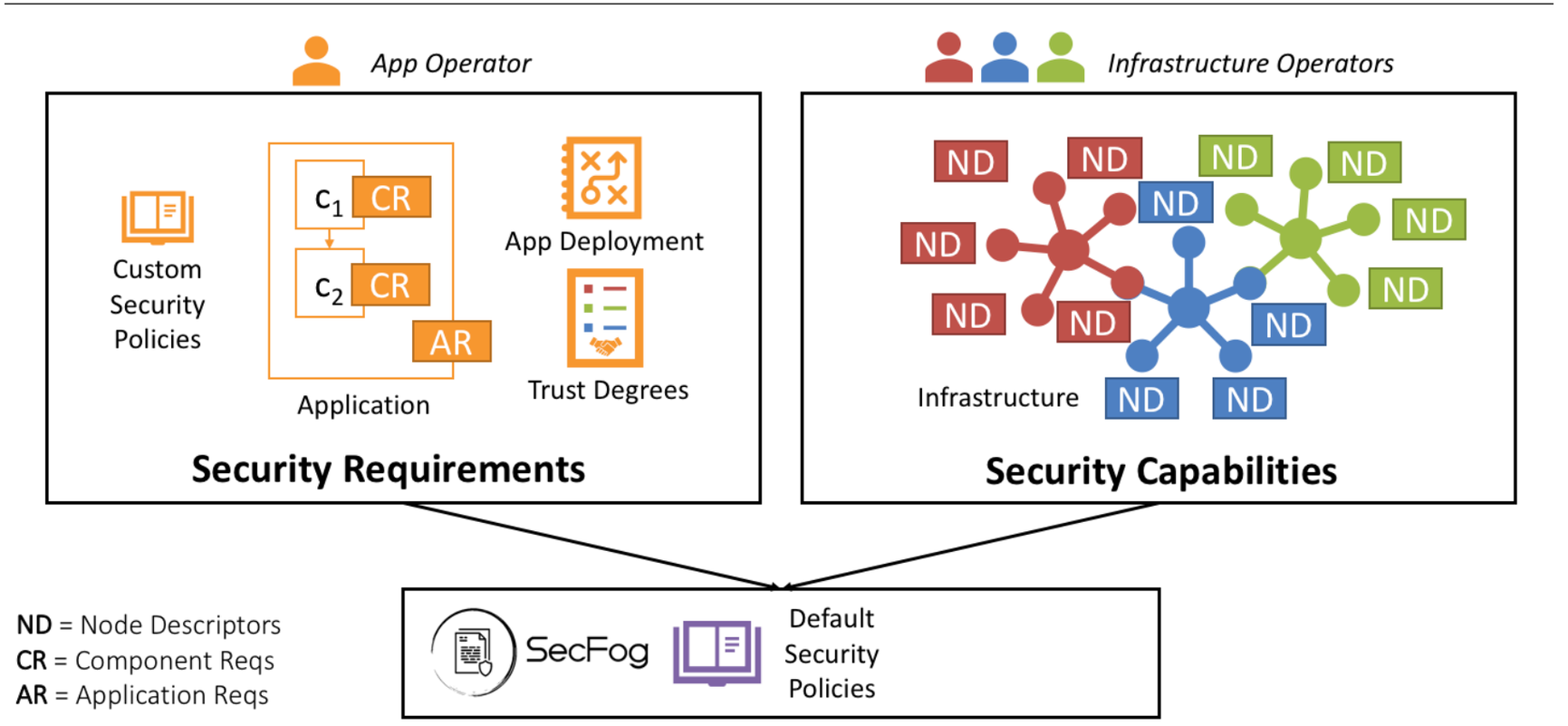
All this constitutes the **Security Requirements** of the considered multi-component app.

Big Picture



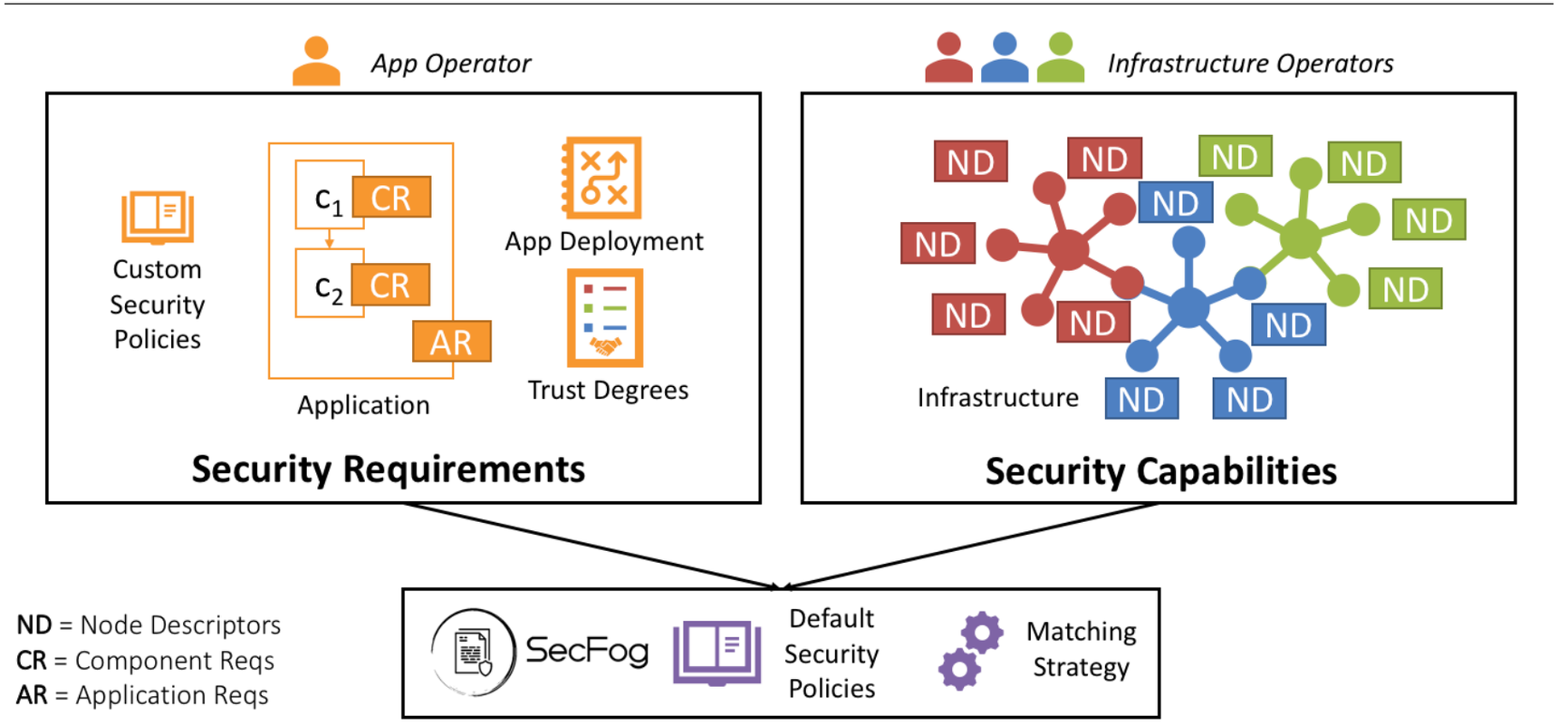
SecFog takes as input the Security Capabilities and...

Big Picture



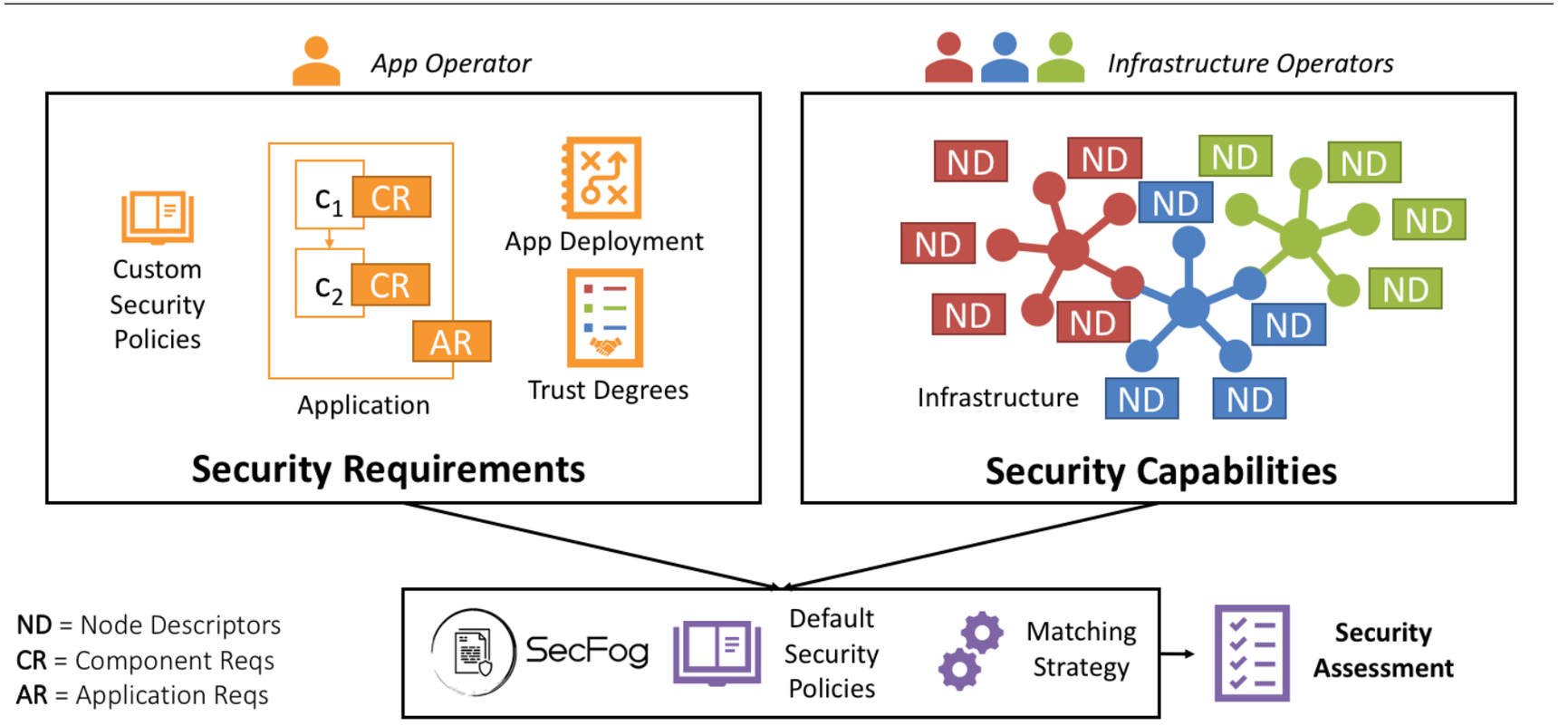
... the Security Requirements.

Big Picture



A matching strategy is used to determine secure deployments...

Big Picture



... and to assess their **Security Level**.

Implementation

SecFog has been prototyped using the **ProbLog2** language.

The prototype:

1. **Matches** application security **requirements with** infrastructure **capabilities** (*generate & test* approach), and
2. Obtains the **security level** of each deployment by multiplying the **reliability of** all exploited **security capabilities**, weighted by **trust degrees**.

It is *only* sixteen (16) lines of code 💻

(🎉 We are gonna write them all! 🎉)

A note on point (2) of the previous slide...

If I use a single lock for my bike, the security level of me locking my bike is the "certified" **reliability of the lock** $\times$ **my level of trust in the lock vendor** (has (s)he got a cut of my key?)

A note on point (2) of the previous slide...

What if I'm in Pisa and I decide to use **more locks**?

Now change locks with security capabilities and vendor with infrastructure operator...

Problog 101

ProbLog is a Python package that permits writing logic programs that encode complex interactions between large sets of heterogeneous components, capturing the inherent uncertainties that are present in real-life situations.

Problog programs are composed of **facts***

```
p::f.  % statement f is true with probability p
```

and **rules**

```
r :- c1, ..., cn.  % property r is inferred when c1 & ... & cn hold
```

* A fact declared simply as `f.` is assumed to be `true` with probability `1`.

Problog 101

- ProbLog programs are logic programs in which some of the facts are annotated with (their) probabilities.
- Each program defines a probability distribution over logic programs where a fact `p::f.` is considered `true` with probability `p` and false with probability `1-p`.
- The ProbLog engine determines the success probability of a query `q` as the probability that `q` has *a* proof, given the distribution over logic programs.

👓 ☕ We are ready to implement SecFog!

SecFog - Deployments

Given a list of app components `[C|Cs]`, we can *recursively* define a **deployment** as the association of each component `C` to a node `N`, i.e. `d(C,N)`:

```
:- use_module(library(lists)).

deployment([],[]). % an empty set of components leads to an empty deployment
deployment([C|Cs],[d(C,N)|D]) :-
        node(N,_),
        deployment(Cs,D).
```

We can try this (also online):

```
node(fog1, fogOp1). % node(id, operatorId)
node(fog2, fogOp2).
node(cloud1, cloudOp).

query(deployment([iot_controller, data_storage, dashboard], _)).
```

SecFog - Secure Deployment

We first write a rule to check:

1. if a component `C` is deployed to an existing node `N` (as per deployment `D`), and
2. that the infrastructure operator `Op` managing `N` is trustable according to the application operator.

```
secure(C, N, D) :-
        member(d(C,N), D), % is the couple d(C,N) in D?
        node(N, Op),
        trustable(Op).
```

Try it:

```
0.8::trustable(fogOp1).
0.5::trustable(fogOp2).
query(secure(iot_controller, _,[d(iot_controller,_)])).
```

SecFog - Secure Application Deployment

Now that we have a way to check that an app component is securely deployed, we write rules to recursively check that the application as a whole was securely deployed.

```
secureApp(A,D) :-
    app(A, L),
    deployment(L,D),
    secureComponents(A,L,D).

secureComponents(A, [], _).
secureComponents(A, [C|Cs],D) :-
    secureComponent(C,N,D),
    secureComponents(A, Cs,D).
```

📦 That's all folks! These are the default security policies of SecFog.
As promised, they are only 16 lines of code (import's excluded 😏).

How to use SecFog

The app operator needs to define a `secureComponent(C,N,D)` for each app component, always including the predicate `secure(C, N, D)`.

Then, it is possible to run the query:

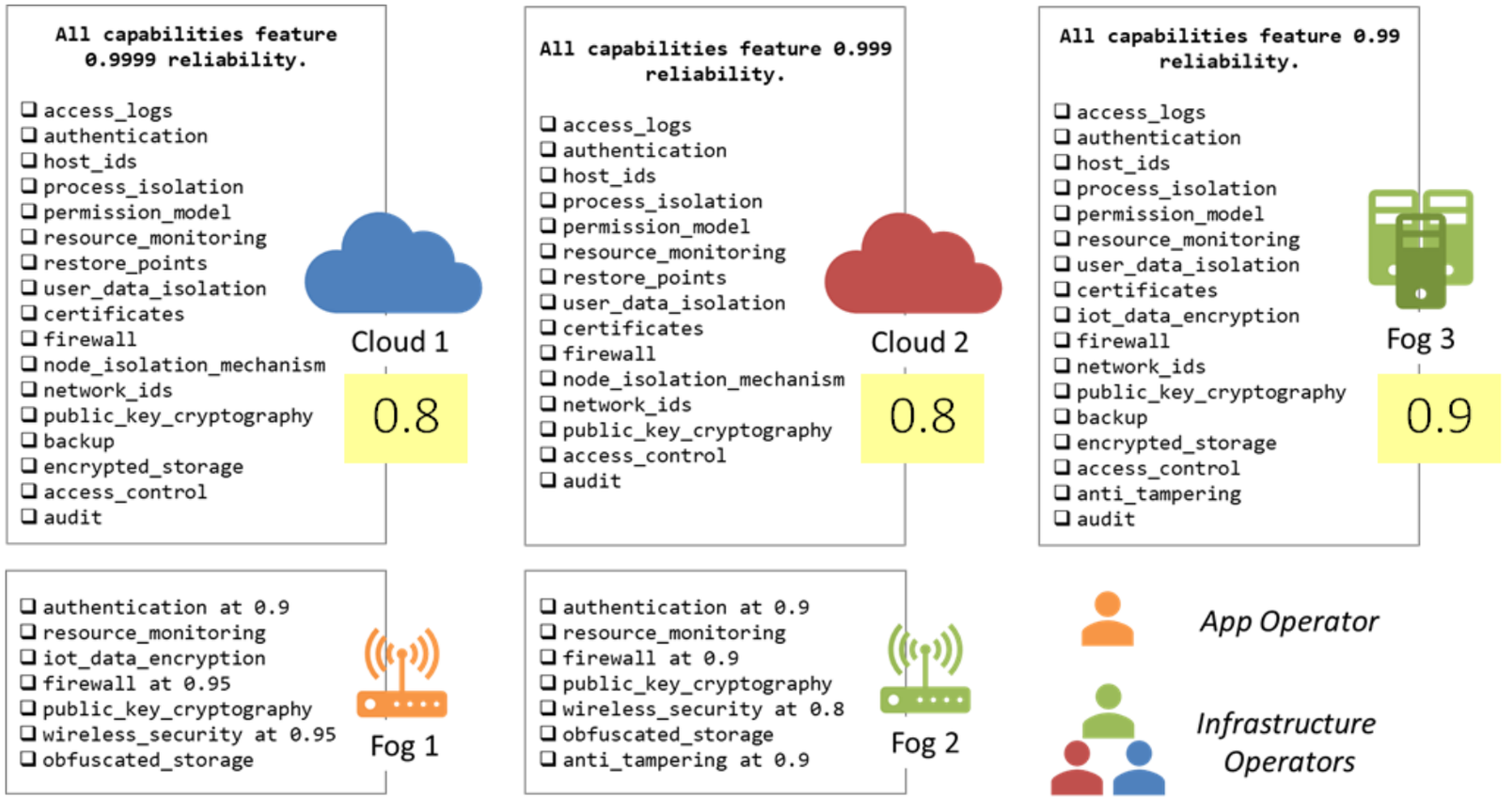
```
query(secureApp(A,_)).
```

to obtain all secure deployments and their security level.

Let's see a **complete example**.

Example: Node Descriptors

The file `infrastructure.pl` programmatically declares the Node Descriptors for the infrastructure skecthec below. How do we include trust degrees (as per the yellow boxes)?



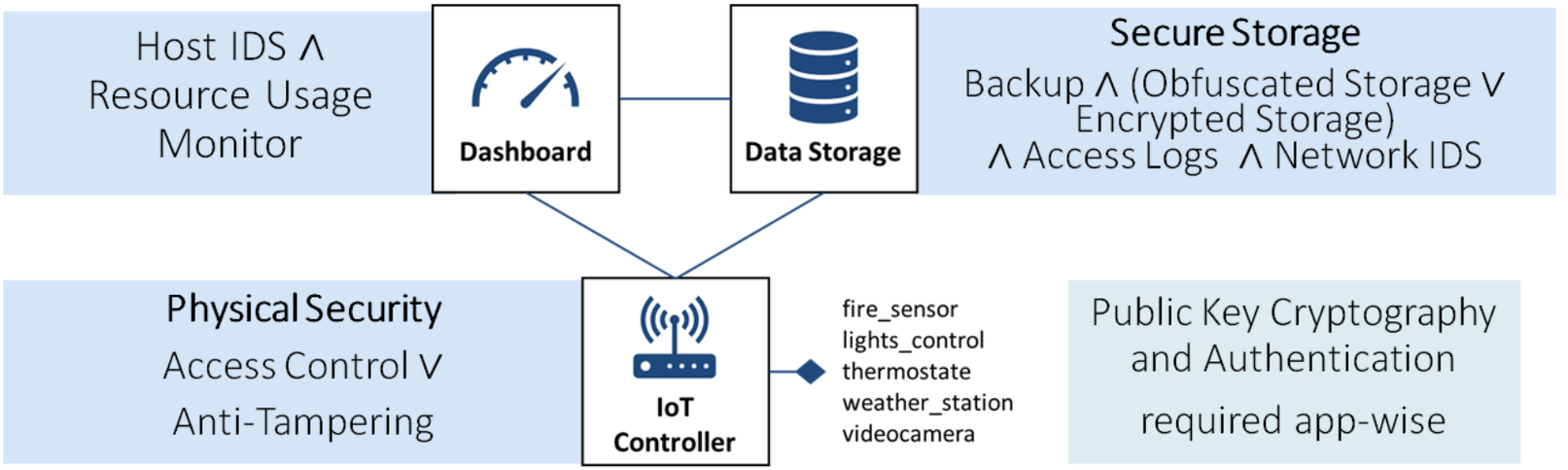
Example: Trust Degrees

Yes 👍 as done before!

```
%trust degrees
0.8::trustable(cloudOp1).
0.8::trustable(cloudOp2).
0.9::trustable(fogOp).
trustable(appOp).
```

Example: Application

The application



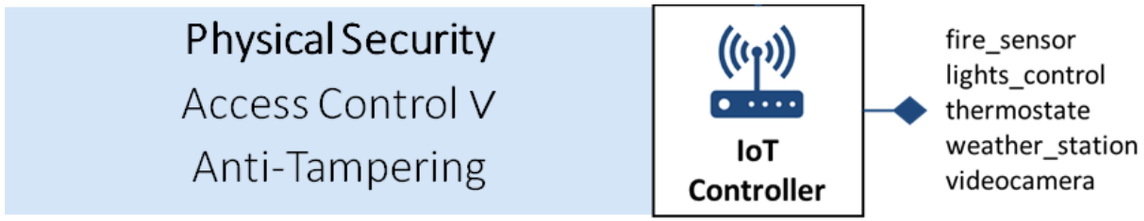
can be specified as:

```
app(smartbuilding, [iot_controller, data_storage, dashboard]).
```

What about its **security requirements**?

Example: Component Security Requirements

The requirements of the IoT Controller component



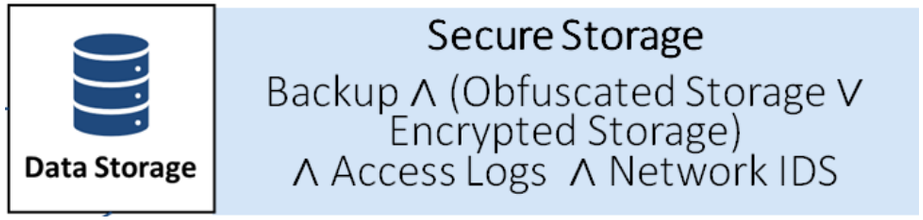
can be specified as:

```
physical_security(N) :- anti_tampering(N); access_control(N).

secureComponent(iot_controller, N, D) :-
    physical_security(N),
    secure(iot_controller, N, D).
```

Example: Component Security Requirements

The requirements of the DataStorage component



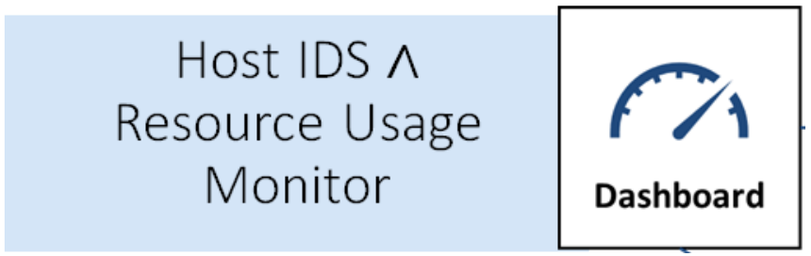
can be specified as:

```
secure_storage(N) :-
    backup(N),
    (encrypted_storage(N); obfuscated_storage(N)).

secureComponent(data_storage, N, D) :-
    secure_storage(N),
    access_logs(N),
    network_ids(N),
    secure(data_storage, N, D).
```

? Pop Quiz ?

Can you write component security requirements for the Dashboard?



Any volunteers?

```
secureComponent(dashboard, N, D) :-
```

Example: App Security Requirements

Finally, app-wise security requirements

Public Key Cryptography and Authentication required app-wise

can be specified as:

```
extras([]).
extras([d(C,N)|Ds]) :-
    public_key_cryptography(N),
    authentication(N),
    extras(Ds).
```

Example: Query

In this case, we (re-)define also:

```
mySecureApp(A,D) :-
    secureApp(A,D),
    deployment(L,D),
    extras(D).
```

Which will be used to query the system on eligible deployments and security levels:

```
query(mySecureApp(A,_)).
```

(All `example.pl` code is now on Moodle)

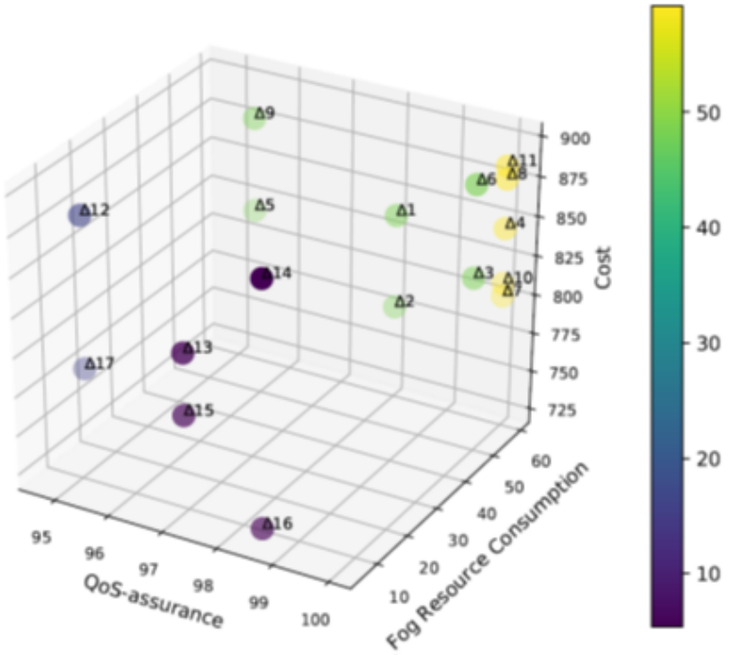
To run the programmed specification, issue the command

```
$ problog ./example.pl
```

SecFog Output

```
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,cloud1), d(dashboard,cloud1)]): 0.79928029
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,cloud1), d(dashboard,cloud2)]): 0.63699776
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,cloud1), d(dashboard,fog3)]): 0.69114513
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,fog3), d(dashboard,cloud1)]): 0.67752684
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,fog3), d(dashboard,cloud2)]): 0.53996463
mySecureApp(smartbuilding,[d(iot_controller,cloud1), d(data_storage,fog3), d(dashboard,fog3)]): 0.66417689
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,cloud1), d(dashboard,cloud1)]): 0.63757163
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,cloud1), d(dashboard,cloud2)]): 0.63642441
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,cloud1), d(dashboard,fog3)]): 0.55131415
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,fog3), d(dashboard,cloud1)]): 0.54045108
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,fog3), d(dashboard,cloud2)]): 0.67448315
mySecureApp(smartbuilding,[d(iot_controller,cloud2), d(data_storage,fog3), d(dashboard,fog3)]): 0.66238504
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,cloud1), d(dashboard,cloud1)]): 0.5827336
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,cloud1), d(dashboard,cloud2)]): 0.46441781
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,cloud1), d(dashboard,fog3)]): 0.55988355
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,fog3), d(dashboard,cloud1)]): 0.54885163
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,fog3), d(dashboard,cloud2)]): 0.54687823
mySecureApp(smartbuilding,[d(iot_controller,fog2), d(data_storage,fog3), d(dashboard,fog3)]): 0.67268088
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,cloud1), d(dashboard,cloud1)]): 0.70503715
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,cloud1), d(dashboard,cloud2)]): 0.56188935
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,cloud1), d(dashboard,fog3)]): 0.69114513
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,fog3), d(dashboard,cloud1)]): 0.67752684
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,fog3), d(dashboard,cloud2)]): 0.67509079
mySecureApp(smartbuilding,[d(iot_controller,fog3), d(data_storage,fog3), d(dashboard,fog3)]): 0.83038718
```

FogTorchΠ Output



Dep. ID	IoTController	DataStorage	Dashboard	QoS	Resources	Cost
$\Delta 1$	Fog 2	Fog 3	Cloud 2	98.6%	48.4%	€856.7
$\Delta 2$	Fog 2	Fog 3	Cloud 1	98.6%	48.4%	€798.7
$\Delta 3$	Fog 3	Fog 3	Cloud 1	100%	48.4%	€829.7
$\Delta 4$	Fog 2	Fog 3	Fog 1	100%	59.2%	€844.7
$\Delta 5$	Fog 1	Fog 3	Cloud 1	96%	48.4%	€837.7
$\Delta 6$	Fog 3	Fog 3	Cloud 2	100%	48.4%	€887.7
$\Delta 7$	Fog 3	Fog 3	Fog 2	100%	59.2%	€801.7
$\Delta 8$	Fog 3	Fog 3	Fog 1	100%	59.2%	€875.7
$\Delta 9$	Fog 1	Fog 3	Cloud 2	96%	48.4%	€895.7
$\Delta 10$	Fog 1	Fog 3	Fog 2	100%	59.2%	€809.7
$\Delta 11$	Fog 1	Fog 3	Fog 1	100%	59.2%	€883.7
$\Delta 12^*$	Fog 2	Cloud 2	Fog 1	94.7%	16.1%	€870.7
$\Delta 13^*$	Fog 2	Cloud 2	Cloud 1	97.2%	5.4%	€824.7
$\Delta 14^*$	Fog 2	Cloud 2	Cloud 2	98.6%	5.4%	€882.7
$\Delta 15^*$	Fog 2	Cloud 1	Cloud 2	97.2%	5.4%	€785.7
$\Delta 16^*$	Fog 2	Cloud 1	Cloud 1	98.6%	5.4%	€727.7
$\Delta 17^*$	Fog 2	Cloud 1	Fog 1	94.7%	16.1%	€773.7

How to combine QoS-assurance, resource usage, cost and security? 💡 Any ideas?

Multi-objective Optimisation

We can **rank** deployments according to

$$r(\Delta) = \sum_{m \in M} \omega_m \times \overline{m(\Delta)}$$

where

- M is the set of metrics to be optimised,
- ω_m is the weight assigned to each metric, and
- $\overline{m(\Delta)}$ is the normalised value of the metric m for deployment Δ.

The better the deployment, the higher the ranking 👑

Results

Most probably, we aim at:

- *maximising* QoS-assurance and security
- *minimising* cost

For Fog resource usage we might go either:

- **Cloud-ward** = minimise Fog resource usage
- **Fog-ward** = maximise Fog resource usage

				Fog-ward	*Cloud-ward*
Dep. ID	IoTController	DataStorage	Dashboard	$r_F(\Delta)$	$r_C(\Delta)$
$\Delta 1$	Fog 2	Fog 3	Cloud 2	0.53	0.28
$\Delta 2$	Fog 2	Fog 3	Cloud 1	0.63	0.38
$\Delta 3$	Fog 3	Fog 3	Cloud 1	0.85	0.60
$\Delta 6$	Fog 3	Fog 3	Cloud 2	0.75	0.50
$\Delta 15^*$	Fog 2	Cloud 1	Cloud 2	0.15	0.40
$\Delta 16^*$	Fog 2	Cloud 1	Cloud 1	0.51	0.76

Conclusions

SecFog **help(s)** **Fog app operators** **who** **want to** **determine secure app deployments** **by** **reducing manual tuning** **and** **considering app reqs, infrastructure capabilities and trust** **unlike** **existing approaches***

* With which SecFog can be synergically used, as shown with FogTorchΠ.

Future Work

- 🔐 Better quantify trust degrees and reliability (perhaps using a semi-ring other than probability)
- ⌚ Improve scalability over large infrastructures via meta-heuristics (e.g., bio-inspired).
- ✏️ Engineer the whole FogTorchΠ + SecFog methodology into a (micro-)service and experiment with them in lifelike settings.

🤓 🟦 Yes, if you liked the ~~Fog~~ course... you can ask us for a thesis!

An (Interactive) Introduction to

Stefano Forti

Department of Computer Science, University of Pisa, Italy

Reading: A. Brogi, S. Forti, G.-L. Ferrari, Secure Apps in the Fog: Anything to Declare?, 2018.

This slideset was realised using Marp: **https://github.com/yhatt/marp**