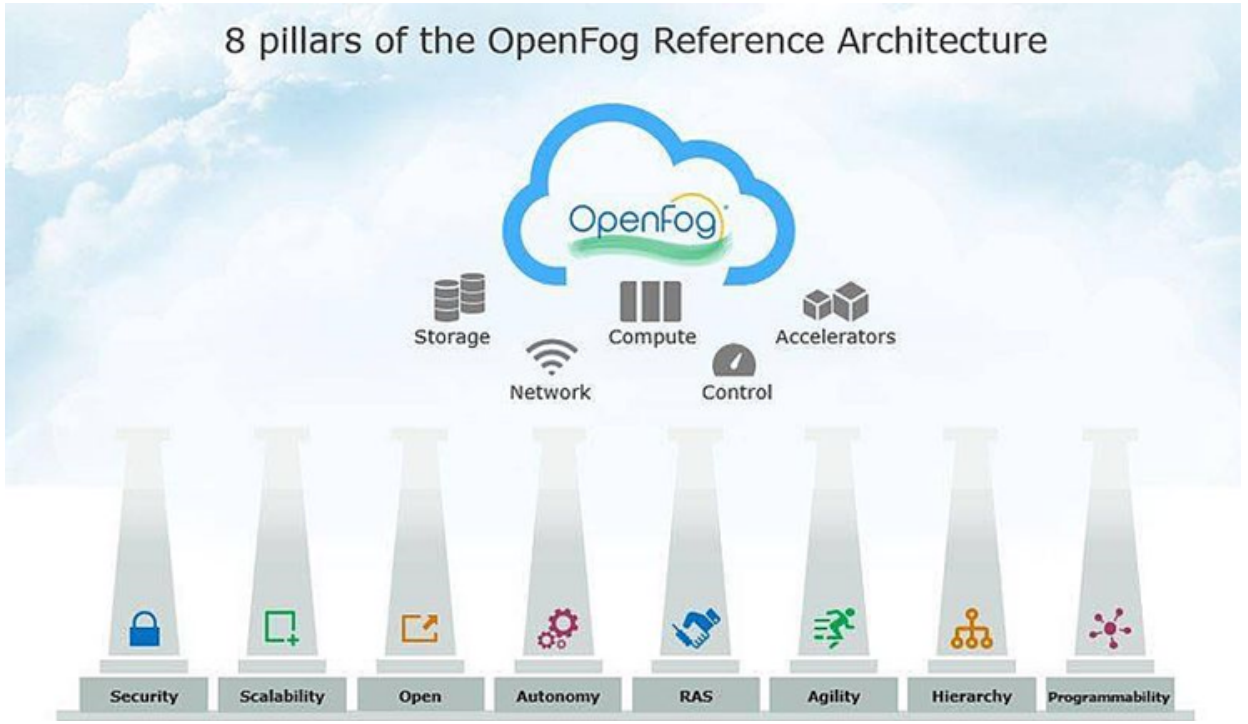
An (Interactive) Introduction to

Stefano Forti

Department of Computer Science, University of Pisa, Italy

Reading: S. Forti, G.-L. Ferrari, A. Brogi, [Secure Cloud-Edge Deployments, with Trust], 2021.

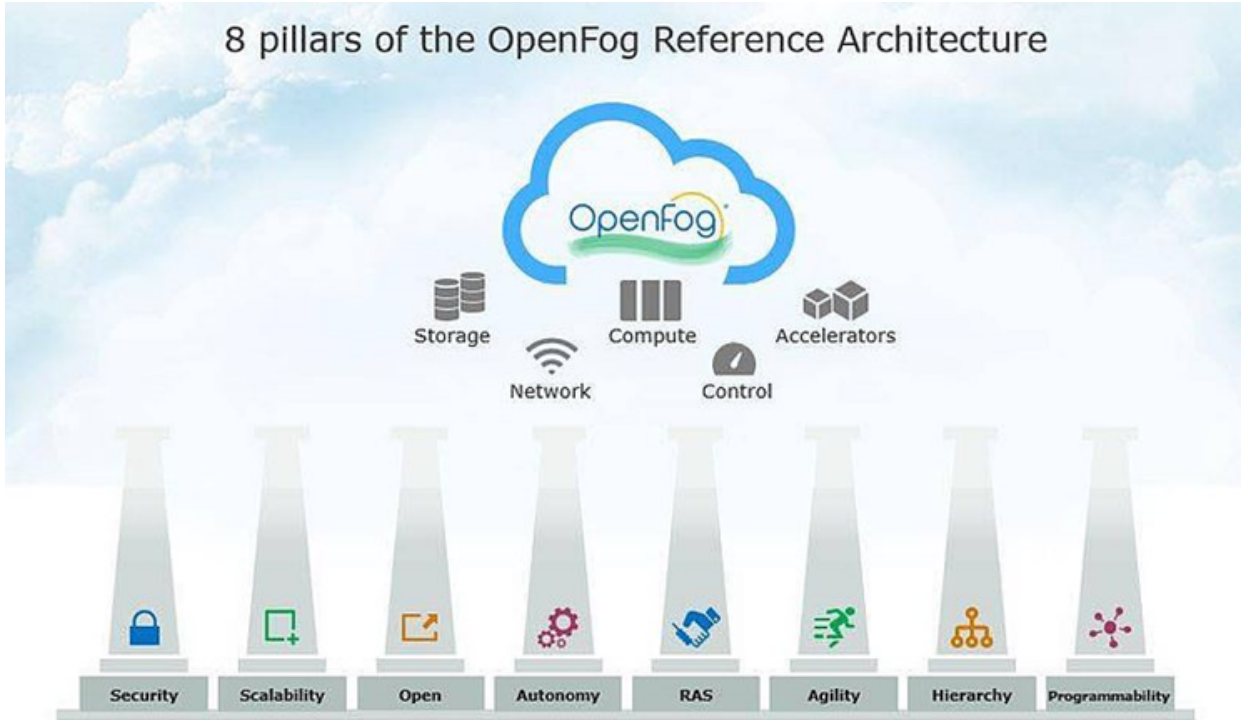
Security in the Fog



Security is one of the pillars of OpenFog Reference Architecture.

Security in the Fog

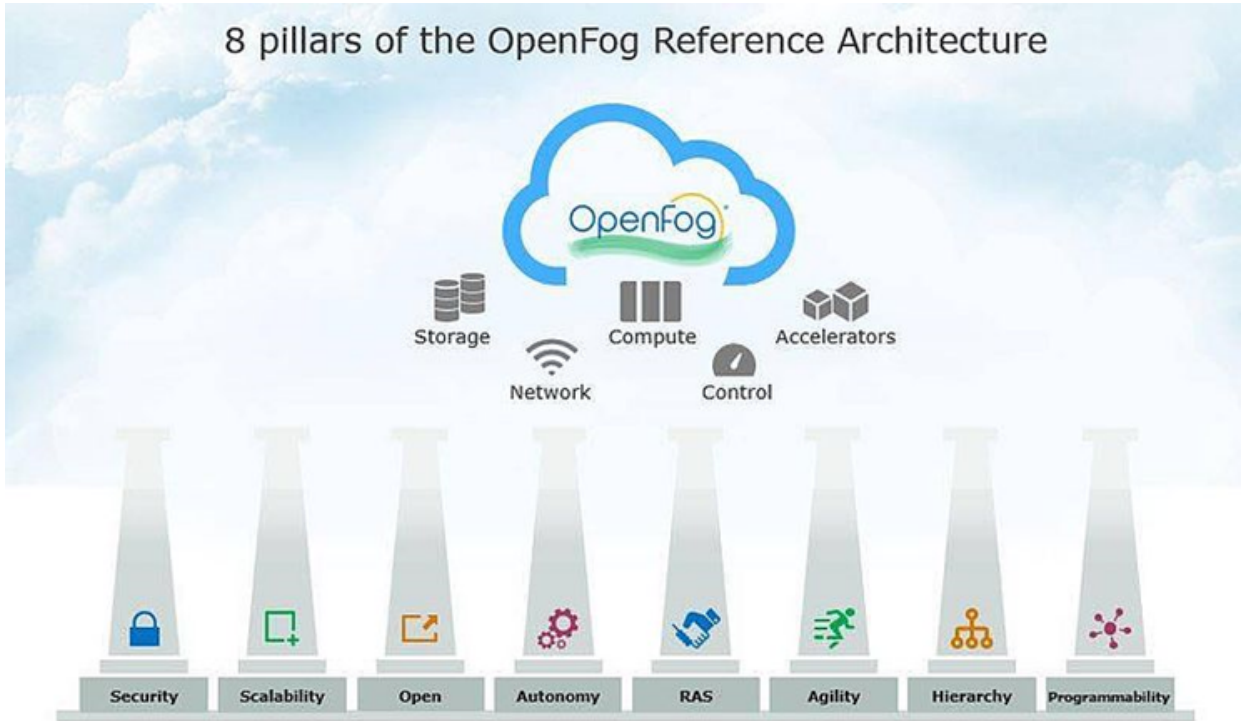
- The Fog will **increase** the number of **security enforcement points**.



Security is one of the pillars of OpenFog Reference Architecture.

Security in the Fog

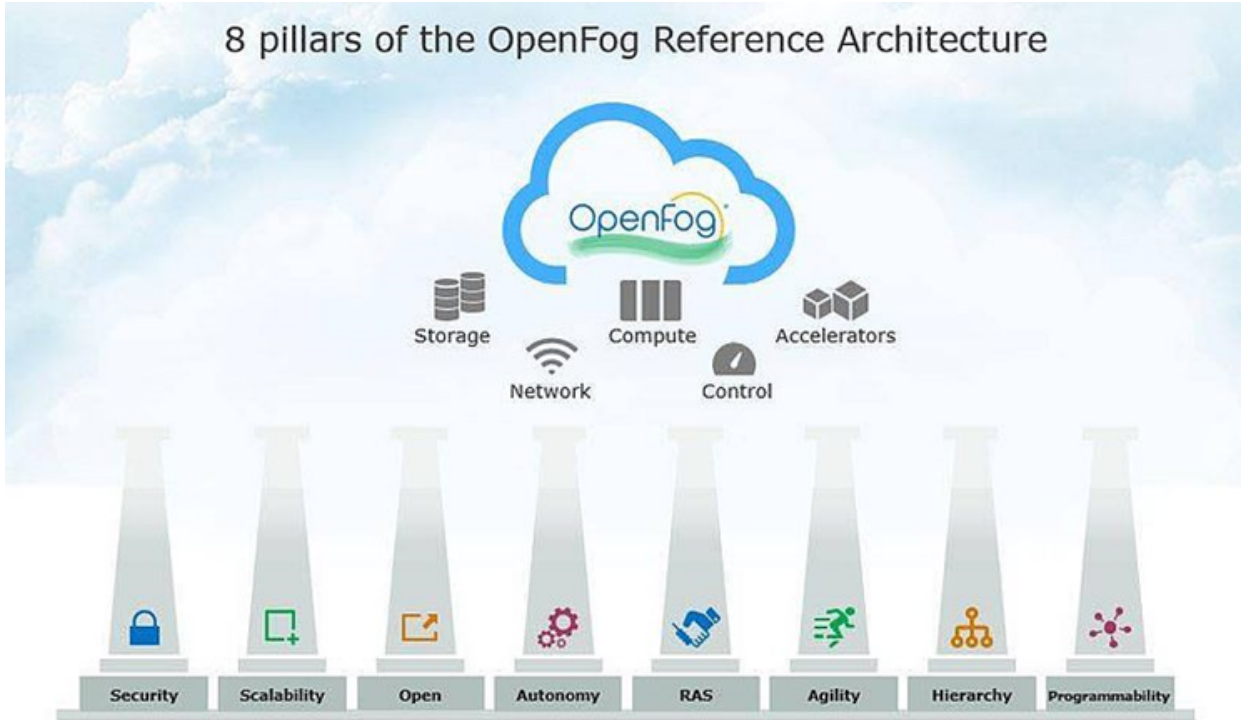
- The Fog will **increase** the number of **security enforcement points**.
- The Fog will **share** with the Cloud many **security threats** (e.g., virtualisation-related).



Security is one of the pillars of OpenFog Reference Architecture.

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).

Considered Problem

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

> Given
>
> - a **multi-component application** A with **requirements** R

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)

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

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.

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. $P' \rightarrow_p P$.

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 😎)

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.

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.

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.

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.

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.

Declarative Application Placement

1. solves CDP via **backtracking** search (exp-🕐, alas!)

Declarative Application Placement

1. solves CDP via **backtracking** search (exp-🕐, alas!)
2. can estimate QoS-assurance by varying links QoS exploiting probabilistic declarations of nodes and links 🎰

Declarative Application Placement

1. solves CDP via **backtracking** search (exp-🕐, alas!)
2. can estimate QoS-assurance by varying links QoS exploiting probabilistic declarations of nodes and links 🎰

👮 **What about application deployment security?**

Our contribution

A **declarative methodology** to assess the **security level** of multi-component application deployments in Fog scenarios, whilst considering trust relations among involved stakeholders.

Security Capabilities

SecFog needs a **vocabulary** to specify **security capabilities** available/required 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).

Security Capabilities

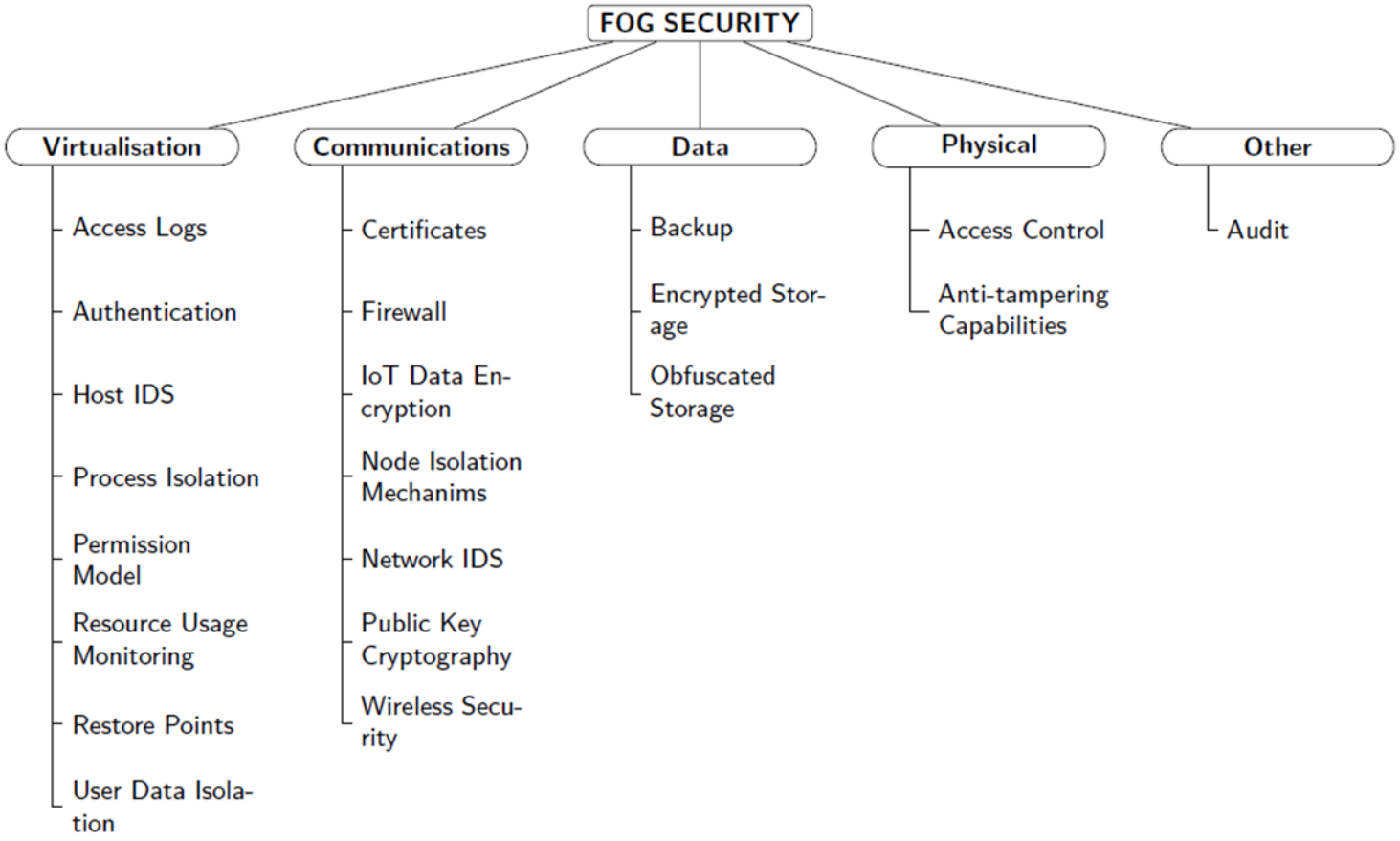
SecFog needs a **vocabulary** to specify **security capabilities** available/required in Fog scenarios.

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

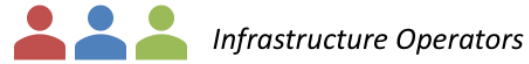
We hence proposed *our* taxonomy (based on recent surveys 📚)



Big Picture

How does SecFog work?

Big Picture



Infrastructure Operators

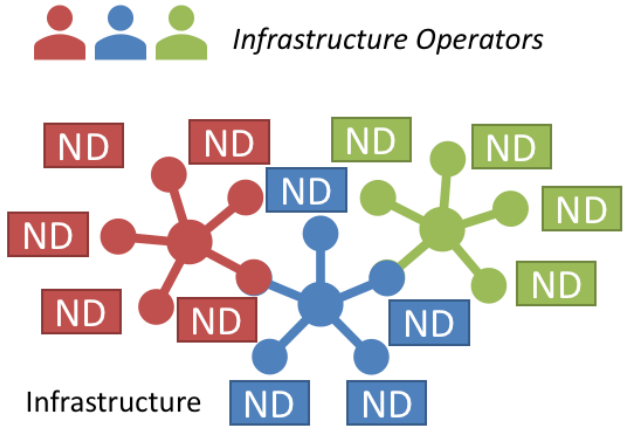
Infrastructure

SecFog

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

Indeed, Fog deployments will span various service providers.

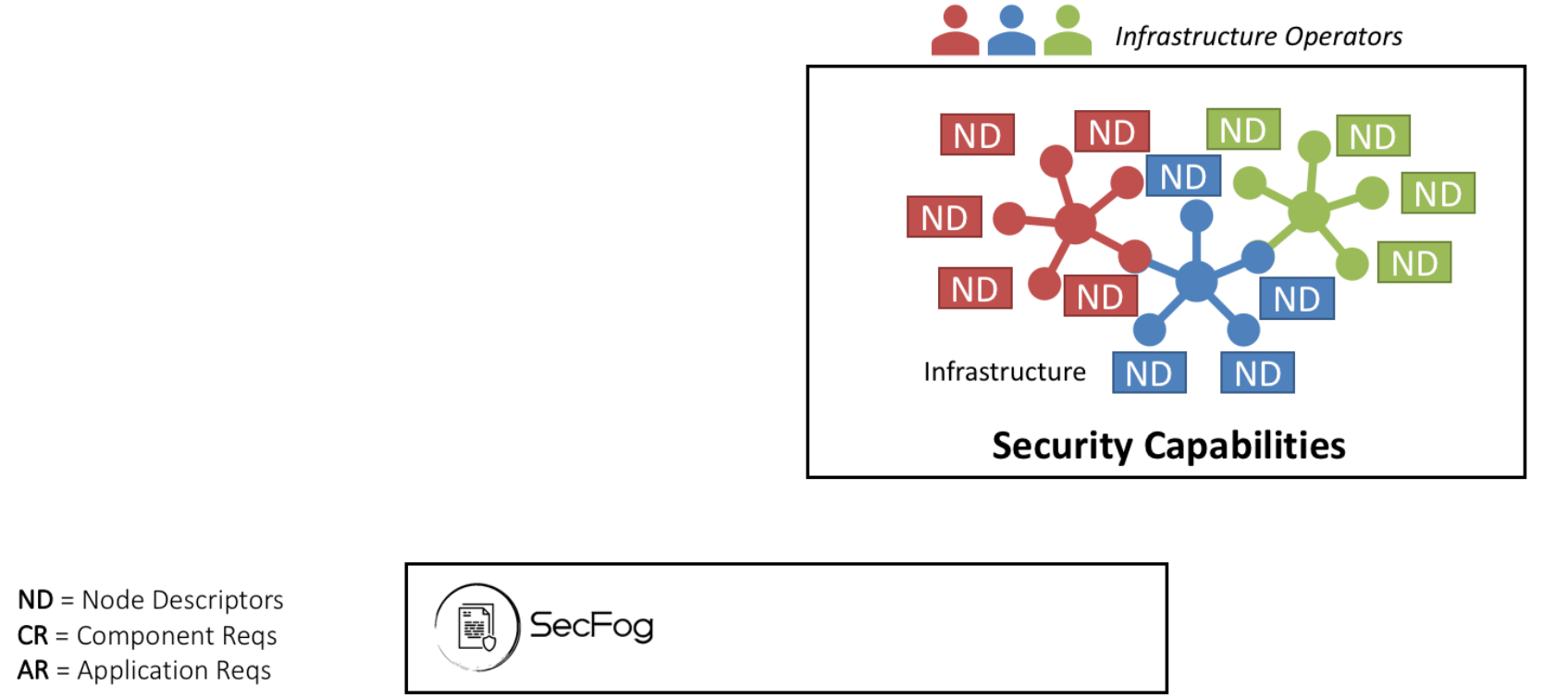
Big Picture



SecFog

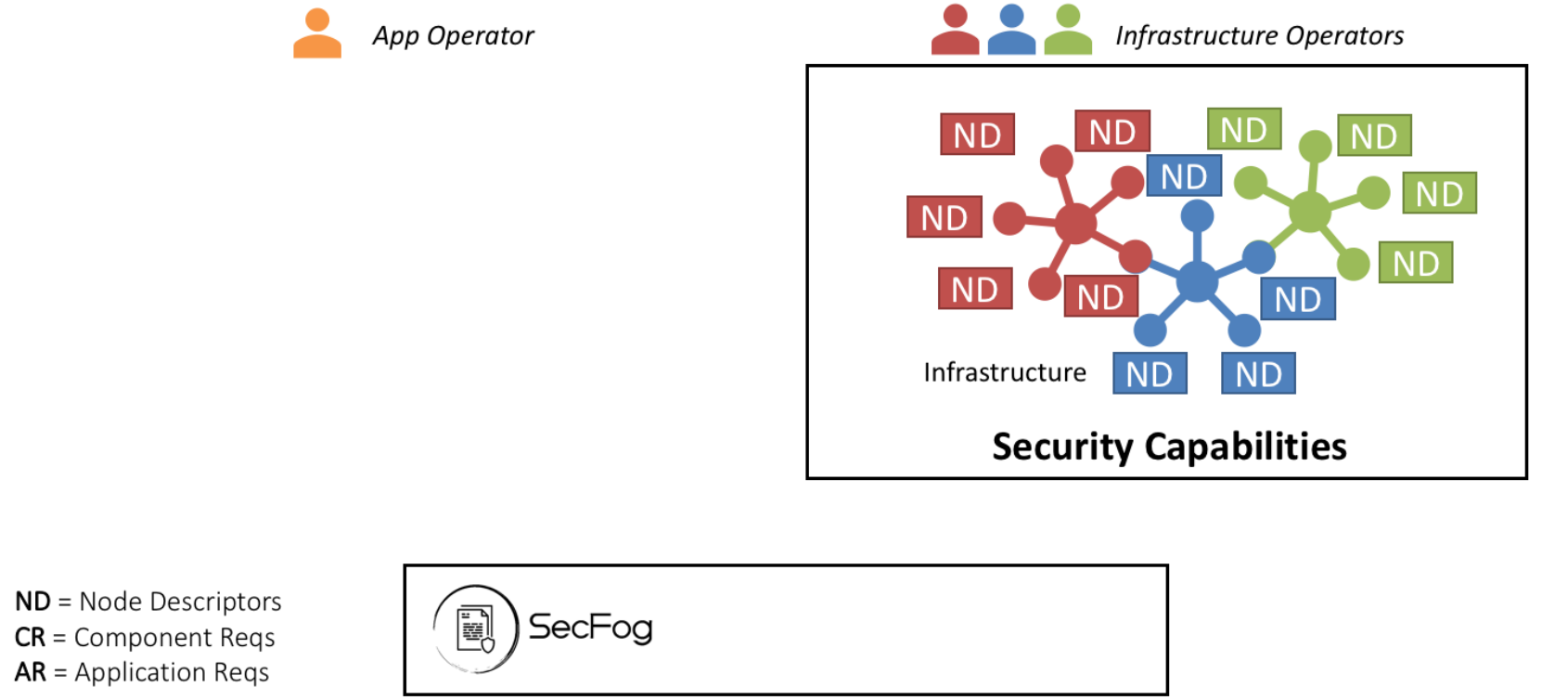
We assume that each node will self-describe its capabilities (and their *effectiveness against attacks*) through a **Node Descriptor** (ND) using the taxonomy vocabulary.

Big Picture



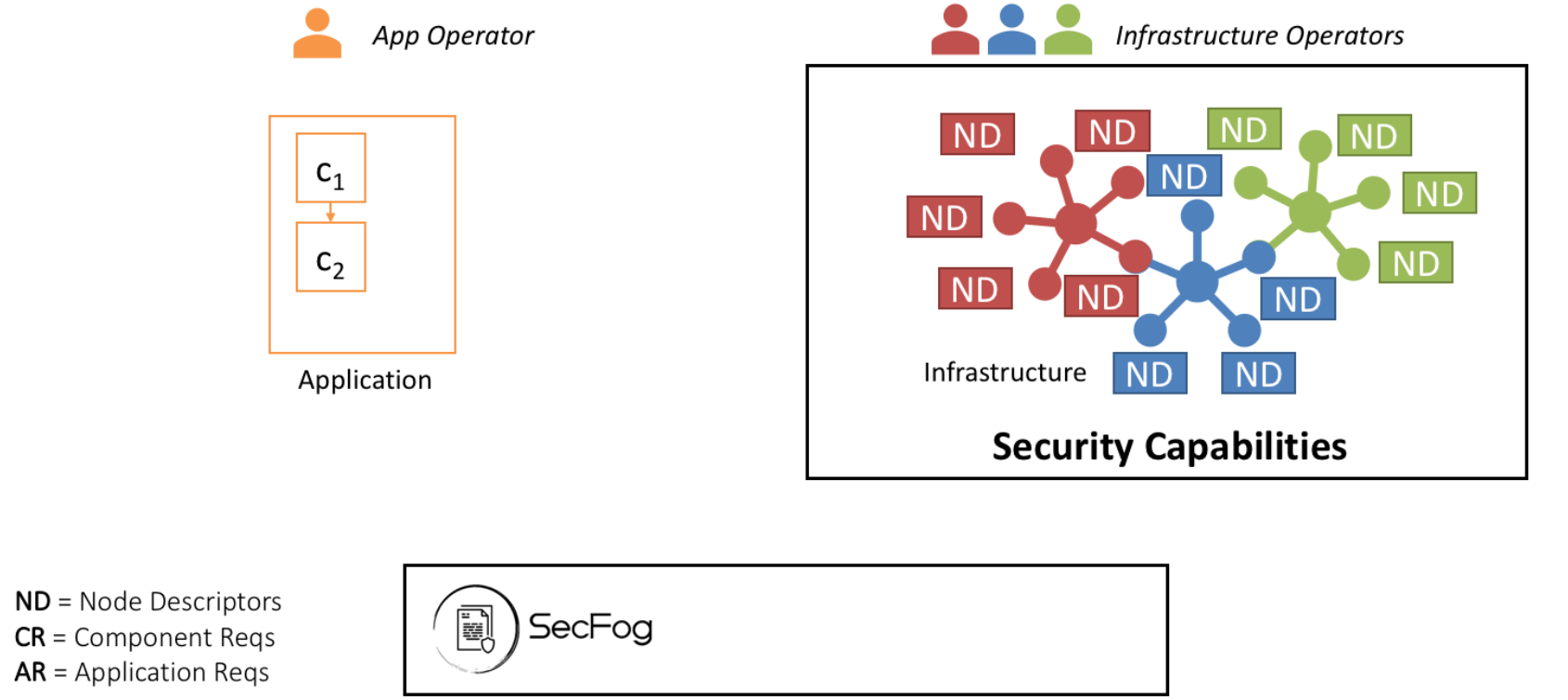
In this way, we get a complete description of the available **Security Capabilities**.

Big Picture



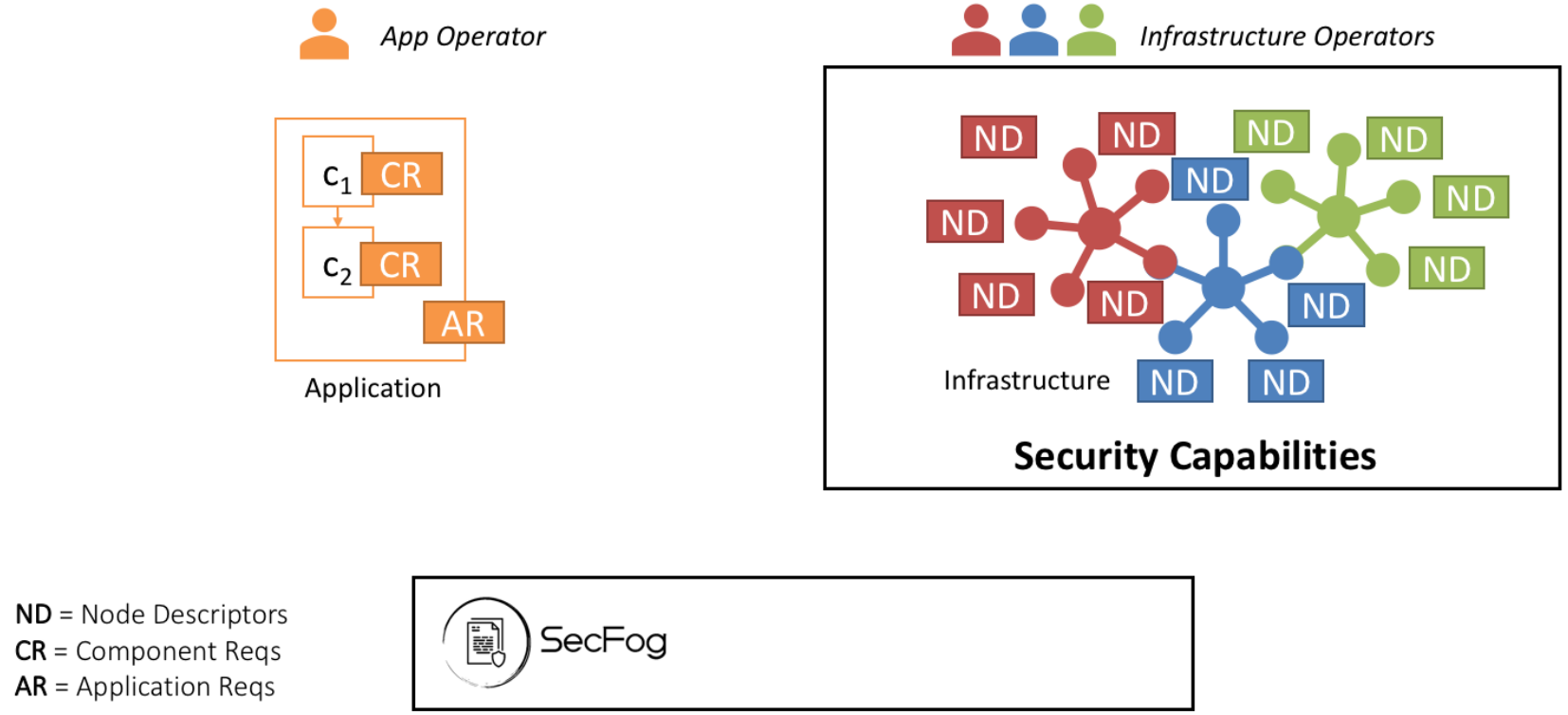
On the other hand, we consider the **App Operator** that will describe their app.

Big Picture



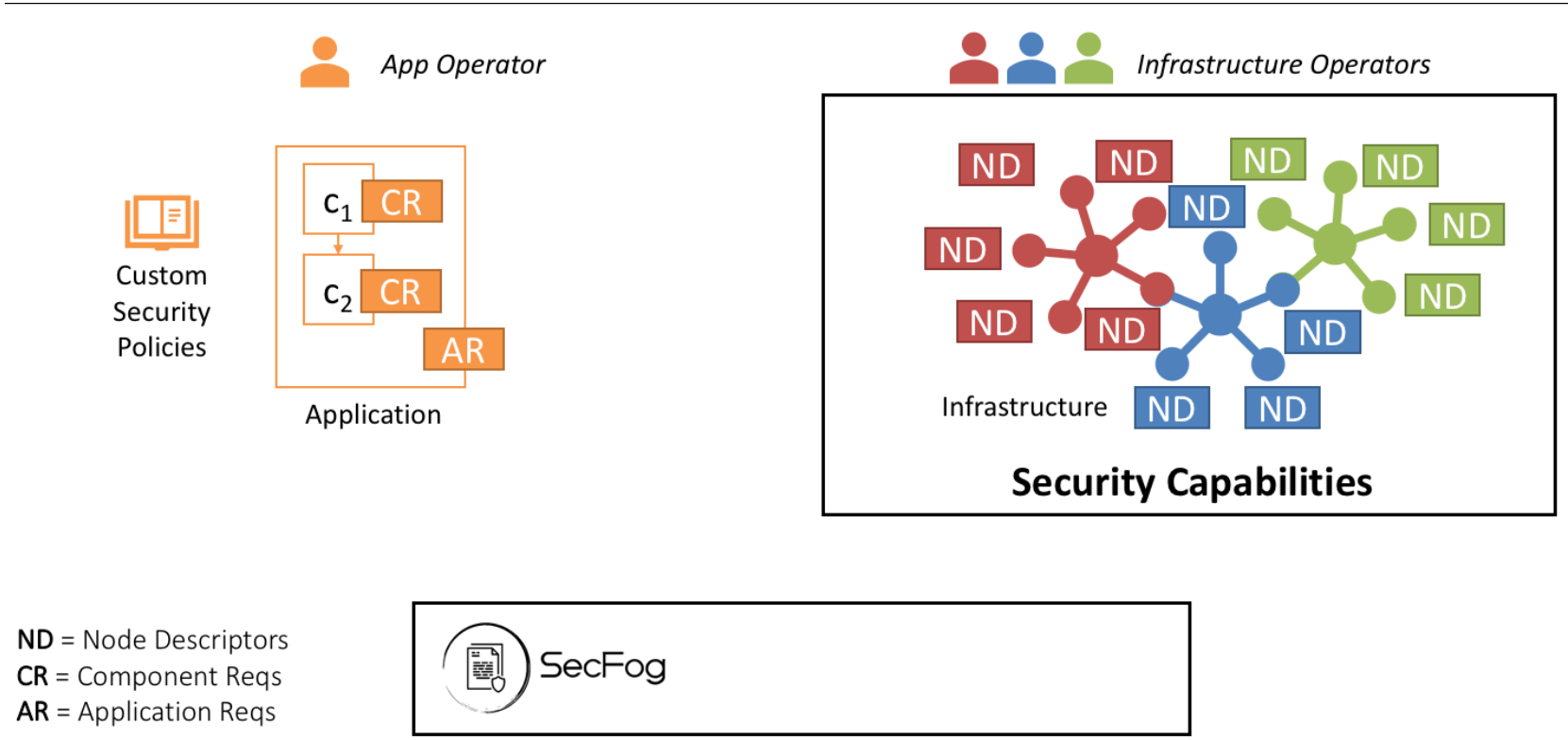
This can be done by specifying the **app topology**...

Big Picture



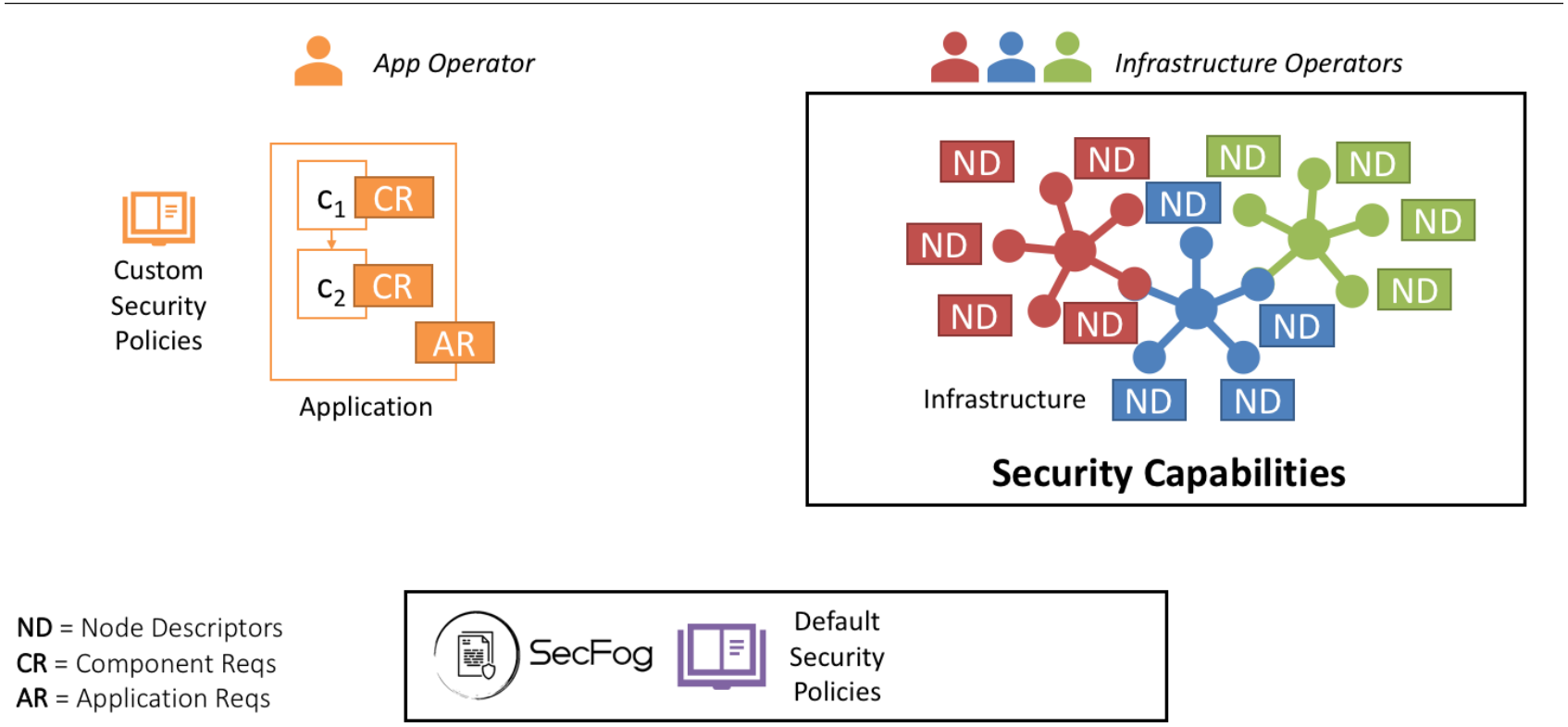
... and by annotating it with security requirements both at the **component** (CR) and **application** (AR) level, again relying on the taxonomy dictionary.

Big Picture



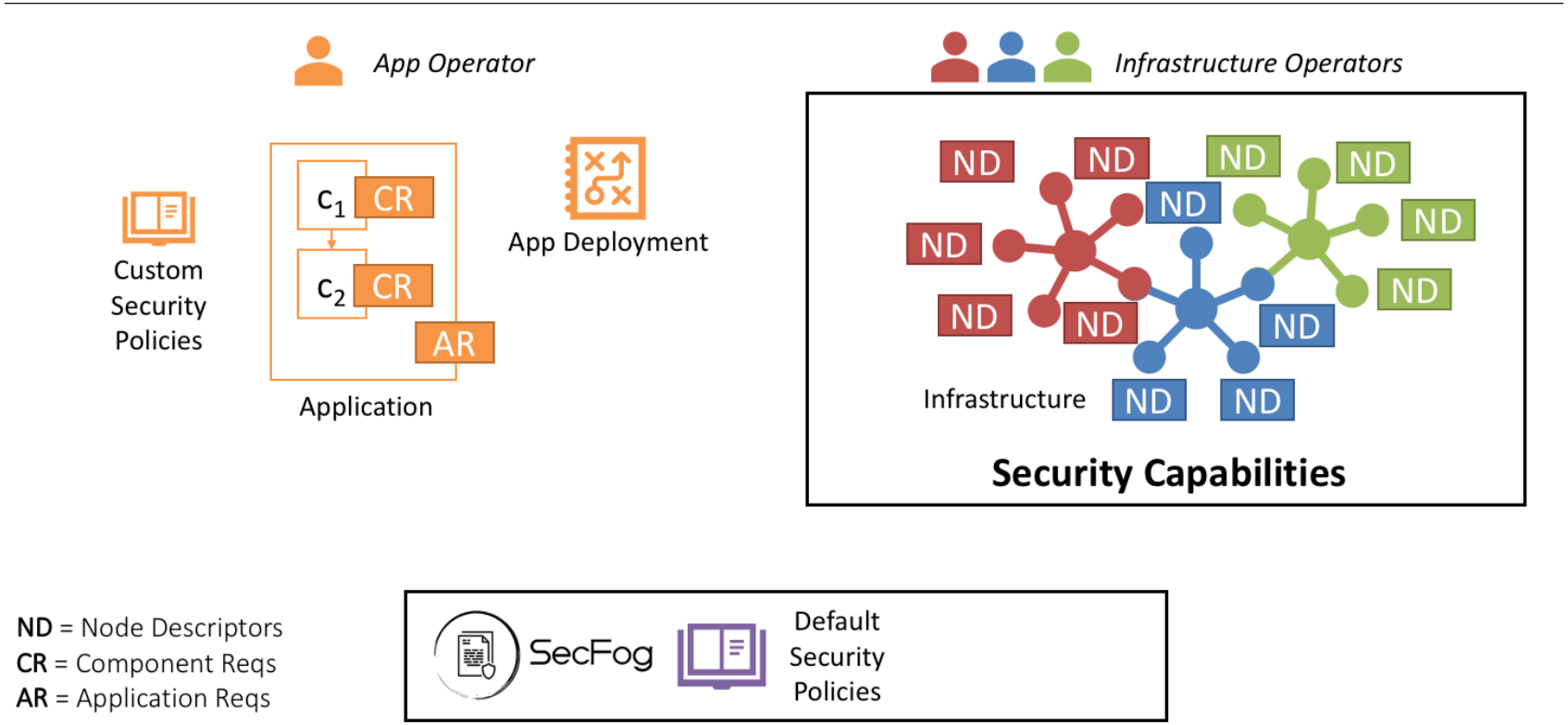
Secuirity requirements can be expressed in term of **Custom Security Policies**.

Big Picture



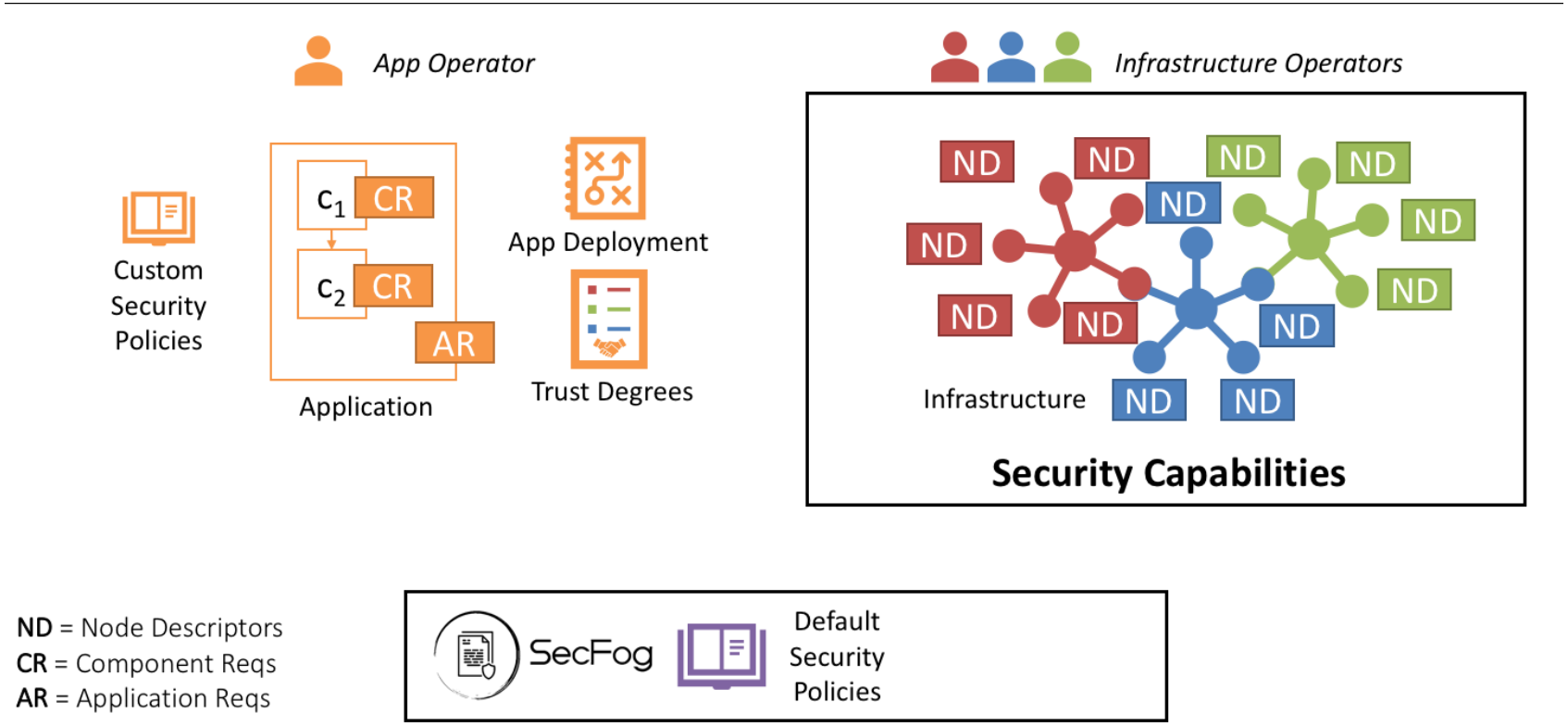
Custom Security Policies are expressed in terms of the **Default Security Policies** of SecFog.

Big Picture



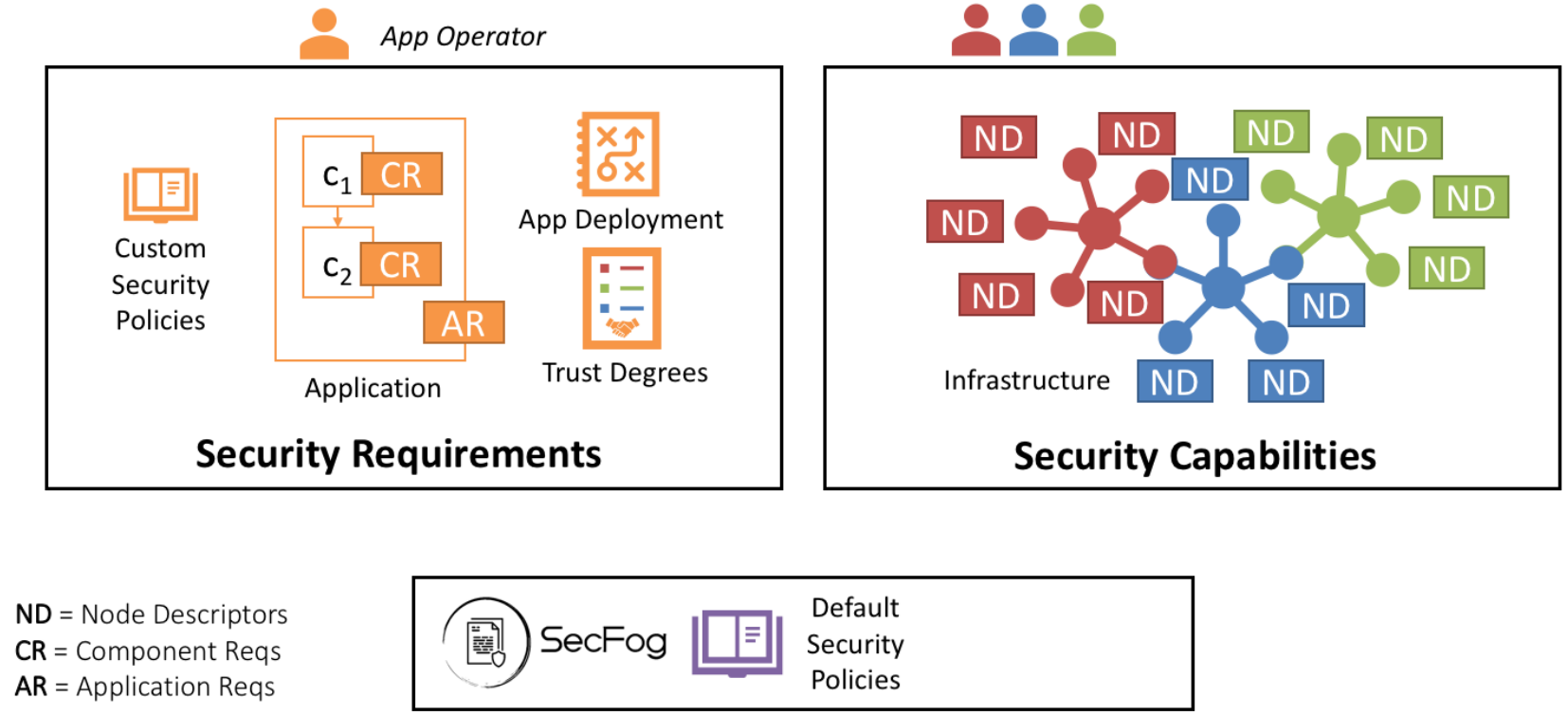
The App Operator can also specify complete or partial **App Deployments** of their application.

Big Picture



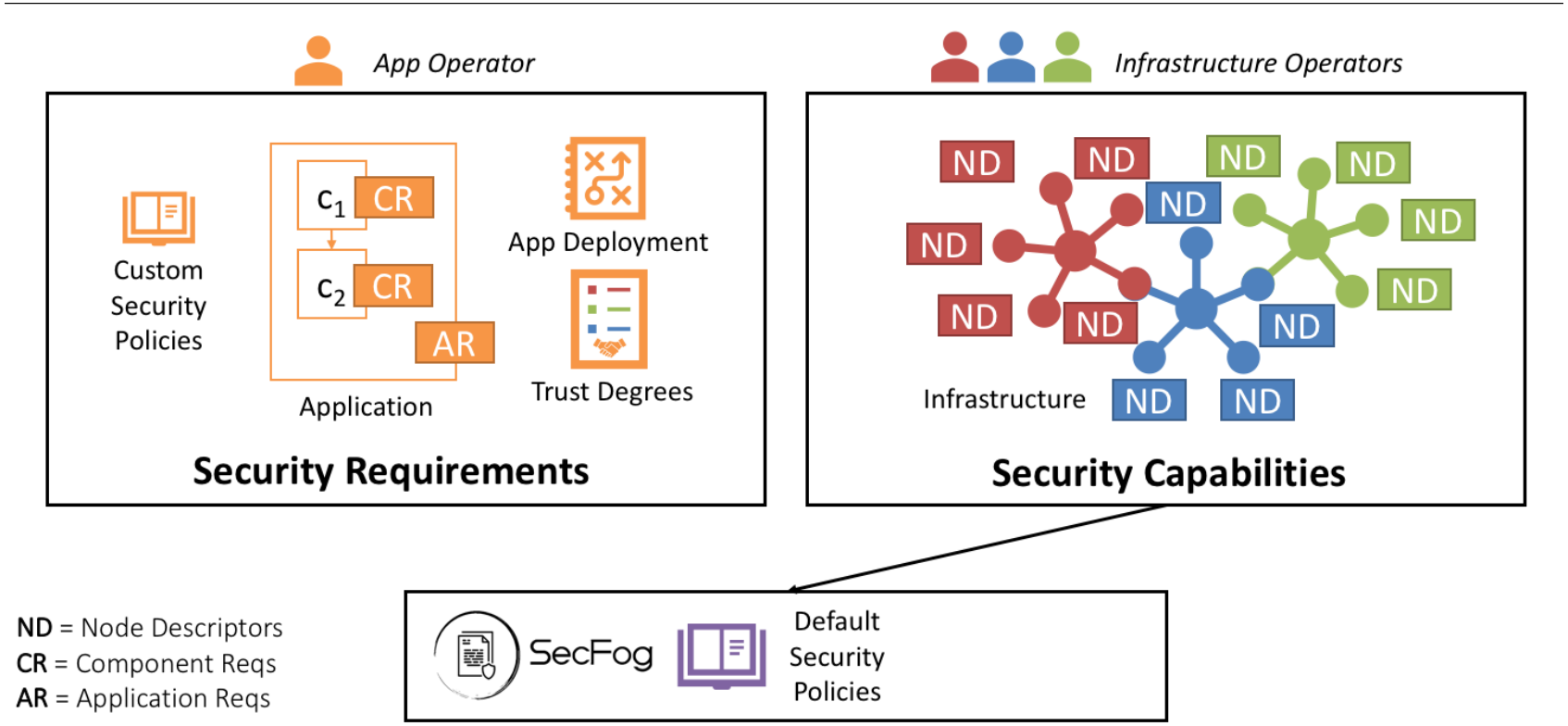
Finally, the App Operator can specify the **Trust Degrees** towards different Infrastructure Operators.

Big Picture



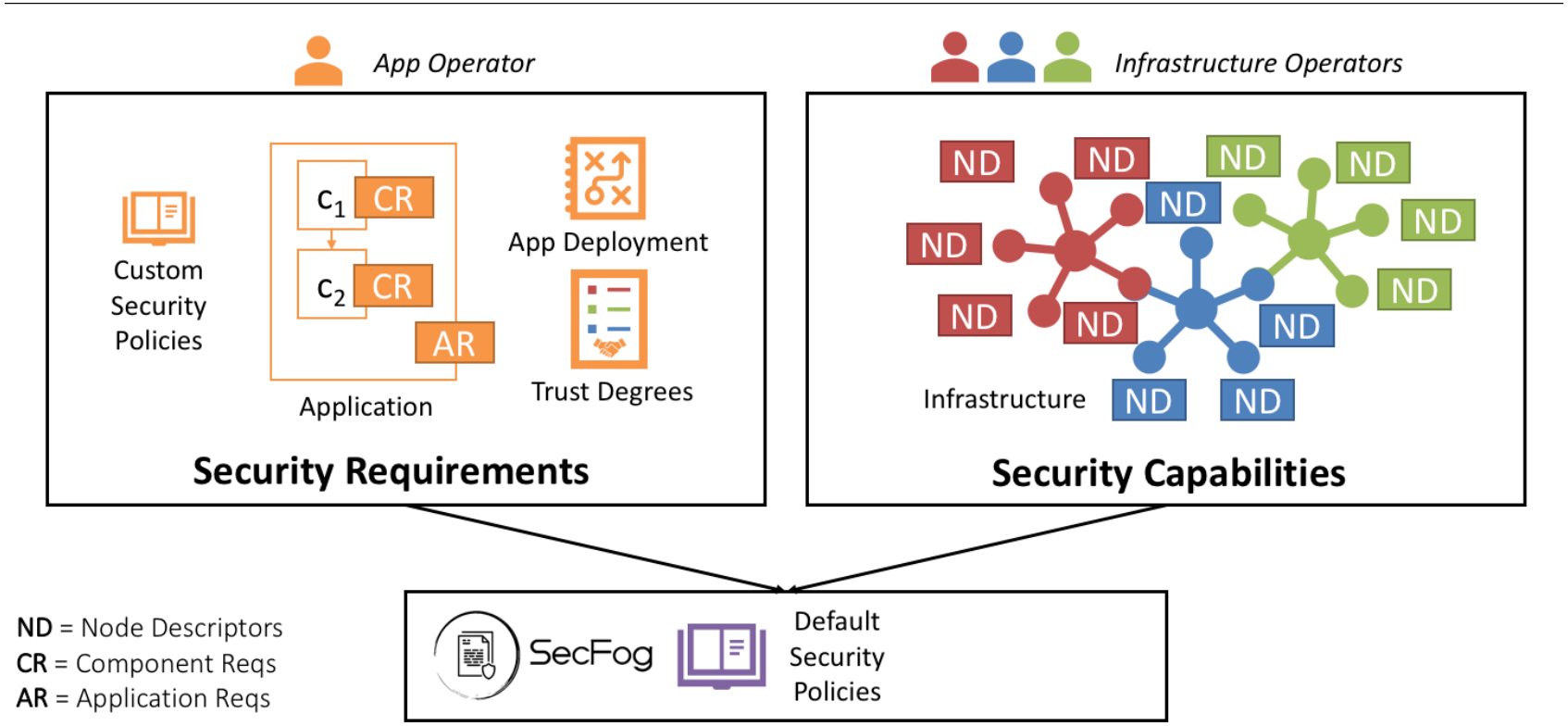
All this constitutes the **Security Requirements** of the considered multi-component app.

Big Picture



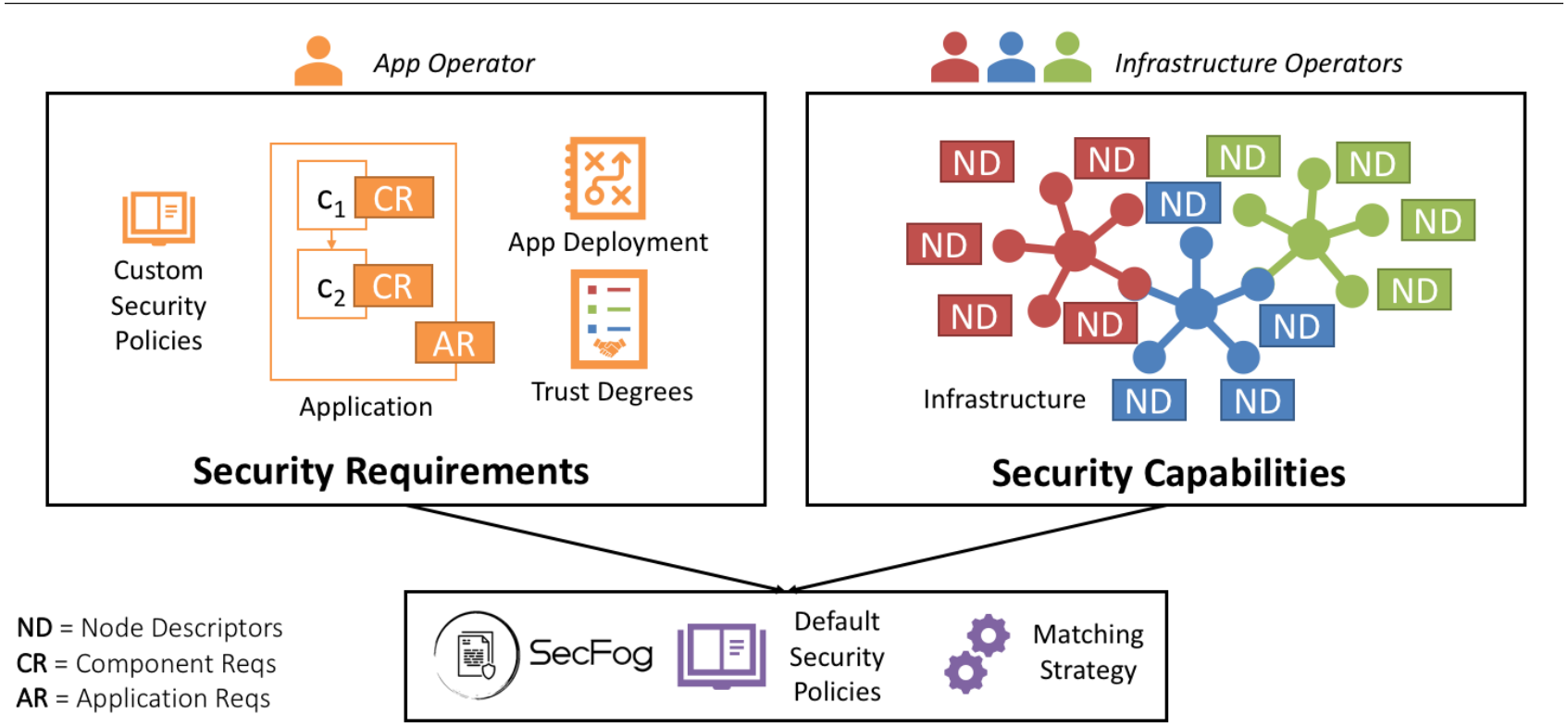
SecFog takes as input the Security Capabilities and…

Big Picture



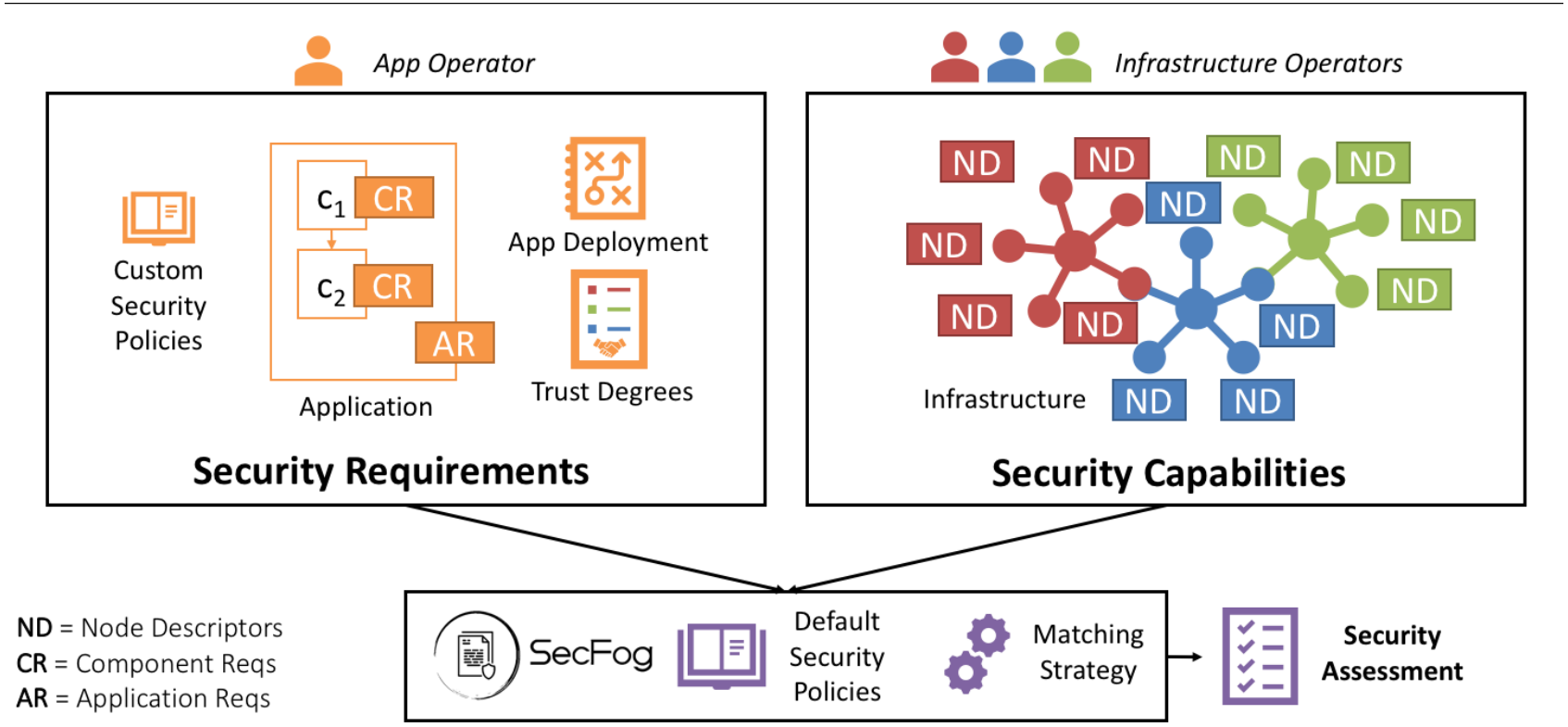
... the Security Requirements.

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.

Implementation

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

The prototype:

Implementation

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

The prototype:

1. **Matches** application security **requirements with** infrastructure **capabilities**

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**.

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**.

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* fifteen (15) lines of code 💻

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* fifteen (15) 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 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 101

ProbLog is a Python package that 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
```

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

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.

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`.

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.

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.

Reasoning on Possible Worlds: Intuition

A Problog program leverages input probability distributions to analyse all possible Prolog programs (i.e., worlds) that could be generated according to them.

Reasoning on Possible Worlds: Intuition

A Problog program leverages input probability distributions to analyse all possible Prolog programs (i.e., worlds) that could be generated according to them.

Assuming that $\Omega(q)$ is the set of possible worlds W that entail a valid proof for a certain query q (i.e., $\Omega(q) = \{W \mid W \models q\}$), the Problog engine computes the probability $p(q)$ that q holds as

Reasoning on Possible Worlds: Intuition

A Problog program leverages input probability distributions to analyse all possible Prolog programs (i.e., worlds) that could be generated according to them.

Assuming that $\Omega(q)$ is the set of possible worlds W that entail a valid proof for a certain query q (i.e., $\Omega(q) = \{W \mid W \models q\}$), the Problog engine computes the probability $p(q)$ that q holds as

$$p(q) = \sum_{W \in \Omega(q)} \prod_{f \in W} p(f)$$

where f are facts within a certain possible world, and $p(f)$ is the probability they are labelled with.

Reasoning on Possible Worlds: Intuition

A Problog program leverages input probability distributions to analyse all possible Prolog programs (i.e., worlds) that could be generated according to them.

Assuming that $\Omega(q)$ is the set of possible worlds W that entail a valid proof for a certain query q (i.e., $\Omega(q) = \{W \mid W \models q\}$), the Problog engine computes the probability $p(q)$ that q holds as

$$p(q) = \sum_{W \in \Omega(q)} \prod_{f \in W} p(f)$$

👓☕ **We are ready to implement SecFog!**

SecFog

The *Generate & Test* strategy of SecFog looks for a secure deployment `D` of application `A`, which is managed by app operator `OpA`.

SecFog

The *Generate & Test* strategy of SecFog looks for a secure deployment `D` of application `A`, which is managed by app operator `OpA`.

We assume application `A` is made from a list `L` of components.

SecFog

The *Generate & Test* strategy of SecFog looks for a secure deployment `D` of application `A`, which is managed by app operator `OpA`.

We assume application `A` is made from a list `L` of components. Then:

```
secFog(OpA, A, D) :-
    app(A, L),
    deployment(OpA, L, D).
```

SecFog - Deployments

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

SecFog - Deployments

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

```
deployment(_,[],[]).

deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    deployment(OpA,Cs,D).
```

SecFog - Deployments

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

```
deployment(_,[],[]).

deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    deployment(OpA,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 now extend the concept of secure deployment with a predicate that checks if a component `C` meets its security requirements when deployed to an existing node `N` (as per deployment `D`):

SecFog - Secure Deployment

We now extend the concept of secure deployment with a predicate that checks if a component `C` meets its security requirements when deployed to an existing node `N` (as per deployment `D`):

```
deployment(_,[],[]).
deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    securityRequirements(C,N),
    deployment(OpA,Cs,D).
```

SecFog - WeatherMonitor Example

Application, specified by appOp

```
app(weatherApp, [weatherMonitor]).
securityRequirements(weatherMonitor, N) :-
    (anti_tampering(N); access_control(N)),
    (wireless_security(N); iot_data_encryption(N)).
```

SecFog - WeatherMonitor Example

Application, specified by appOp

```
app(weatherApp, [weatherMonitor]).
securityRequirements(weatherMonitor, N) :-
    (anti_tampering(N); access_control(N)),
    (wireless_security(N); iot_data_encryption(N)).
```

Infrastructure, specified by cloudOp and edgeOp

```
node(cloud, cloudOp).
0.99::anti_tampering(cloud).
0.99::access_control(cloud).
0.99::iot_data_encryption(cloud).

node(edge, edgeOp).
0.8::anti_tampering(edge).
0.9::wireless_security(edge).
0.9::iot_data_encryption(edge).
```

WeatherMonitor Example

Application, specified by appOp

```
app(weatherApp, [weatherMonitor]).
securityRequirements(weatherMonitor, N) :-
    (anti_tampering(N); access_control(N)),
    (wireless_security(N); iot_data_encryption(N)).
```

Infrastructure, specified by cloudOp and edgeOp

```
node(cloud, cloudOp).
0.99::anti_tampering(cloud).
0.99::access_control(cloud).
0.99::iot_data_encryption(cloud).

node(edge, edgeOp).
0.8::anti_tampering(edge).
0.9::wireless_security(edge).
0.9::iot_data_encryption(edge).
```

```
query(secFog(appOp,weatherApp,D)).
```

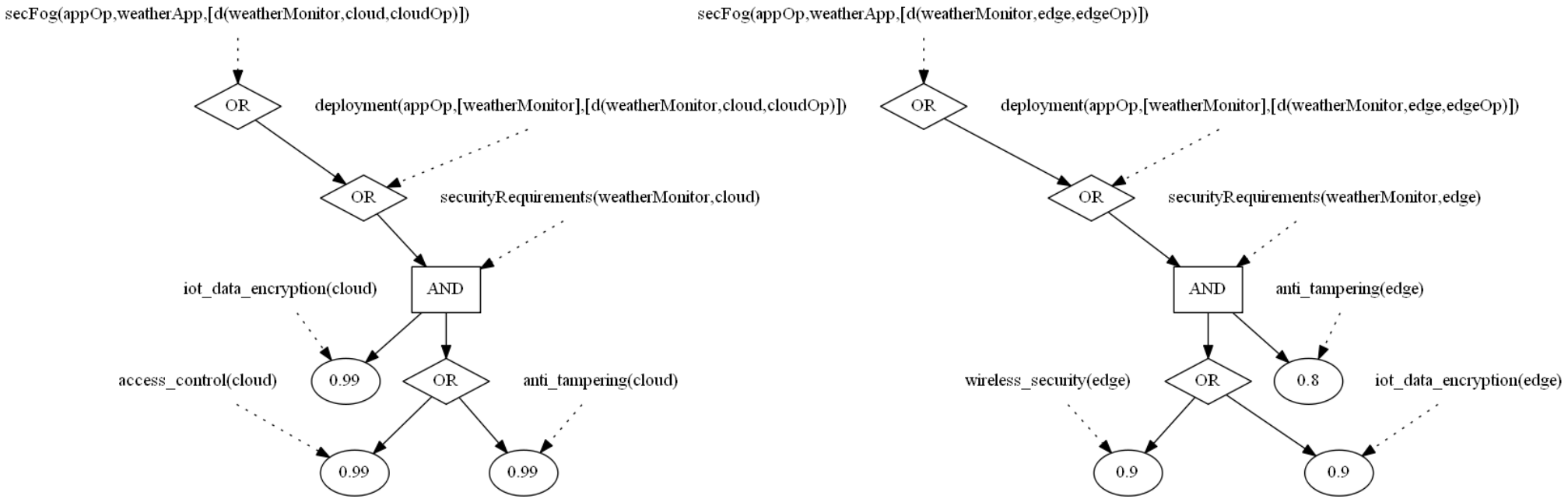
WeatherMonitor Results

Results of the query are as follows:

Query ▼	Location	Probability
secFog(appOp,weatherApp,[d(weatherMonitor,cloud,cloudOp)])	31:7	0.989901
secFog(appOp,weatherApp,[d(weatherMonitor,edge,edgeOp)])	31:7	0.792

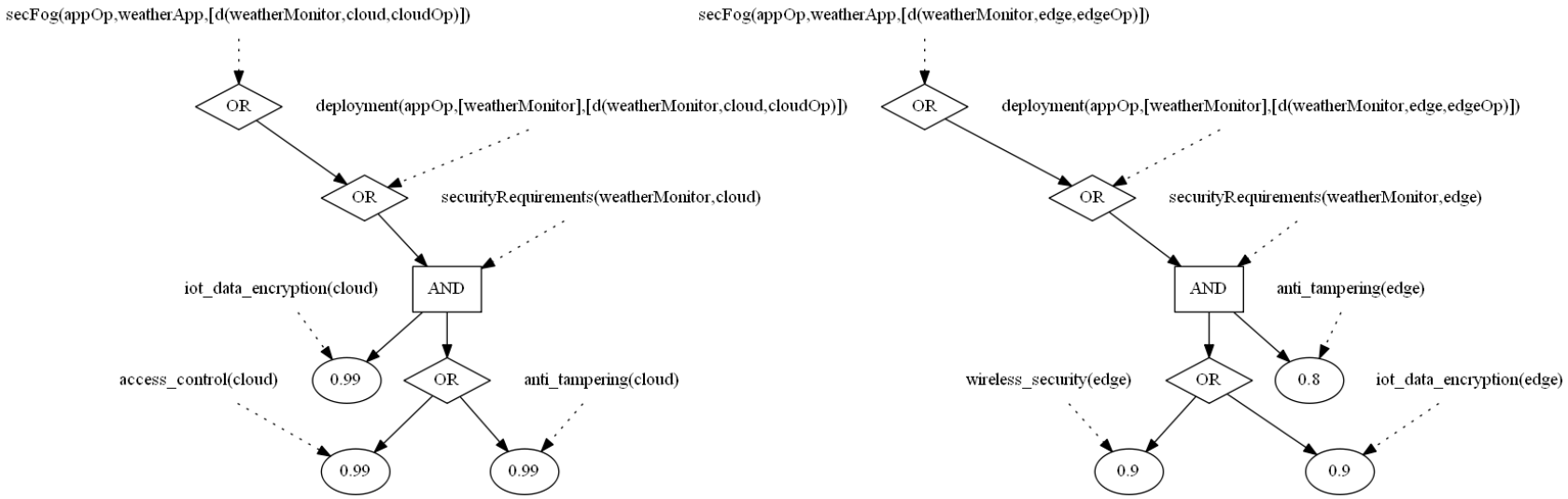
WeatherMonitor Results

By running SecFog in Problog `ground` mode, we can explain those results via the AND-OR trees of the two ground programs that led to the output.



WeatherMonitor Results

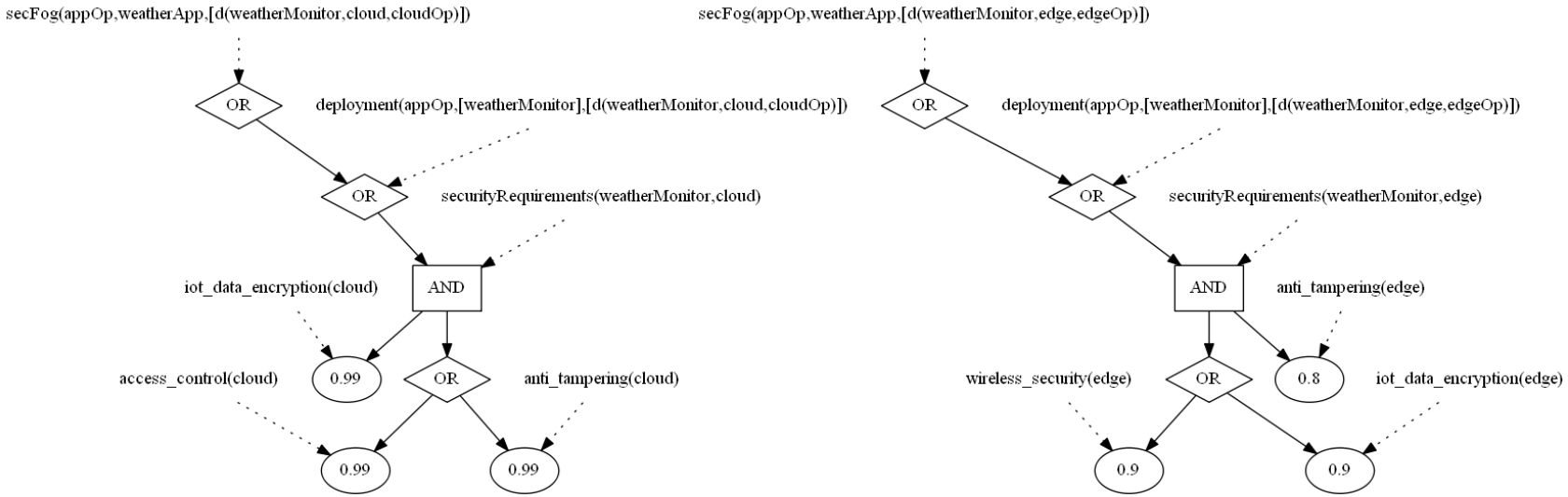
By running SecFog in Problog `ground` mode, we can explain those results via the AND-OR trees of the two ground programs that led to the output.



For instance, the Cloud deployment (left handside tree) is obtained as:

WeatherMonitor Results

By running SecFog in Problog `ground` mode, we can explain those results via the AND-OR trees of the two ground programs that led to the output.

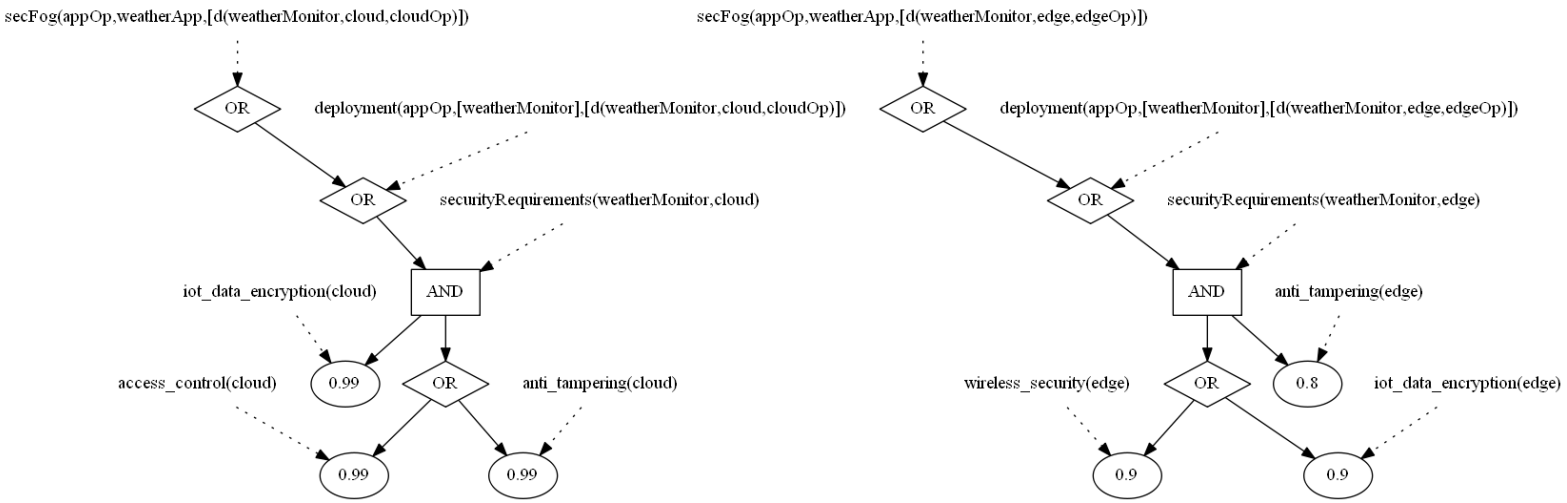


For instance, the Cloud deployment (left handside tree) is obtained as:

$$\mathrm{p}(\texttt{anti\_tampering(cloud)}) \times p(\texttt{iot\_data\_encryption(cloud)}) +$$
$$(1 - p(\texttt{anti\_tampering(cloud)})) \times \ p(\texttt{access\_control(cloud)}) \times \ p(\texttt{iot\_data\_encryption(cloud)}) =$$
$$.99 \times .99 + (1 - .99) \times .99 \times .99 =$$
$$0.989901$$

WeatherMonitor Results

By running SecFog in Problog `ground` mode, we can explain those results via the AND-OR trees of the two ground programs that led to the output.



For instance, the Cloud deployment (left handside tree) is obtained as:

$$\begin{aligned}&\mathtt{p(anti\_tampering(cloud))} \times p(\mathtt{iot\_data\_encryption(cloud)}) + \\ &(1 - p(\mathtt{anti\_tampering(cloud)})) \times p(\mathtt{access\_control(cloud)}) \times p(\mathtt{iot\_data\_encryption(cloud)}) = \\ &.99 \times .99 + (1 - .99) \times .99 \times .99 = \\ &0.989901\end{aligned}$$

This kind of proofs can be obtained by using the `explain` mode of Problog.

SecFog - Default Trust Model

We now extend SecFog to check that the infrastructure operator `OpN` managing `N` -- to which component `C` is deployed -- can be trusted according to the application operator `OpA`.

```
deployment(_,[],[]).
deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    securityRequirements(C,N),
    trusts2(OpA, OpN), % line to be added
    deployment(OpA,Cs,D).
```

SecFog - Default Trust Model

We now extend SecFog to check that the infrastructure operator `OpN` managing `N` -- to which component `C` is deployed -- can be trusted according to the application operator `OpA`.

```
deployment(_,[],[]).
deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    securityRequirements(C,N),
    trusts2(OpA, OpN),
    deployment(OpA,Cs,D).
```

We then include our default trust model:

```
trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

SecFog - Default Trust Model

```
trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

It is worth noting that the default trust model considers direct trust relations (i.e., opinions) from different stakeholders (i.e., `trusts(A,B)`) and combines them so to complete the (possibly partial) trust network.

SecFog - Default Trust Model

```
trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

It is worth noting that the default trust model considers direct trust relations (i.e., opinions) from different stakeholders (i.e., `trusts(A,B)`) and combines them so to complete the (possibly partial) trust network.

Particularly:

SecFog - Default Trust Model

```
trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

It is worth noting that the default trust model considers direct trust relations (i.e., opinions) from different stakeholders (i.e., `trusts(A,B)`) and combines them so to complete the (possibly partial) trust network.

Particularly:

- opinions **along paths** are combined via **multiplication**, and

SecFog - Default Trust Model

```
trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

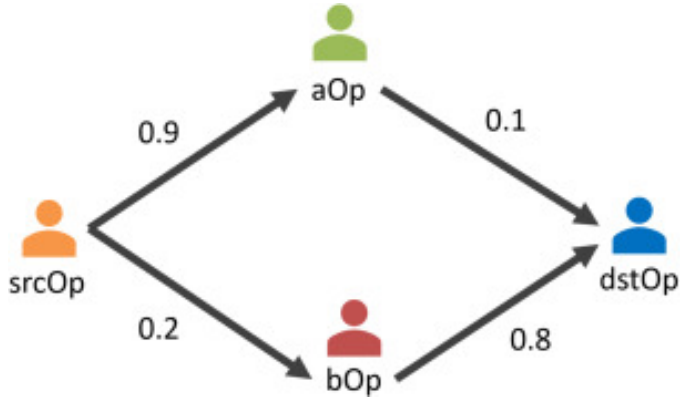
It is worth noting that the default trust model considers direct trust relations (i.e., opinions) from different stakeholders (i.e., `trusts(A,B)`) and combines them so to complete the (possibly partial) trust network.

Particularly:

- opinions **along paths** are combined via **multiplication**, and
- opinions **across paths** are combined via **addition**.

Trust Network Example

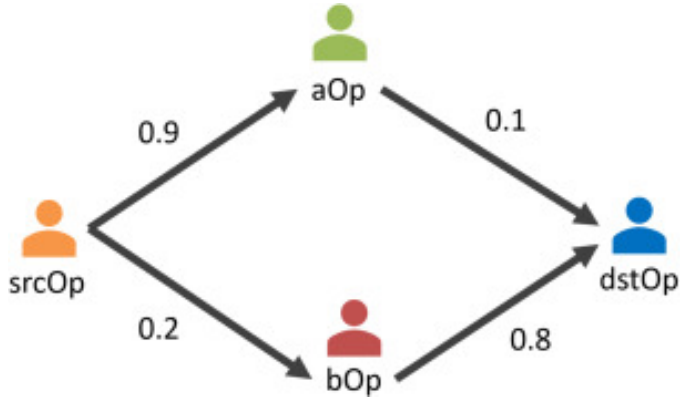
Try to complete the declaration of the following trust network and run the proposed query:




```
%%% trust relations declared by srcOp
0.9::trusts(srcOp, aOp).

%%% trust relations declared by aOp

%%% trust relations declared by bOp


query(trusts2(srcOp, dstOp)).
```

Trust Network Example

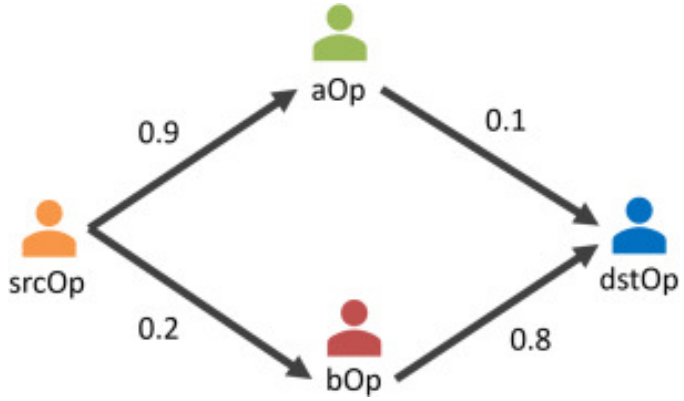
Try to complete the declaration of the following trust network and run the proposed query:




```
%%% trust relations declared by srcOp
0.9::trusts(srcOp, aOp).
0.2::trusts(srcOp, bOp).
%%% trust relations declared by aOp
0.1::trusts(aOp, dstOp).
%%% trust relations declared by bOp
0.8::trusts(bOp, dstOp).

query(trusts2(srcOp, dstOp)).
```

Trust Network Results

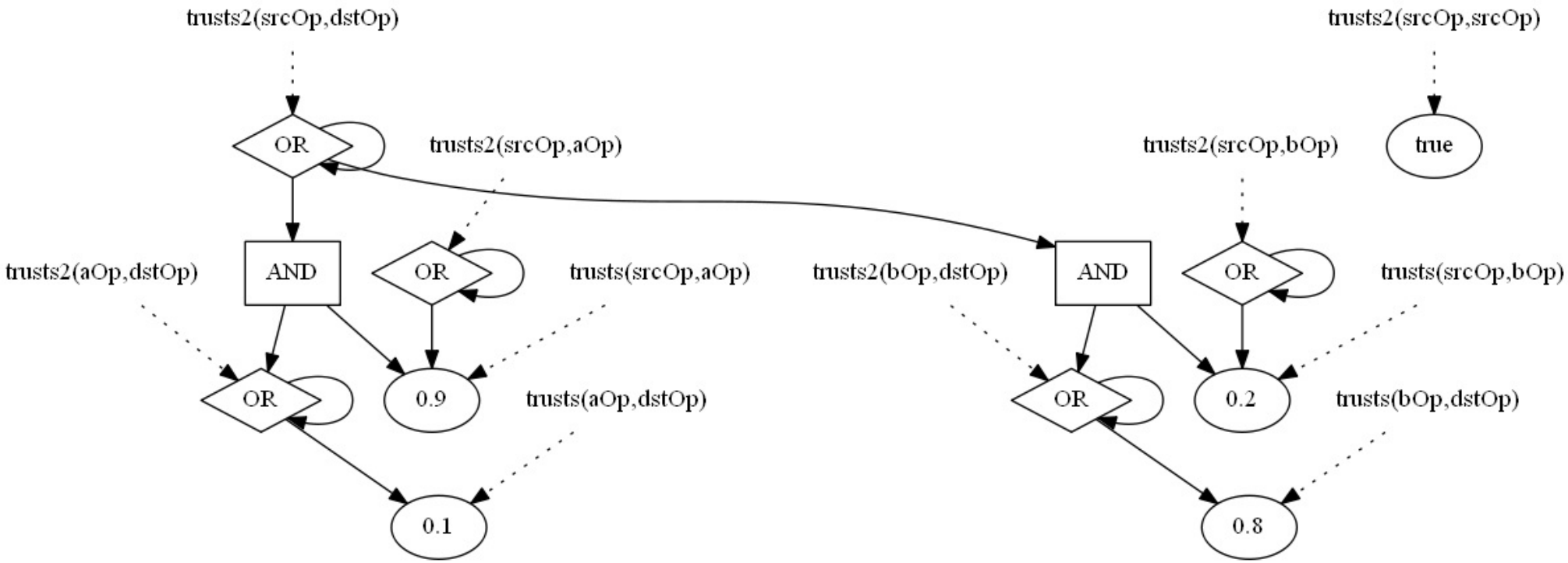
Try to complete the declaration of the following trust network and run the proposed query:




```
%%% trust relations declared by srcOp
0.9::trusts(srcOp, aOp).
0.2::trusts(srcOp, bOp).
%%% trust relations declared by aOp
0.1::trusts(aOp, dstOp).
%%% trust relations declared by bOp
0.8::trusts(bOp, dstOp).

query(trusts2(srcOp, dstOp)). % Result: 0.2356
```

Trust Network Results

Also in this case it is possible to get the AND-OR tree explaining the obtained result:



SecFog

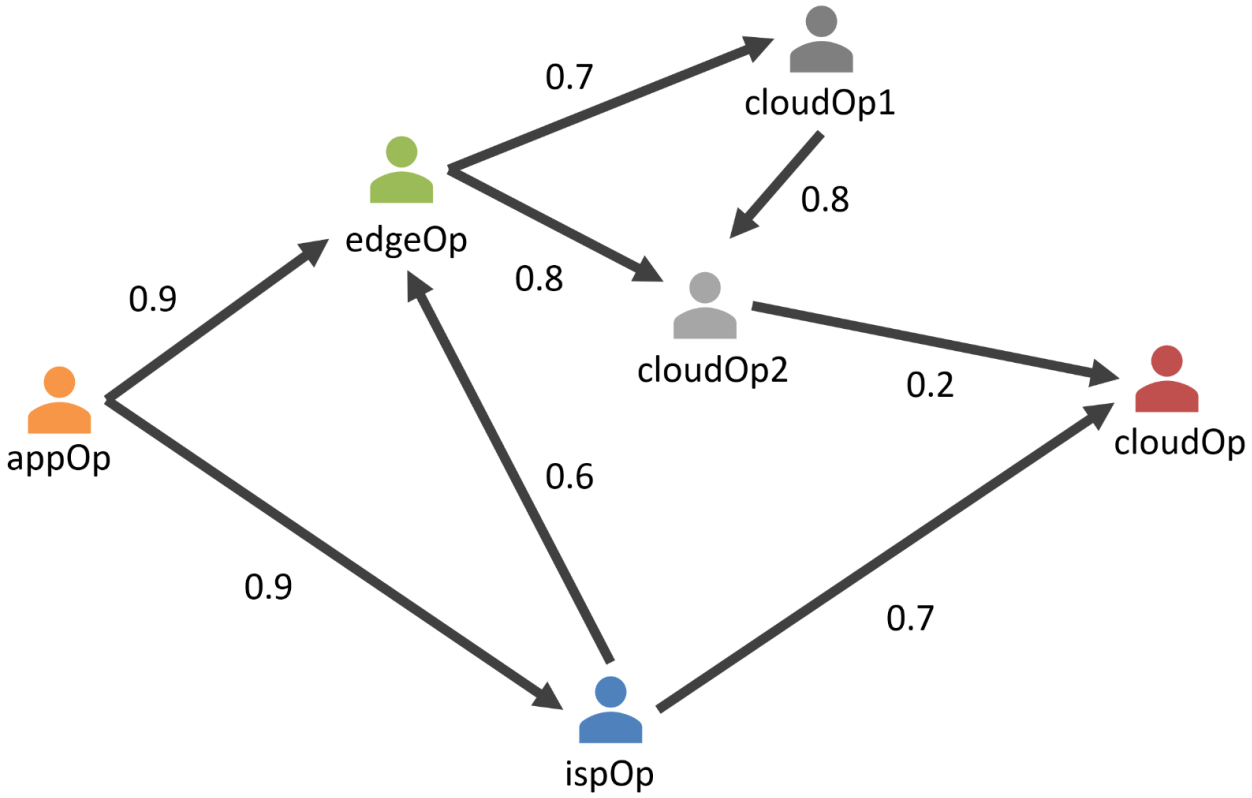
```
secFog(OpA, A, D) :-
    app(A, L),
    deployment(OpA, L, D).

deployment(_,[],[]).
deployment(OpA,[C|Cs],[d(C,N,OpN)|D]) :-
    node(N,OpN),
    securityRequirements(C,N),
    trusts2(OpA, OpN),
    deployment(OpA,Cs,D).

trusts(X,X).
trusts2(A,B) :-
    trusts(A,B).
trusts2(A,B) :-
    trusts(A,C),
    trusts2(C,B).
```

📦 That's all folks! These are the default security policies of SecFog.

As promised, they are only 15 lines of code 😏

Weather Monitor Example (Cont'd)

Let's go through a **complete example**, retaking the Weather Monitor we have seen before, and including this trust network:



Weather Monitor Example (Cont'd)

Let's go through a **complete example**, retaking the Weather Monitor we have seen before, and including this trust network:

```
%%% trust relations declared by appOp
.9::trusts(appOp, edgeOp).
.9::trusts(appOp, ispOp).

%%% trust relations declared by edgeOp
.7::trusts(edgeOp, cloudOp1).
.8::trusts(edgeOp, cloudOp2).

%%% trust relation declared by cloudOp1
.8::trusts(cloudOp1, cloudOp2).

%%% trust relation declared by cloudOp2
.2::trusts(cloudOp2, cloudOp).

%%% trust relations declared by ispOp
.8::trusts(ispOp, cloudOp).
.6::trusts(ispOp, edgeOp).
```

Weather Monitor (Cont'd) Results

Query ▼	Location	Probability
secFog(appOp,weatherApp,[d(weatherMonitor,cloud,cloudOp)])	58:7	0.76017935
secFog(appOp,weatherApp,[d(weatherMonitor,edge,edgeOp)])	58:7	0.755568

Weather Monitor (Cont'd) Results

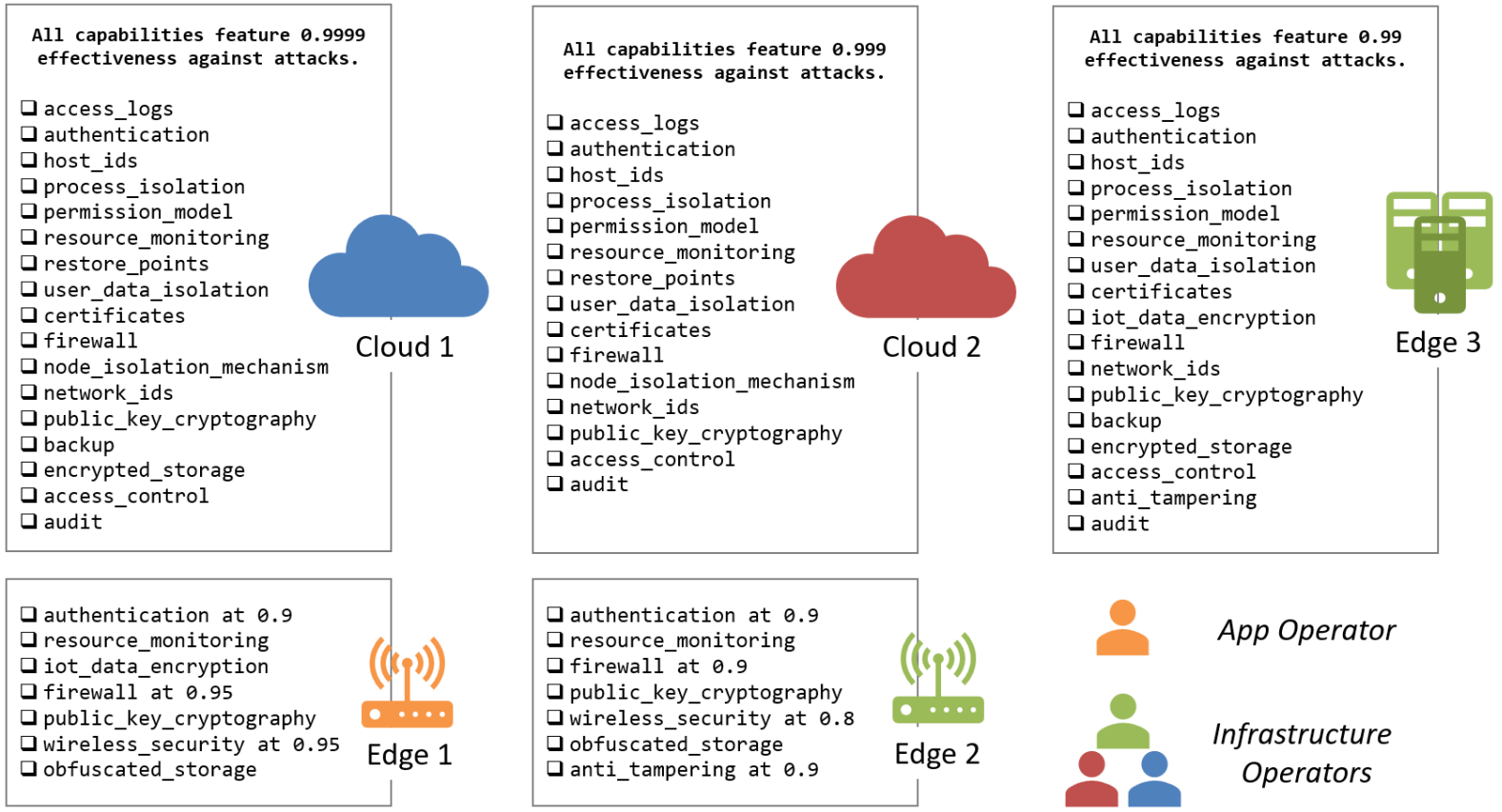
Query ▼	Location	Probability
secFog(appOp,weatherApp,[d(weatherMonitor,cloud,cloudOp)])	58:7	0.76017935
secFog(appOp,weatherApp,[d(weatherMonitor,edge,edgeOp)])	58:7	0.755568

It is worth noting that trust considerations substantially reduce the security level of the Cloud deployment, with respect to the previous assessment based only on the declared effectiveness of security countermeasures:

Query ▼	Location	Probability
secFog(appOp,weatherApp,[d(weatherMonitor,cloud,cloudOp)])	31:7	0.989901
secFog(appOp,weatherApp,[d(weatherMonitor,edge,edgeOp)])	31:7	0.792

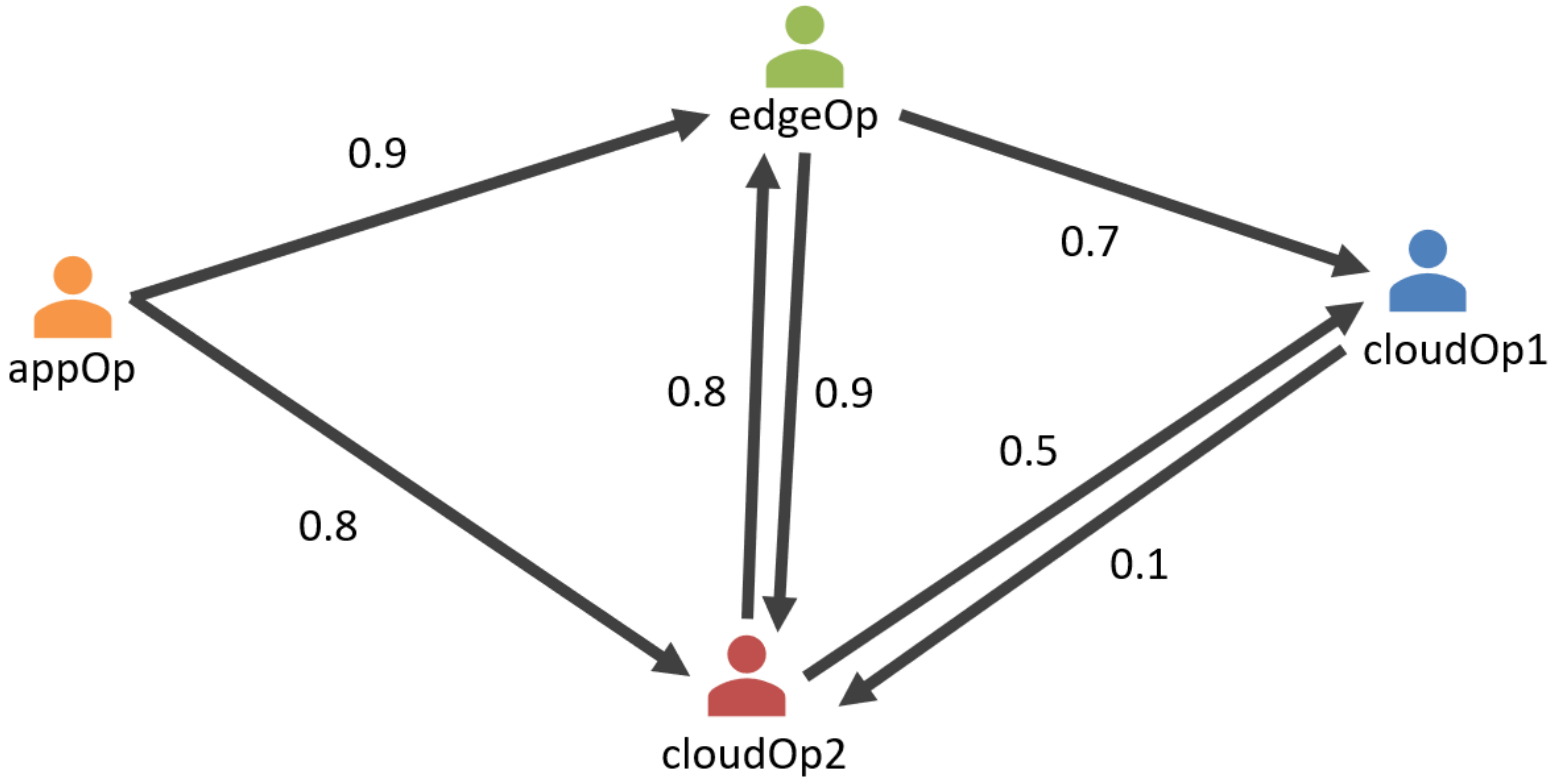
Example: Node Descriptors

The file `infrastructure.pl` programmatically declares the Node Descriptors for the infrastructure sketched below.



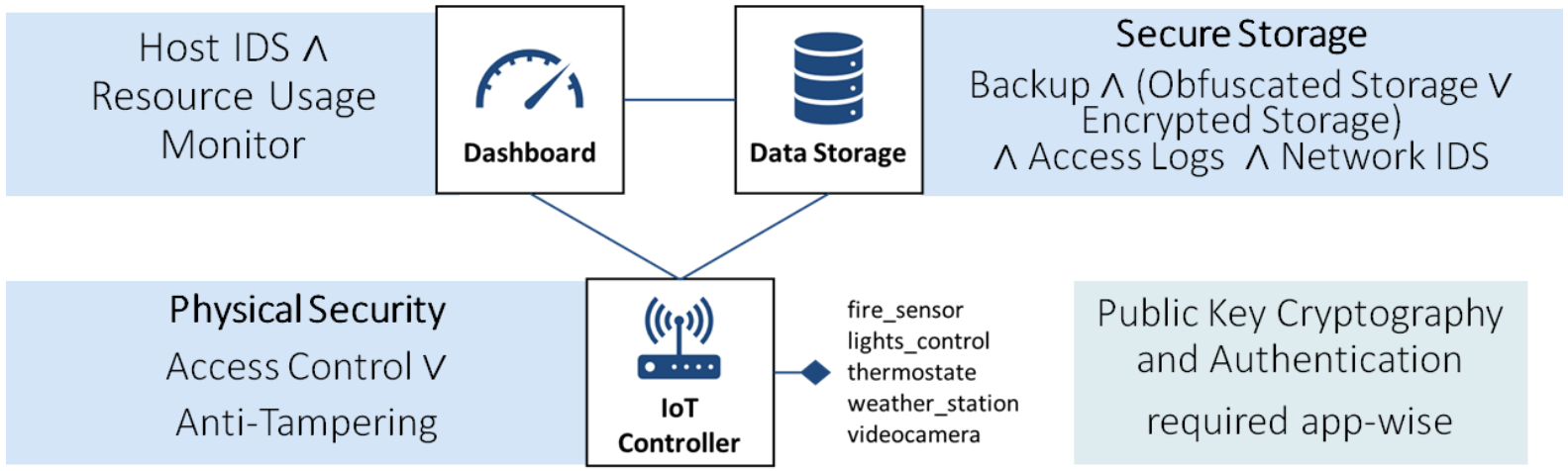
Example: Trust Degrees

The file `infrastructure.pl` programmatically declares the Trust Network sketched below.



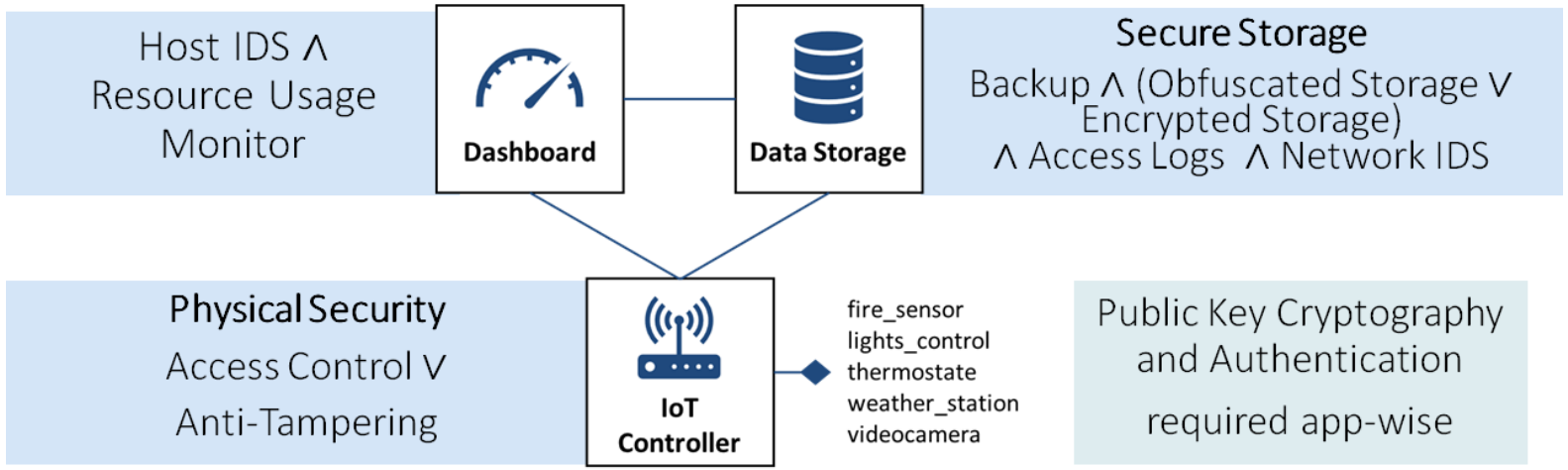
Example: Application

The application



Example: Application

The application

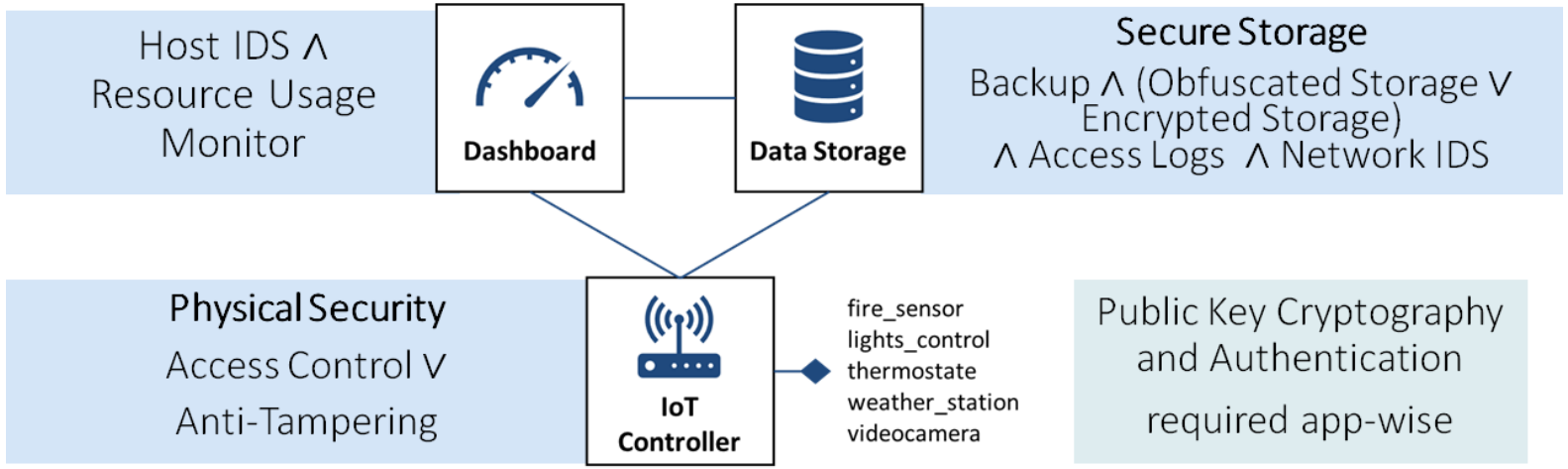


can be specified as:

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

Example: Application

The application



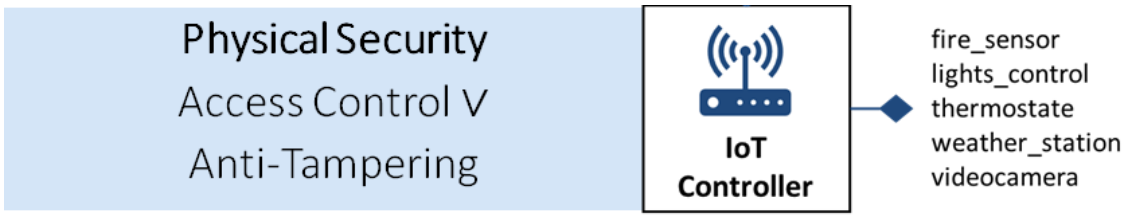
can be specified as:

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

What about its **security requirements**?

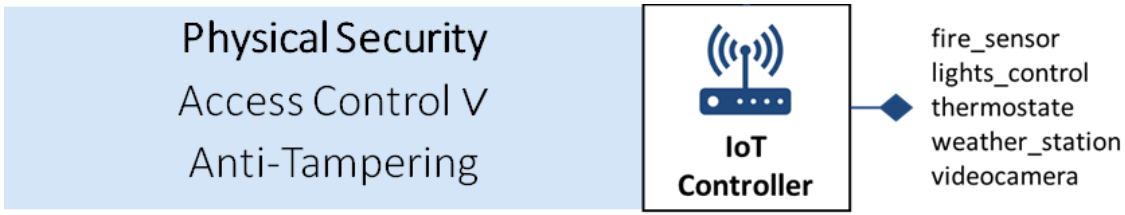
Example: Component Security Requirements

The requirements of the IoT Controller component



Example: Component Security Requirements

The requirements of the IoT Controller component

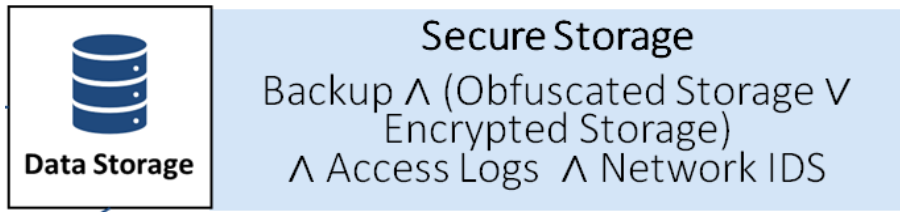


can be specified as:

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

securityRequirements(iot_controller, N) :-
    physical_security(N),
    public_key_cryptography(N),
    authentication(N).
```

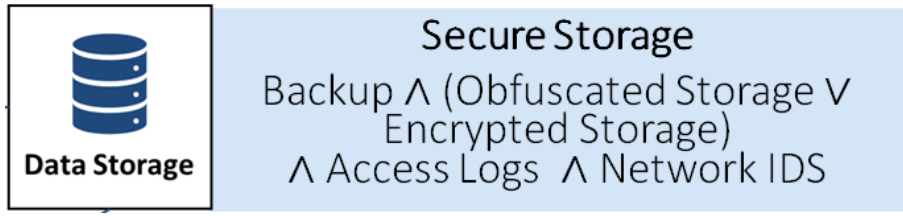
Example: Component Security Requirements

The requirements of the DataStorage component



Example: Component Security Requirements

The requirements of the DataStorage component

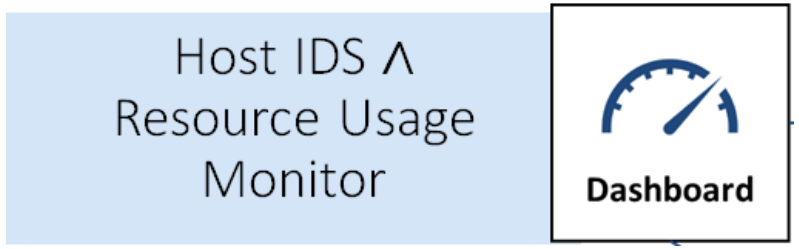


can be specified as:

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

securityRequirements(data_storage, N) :-
    secure_storage(N),
    access_logs(N),
    network_ids(N),
    public_key_cryptography(N),
    authentication(N).
```

❓ Pop Quiz ❓

Can you write component security requirements for the Dashboard?

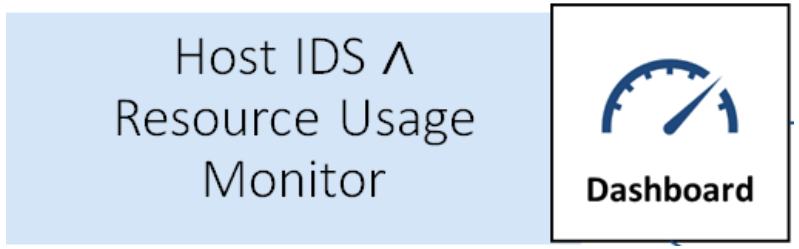


Any volunteers?

```
securityRequirements(dashboard, N) :-
```

❓ Pop Quiz ❓

Can you write component security requirements for the Dashboard?



Any volunteers?

```
securityRequirements(dashboard, N) :-
    host_ids(N),
    resource_monitoring(N),
    public_key_cryptography(N),
    authentication(N).
```

Example: Query

We now run the query:

```
query(secFog(appOp,smartbuilding,D)).
```

SecFog Output

IoTController	DataStorage	Dashboard	Security
Cloud 1	Cloud 1	Cloud 1	0.82
Cloud 1	Cloud 1	Cloud 2	0.81
Cloud 1	Cloud 1	Edge 3	0.78
Cloud 1	Edge 3	Cloud 1	0.77
Cloud 1	Edge 3	Cloud 2	0.75
Cloud 1	Edge 3	Edge 3	0.75
Cloud 2	Cloud 1	Cloud 1	0.81
Cloud 2	Cloud 1	Cloud 2	0.81
Cloud 2	Cloud 1	Edge 3	0.77
Cloud 2	Edge 3	Cloud 1	0.75
Cloud 2	Edge 3	Cloud 2	0.89
Cloud 2	Edge 3	Edge 3	0.87
Edge 2	Cloud 1	Cloud 1	0.66
Edge 2	Cloud 1	Cloud 2	0.65
Edge 2	Cloud 1	Edge 3	0.63
Edge 2	Edge 3	Cloud 1	0.62
Edge 2	Edge 3	Cloud 2	0.72
Edge 2	Edge 3	Edge 3	0.72
Edge 3	Cloud 1	Cloud 1	0.80
Edge 3	Cloud 1	Cloud 2	0.78
Edge 3	Cloud 1	Edge 3	0.78
Edge 3	Edge 3	Cloud 1	0.77
Edge 3	Edge 3	Cloud 2	0.89
Edge 3	Edge 3	Edge 3	0.89

Semiring-based Trust Models

The default trust model of SecFog shows two main limitations:

1. it is **monotonic** (i.e., all paths towards a certain provider Pcontribute increasing the trust degree towards it), and
2. it is **unconditionally transitive** (i.e., if A trusts B and B trusts C then A trusts C).

Semiring-based Trust Models

The default trust model of SecFog shows two main limitations:

1. it is **monotonic** (i.e., all paths towards a certain provider Pcontribute increasing the trust degree towards it), and
2. it is **unconditionally transitive** (i.e., if A trusts B and B trusts C then A trusts C).

To overcome (1) and (2), research proposed more sophisticate models based on semiring which can be easily embedded in SecFog.

Commutative Semirings

A (commutative) **semiring** is an algebraic structure consisting of a 5-tuple:

$$\langle \mathcal{S}, \oplus, \otimes, 0, 1 \rangle$$

where $\mathcal{S}$ is a set of elements, and $\oplus$ and $\otimes$ are two binary operators defined over such elements such that:

- $\oplus$ is commutative and associative, and 0 is its neutral element,
- $\otimes$ is associative, distributes over $\oplus$, and 1 and 0 are its neutral and absorbing elements, respectively.

Commutative Semirings

A (commutative) **semiring** is an algebraic structure consisting of a 5-tuple:

$$\langle \mathcal{S}, \oplus, \otimes, 0, 1 \rangle$$

where $\mathcal{S}$ is a set of elements, and $\oplus$ and $\otimes$ are two binary operators defined over such elements such that:

- $\oplus$ is commutative and associative, and 0 is its neutral element,
- $\otimes$ is associative, distributes over $\oplus$, and 1 and 0 are its neutral and absorbing elements, respectively.

For instance, the embedded model of Problog corresponds to the probability semiring

$$\langle \mathbb{R} \cap [0, 1], +, \times, 0, 1 \rangle$$

where $+$ and $\times$ denote classical addition and multiplication over real numbers.

Reasoning on Possible Worls: Intuition 2

Problog can be extended to arbitrary semirings $\langle \mathcal{S}, \oplus, \otimes, 0, 1 \rangle$ and labelling functions $\alpha(f)$ over the program literals.

Simply, the labelling for query q is obtained as

$$A(q) = \bigoplus_{W \in \Omega(q)} \bigotimes_{f \in W} \alpha(f)$$

A Different Trust Model

Theodorakopoulos and Baras* proposed the following model where trust is represented by couples $\langle t, c \rangle \in \mathcal{S} = (\mathbb{R} \cap [0, 1]) \times (\mathbb{R} \cap [0, 1])$ where t represents a trust value and c the confidence in such trust value assignment, i.e. the *quality* of the declared opinion.

*G. Theodorakopoulos, J. S. Baras, On trust models and trust evaluation metrics for ad-hoc networks, 2016

A Different Trust Model

Theodorakopoulos and Baras* proposed the following model where trust is represented by couples $\langle t, c \rangle \in \mathcal{S} = (\mathbb{R} \cap [0,1]) \times (\mathbb{R} \cap [0,1])$ where t represents a trust value and c the confidence in such trust value assignment, i.e. the *quality* of the declared opinion.

Semiring operations are as follows

$$\langle t, c \rangle \oplus \langle t', c' \rangle = \begin{cases} \langle t, c \rangle & if c > c' \\ \langle t', c' \rangle & if c' > c \\ \langle \max\{t, t'\}, c \rangle, & if c = c' \end{cases}$$

$$\langle t, c \rangle \otimes \langle t', c' \rangle = \langle tt', cc' \rangle$$

*G. Theodorakopoulos, J. S. Baras, On trust models and trust evaluation metrics for ad-hoc networks, 2016

A Different Trust Model

Theodorakopoulos and Baras* proposed the following model where trust is represented by couples $\langle t, c \rangle \in \mathcal{S} = (\mathbb{R} \cap [0, 1]) \times (\mathbb{R} \cap [0, 1])$ where t represents a trust value and c the confidence in such trust value assignment, i.e. the *quality* of the declared opinion.

Semiring operations are as follows

$$\langle t, c \rangle \oplus \langle t', c' \rangle = \begin{cases} \langle t, c \rangle & if c > c' \\ \langle t', c' \rangle & if c' > c \\ \langle \max\{t, t'\}, c \rangle, & if c = c' \end{cases}$$

$$\langle t, c \rangle \otimes \langle t', c' \rangle = \langle tt', cc' \rangle$$

This model is non-monotonic and optimistic. To make it pessimistic, just change $\max$ with $\min$.

*G. Theodorakopoulos, J. S. Baras, On trust models and trust evaluation metrics for ad-hoc networks, 2016

Implementation: αSecFog

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

:- use_semiring(
    sr_plus,   % addition (arity 3)
    sr_times,  % multiplication (arity 3)
    sr_zero,   % neutral element of addition
    sr_one,    % neutral element of multiplication
    sr_neg,    % negation of fact label
    false,     % requires solving disjoint sum problem?
    false).    % requires solving neutral sum problem?

sr_zero((0.0, 0.0)).
sr_one((1.0, 1.0)).
sr_times((Ta, Ca), (Tb, Cb), (Tc, Cc)) :- Tc is Ta*Tb, Cc is Ca*Cb.
sr_plus((Ta, Ca), (Tb, Cb), (Ta, Ca)) :- Ca > Cb.
sr_plus((Ta, Ca), (Tb, Cb), (Tb, Cb)) :- Cb > Ca.
sr_plus((Ta, Ca), (Tb, Cb), (Tc, Ca)) :- Ca == Cb, Tc is max(Ta, Tb).
sr_neg((Ta, Ca), (Tb, Ca)) :- Tb is 1.0-Ta.
```

File is on Moodle 😄

Condition Transitivity

Condition Transitivity

Condition to a radius `R=3`

```
trusts2(A,B) :- trusts2(A,B,3).
trusts2(A,B,D) :-
    D > 0,
    trusts(A,B).
 trusts2(A,B,D) :-
    D > 0,
    trusts(A,C),
    NewD is D - 1,
    trusts2(C,B,NewD).
```

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

- 🔐 Explain *why* a given deployment is NOT secure (and suggest how to fix it)

Future Work

- 🔐 Explain *why* a given deployment is NOT secure (and suggest how to fix it)
- ⌚ Improve scalability over large infrastructures (e.g., via continuous reasoning and meta-heuristics)

Future Work

- 🔐 Explain *why* a given deployment is NOT secure (and suggest how to fix it)
- ⌚ Improve scalability over large infrastructures (e.g., via continuous reasoning and meta-heuristics)
- ✏️ Assess our declarative methodologies over real testbeds

🤓💻 Ain't Prolog cool?

An (Interactive) Introduction to

Stefano Forti

Department of Computer Science, University of Pisa, Italy

Reading: S. Forti, G.-L. Ferrari, A. Brogi, [Secure Cloud-Edge Deployments, with Trust], 2021.

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