The FROST Language

A trusted and user-centric access control language: Enabling delegation of fine-grained policies in shared ecosystems

The Founders

Leif-Nissen Lundbæk

Mathematician (Oxford University)
& former Cryptographer at Daimler

CEO

Prof. Michael R. Huth, PhD

Professor of Computer Science at Imperial College London

CTO

Felix Hahmann

Computer Scientist & former
Daimler Project Manager

COO

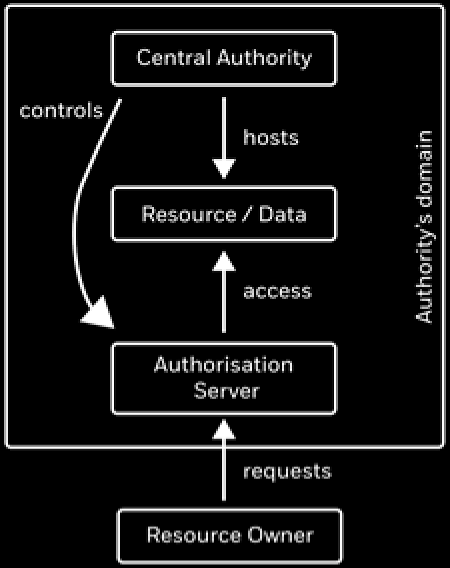
The Illusion of User Control

Authority Managed Access Control

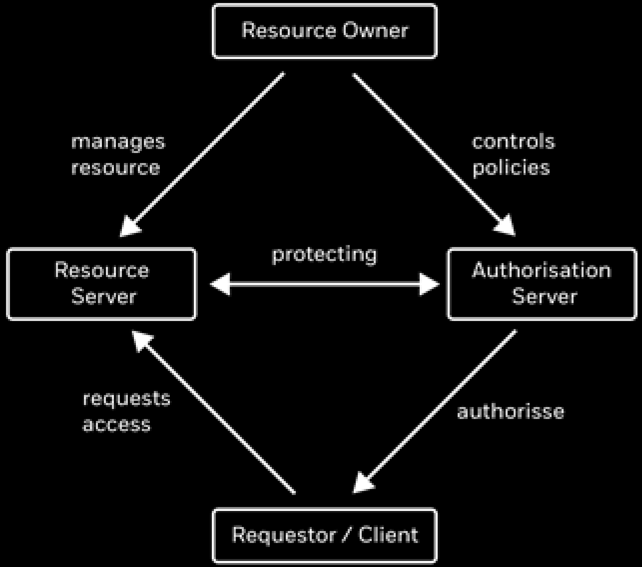
- Platforms (e.g. Amazon Key) centrally manage access control for users
- Resource owners lack control and are dependent on authority
- Limited interoperability and fragmentation

User Managed Access Solution

Authority Managed Access



User Managed Access (UMA)
based on OAuth



Users lack Control over Delegation

Limited control of policies in OAuth2

- OAuth2 for system interoperability of delegation of access
- but limited control of fine-grained policies of delegation to 3rd parties, thus:
- limited P2P sharing possibilities (e.g. a leased car is shared)

Lack of Trust in Shared Infrastructures

Delegation of policies requires trust

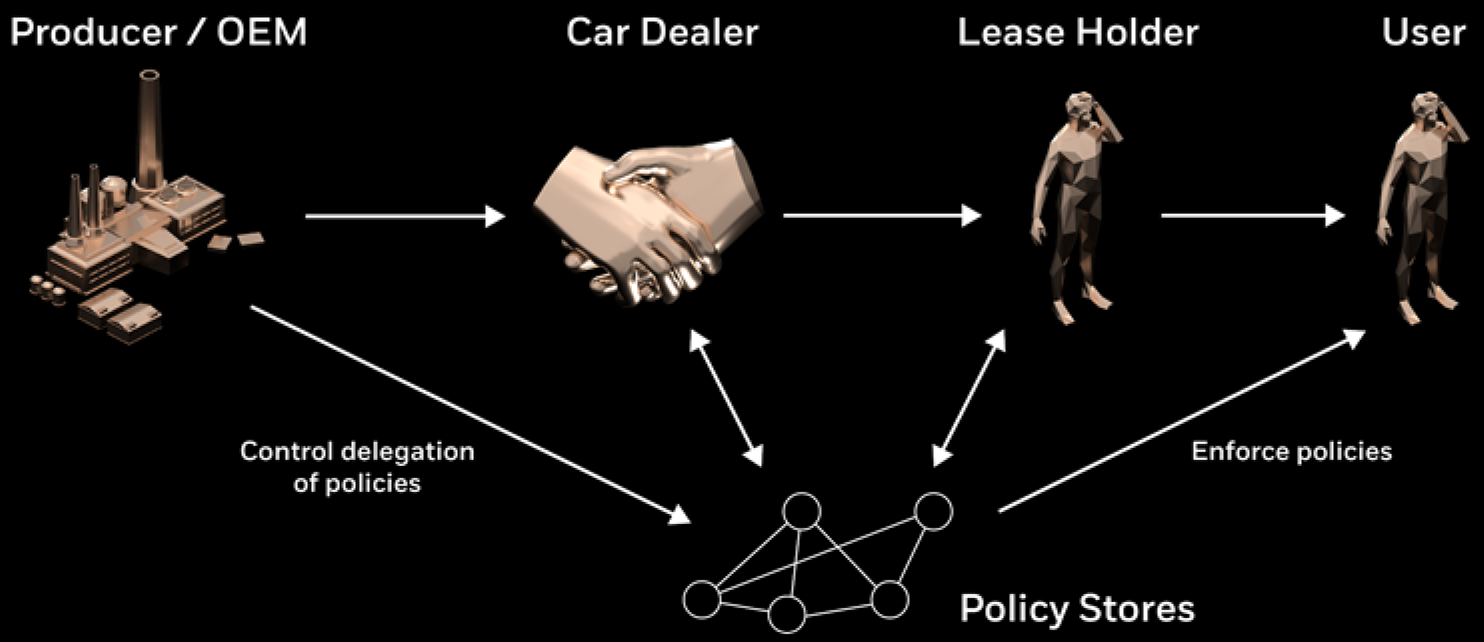
- Centralized storage of policies not trusted by participants, thus:
- OAuth2 is failing to provide system interoperability for complex policies
- But sharing economies require trustworthy interoperable ecosystems

FROST: Our Solution Stack

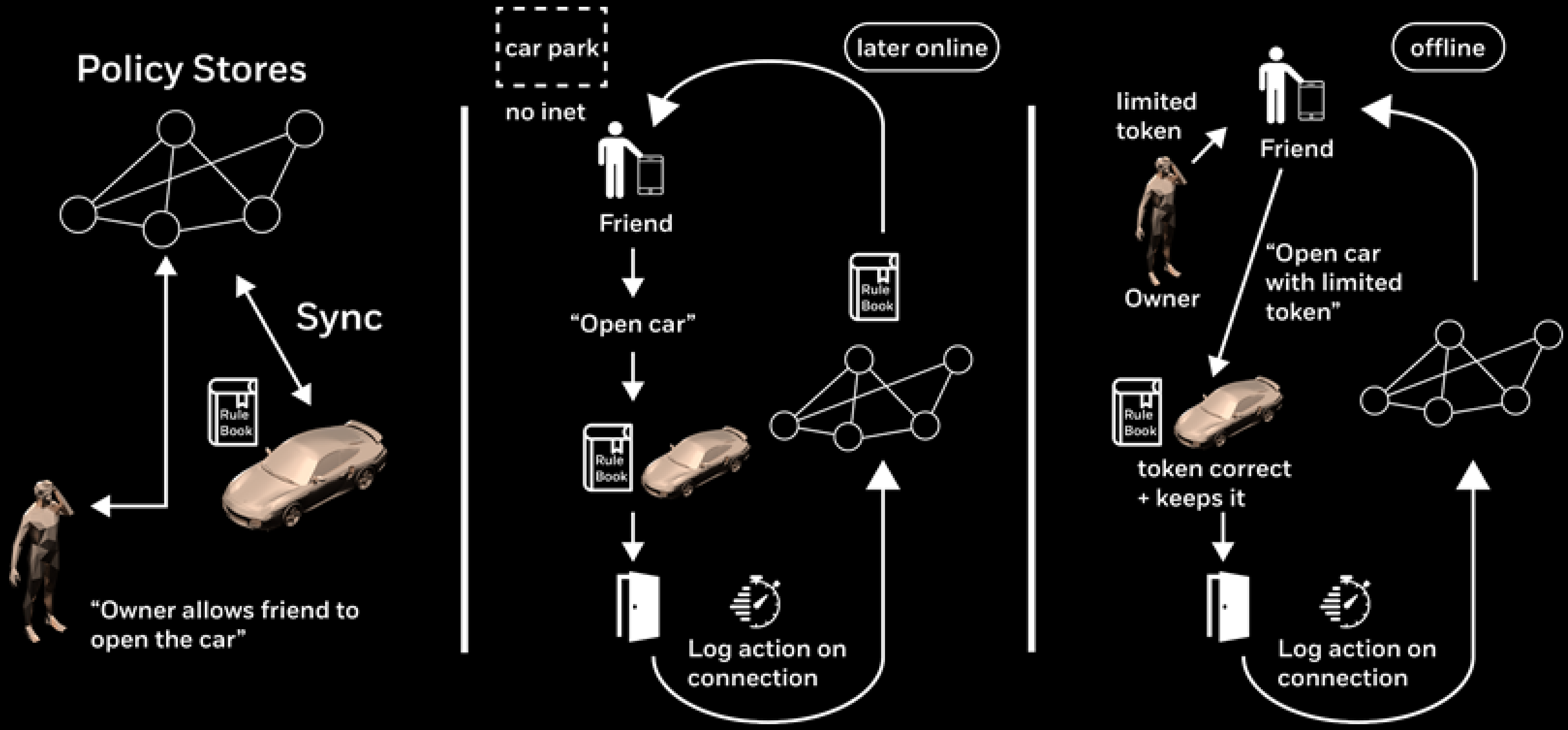
- Stakeholders can program shared ecosystems using *smart* policies for access control and cybersecurity protocols for powerful *delegation*.

  Let’s explore examples before we take a deep technical dive.

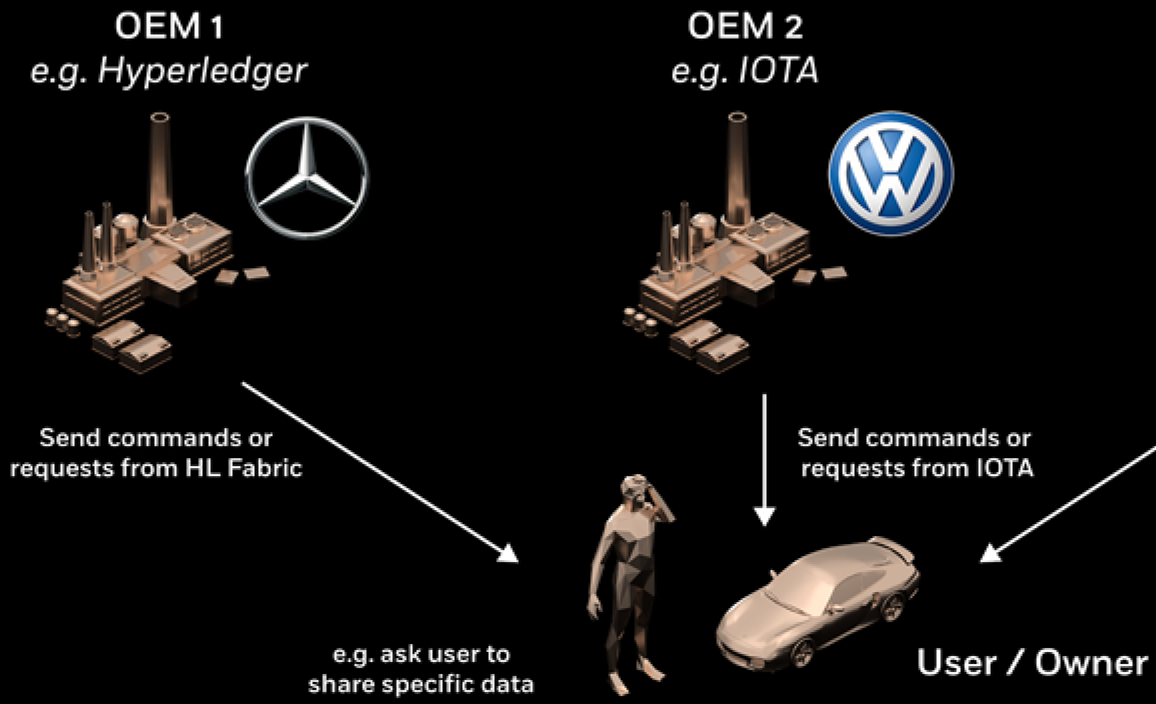
Example: Policy Delegation in Vehicle Lifecycle



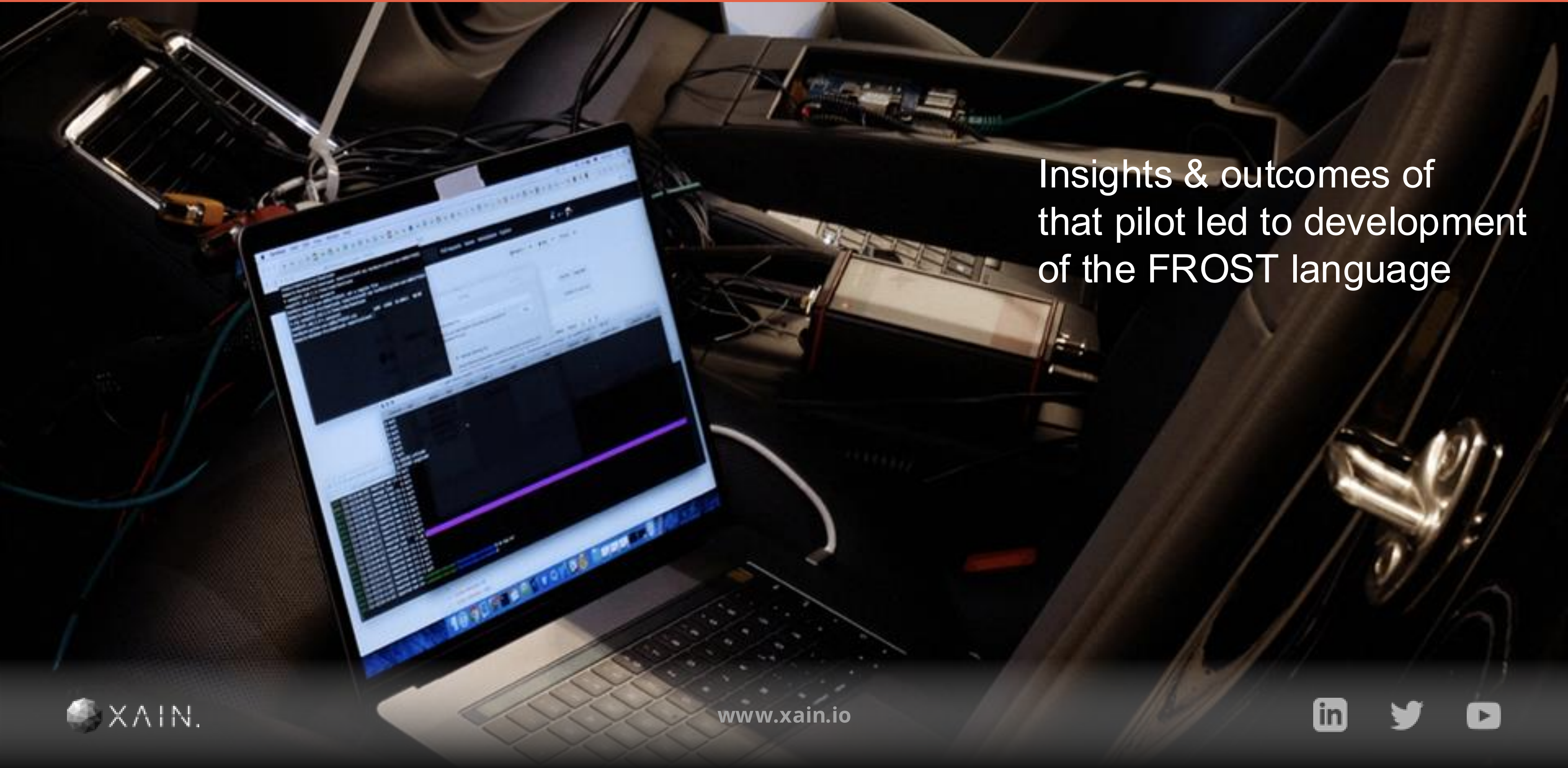
Example: P2P Device Sharing



Example: Interoperability through Expressive FROST Language



Example: Porsche Integration



Insights & outcomes of that pilot led to development of the FROST language

Advantages & USPs of FROST Technology

- FROST provides fine-grained delegation of access for human to machine and machine to machine interactions
- FROST supports domain-specific languages for integrating other systems, e.g. Ethereum, HL Fabric, IOTA
- FROST is based on previous academic research (e.g. PBel)

FROST: A Technical Dive

- What does the FROST language look like?

  How does delegation work?

“FROST” spelled out

- **F = Flexible**, e.g. freedom to delegate, agnostic view of backends
- **R = Resilient**, e.g. through cybersecurity protocols and policy authentication
- **O = Open**, FROST will become open-source, allows stakeholders to create bespoke shared ecosystems
- **S = Service-Enabling**, e.g. tracking parts in life-cycle
- **T = Trusted**, e.g. combination of cybersecurity, validation tools

FROST Technology Stack: recap

- A core language for expressing policies for access controls
- Cybersecurity protocols express/enforce delegation structures
- Access-control architecture that enables “F”, “R”, “O”, “S”, and “T”
- Programming Language tools for testing and validation: compilers, automated provers of correctness, and so forth

FROST Language: based on *rules* & *attributes*

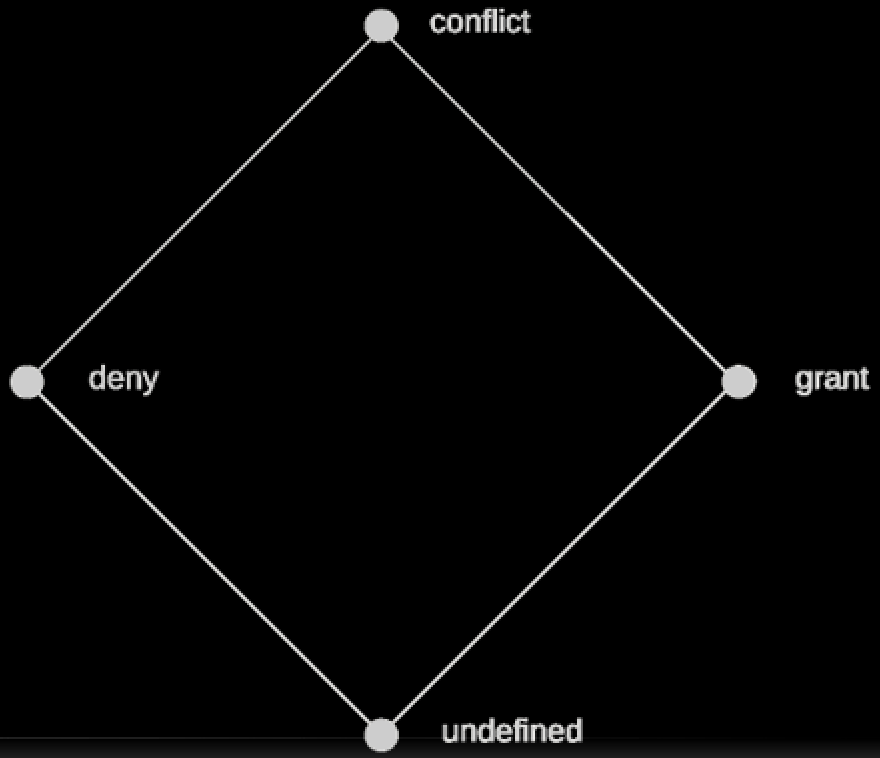
$$
\begin{aligned}
\texttt{grant if}\quad & (\mathbf{object} == vehicle)\ \&\&\ (\mathbf{subject} == vehicle.owner.daughter)\ \&\& \\
& (\mathbf{action} == driveVehicle)\ \&\& \\
& (owner.daughter.isInsured == true)\ \&\& \\
& (0900 \leqslant localTime)\ \&\&\ (localTime \leqslant 2000)
\end{aligned}
$$

Open System Composition: Gaps & Conflicts

conflict =
evidence for **deny** *and* **grant**

undefined =
evidence for neither **grant** nor **deny**

- a policy “gap”



FROST Core Language

$$
\begin{aligned}
dec &::= \texttt{grant} \mid \texttt{deny} \mid \texttt{undef} \mid \texttt{conflict} \\
term &::= constant \mid entity \mid op(term, \ldots, term) \mid term.attribute \\
cond &::= (term == term) \mid (term < term) \mid (term \leqslant term) \mid \cdots \\
&\qquad \mathsf{true} \mid \neg cond \mid (cond \; \&\& \; cond) \mid (cond \,||\, cond) \\
rule &::= \texttt{grant if } cond \mid \texttt{deny if } cond \\
guard &::= \mathsf{true} \mid pol \; \texttt{eval} \; dec \mid (guard \; \&\& \; guard) \\
pol &::= dec \mid rule \mid \texttt{case}\,\{\, [guard : pol]^{+} \, [\mathsf{true} : pol] \,\}
\end{aligned}
$$

Example case-policy: information *join*

... of policies

P, *Q*

```
case {
    [(P eval undef): Q]
    [(Q eval undef): P]
    [(P eval conflict): conflict]
    [(Q eval conflict): conflict]
    [((P eval deny) && (Q eval grant)): conflict]
    [((P eval grant) && (Q eval deny)): conflict]
    [true: P]
}
```

Example case-policy: information *join*

$$
\begin{array}{l}
\texttt{case}\ \{\\
\quad [(P\ \texttt{eval undef}) \colon Q]\\
\quad [(Q\ \texttt{eval undef}) \colon P]\\
\quad [(P\ \texttt{eval conflict}) \colon \texttt{conflict}]\\
\quad [(Q\ \texttt{eval conflict}) \colon \texttt{conflict}]\\
\quad [((P\ \texttt{eval deny})\ \&\&\ (Q\ \texttt{eval grant})) \colon \texttt{conflict}]\\
\quad [((P\ \texttt{eval grant})\ \&\&\ (Q\ \texttt{eval deny})) \colon \texttt{conflict}]\\
\quad [\mathsf{true} \colon P]\\
\}
\end{array}
$$

… of policies

P, *Q*

$$
\begin{array}{rcl}
\mathit{dec} & ::= & \texttt{grant} \mid \texttt{deny} \mid \texttt{undef} \mid \texttt{conflict}\\
\mathit{term} & ::= & \mathit{constant} \mid \mathit{entity} \mid \mathit{op}(\mathit{term},\ldots,\mathit{term}) \mid \mathit{term.attribute}\\
\mathit{cond} & ::= & (\mathit{term} == \mathit{term}) \mid (\mathit{term} < \mathit{term}) \mid (\mathit{term} \leqslant \mathit{term}) \mid \cdots\\
 & & \mathsf{true} \mid \neg\mathit{cond} \mid (\mathit{cond}\ \&\&\ \mathit{cond}) \mid (\mathit{cond} \,\|\, \mathit{cond})\\
\mathit{rule} & ::= & \texttt{grant if}\ \mathit{cond} \mid \texttt{deny if}\ \mathit{cond}\\
\mathit{guard} & ::= & \mathsf{true} \mid \mathit{pol}\ \texttt{eval}\ \mathit{dec} \mid (\mathit{guard}\ \&\&\ \mathit{guard})\\
\mathit{pol} & ::= & \mathit{dec} \mid \mathit{rule} \mid \texttt{case}\ \{\ [\mathit{guard} \colon \mathit{pol}]^{+}\ [\mathsf{true} \colon \mathit{pol}]\ \}
\end{array}
$$

Example case-policy: information *join*

$$
\begin{array}{l}
\mathtt{case}\ \{\\
\quad [(P\ \mathtt{eval}\ \mathtt{undef}) \colon Q]\\
\quad [(Q\ \mathtt{eval}\ \mathtt{undef}) \colon P]\\
\quad [(P\ \mathtt{eval}\ \mathtt{conflict}) \colon \mathtt{conflict}]\\
\quad [(Q\ \mathtt{eval}\ \mathtt{conflict}) \colon \mathtt{conflict}]\\
\quad [((P\ \mathtt{eval}\ \mathtt{deny})\ \&\&\ (Q\ \mathtt{eval}\ \mathtt{grant})) \colon \mathtt{conflict}]\\
\quad [((P\ \mathtt{eval}\ \mathtt{grant})\ \&\&\ (Q\ \mathtt{eval}\ \mathtt{deny})) \colon \mathtt{conflict}]\\
\quad [\mathsf{true} \colon P]\\
\}
\end{array}
$$

... of policies

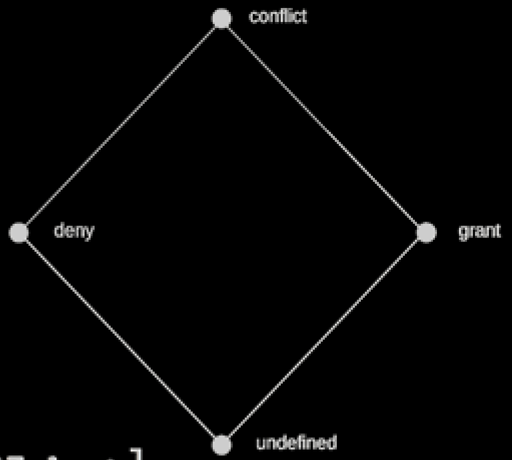
P, *Q*

$$
\begin{array}{rcl}
dec & ::= & \mathtt{grant} \mid \mathtt{deny} \mid \mathtt{undef} \mid \mathtt{conflict}\\
term & ::= & constant \mid entity \mid op(term, \ldots, term) \mid term.attribute\\
cond & ::= & (term == term) \mid (term < term) \mid (term \leqslant term) \mid \cdots\\
 & & \mathsf{true} \mid \neg cond \mid (cond\ \&\&\ cond) \mid (cond \mid\mid cond)\\
rule & ::= & \mathtt{grant\ if}\ cond \mid \mathtt{deny\ if}\ cond\\
guard & ::= & \mathsf{true} \mid pol\ \mathtt{eval}\ dec \mid (guard\ \&\&\ guard)\\
pol & ::= & dec \mid rule \mid \mathtt{case}\ \{\ [guard \colon pol]^{+}\ [\mathsf{true} \colon pol]\ \}
\end{array}
$$

Q. If we get here, what do *P* and *Q* eval to?

Example case-policy: information *join*

… of policies
P, *Q*



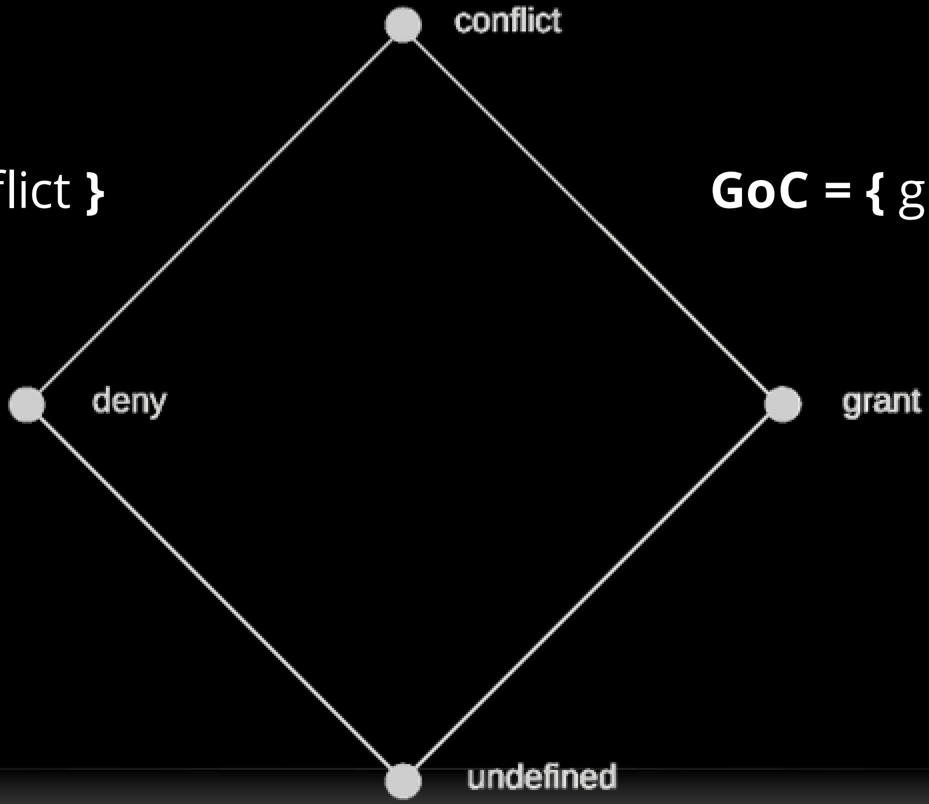

```
case {
    [(P eval undef): Q]
    [(Q eval undef): P]
    [(P eval conflict): conflict]
    [(Q eval conflict): conflict]
    [((P eval deny) && (Q eval grant)): conflict]
    [((P eval grant) && (Q eval deny)): conflict]
    [true: P]
}
```

Q. If we get here, what do *P* and *Q* eval to?

Compiling Policies into Circuits: GoC, DoC

DoC = { deny, conflict **}**

GoC = { grant, conflict **}**



GoC Compilation, Join Normalform

$$\mathsf{GoC}(\mathtt{grant}) \equiv \mathsf{true} \qquad \mathsf{GoC}(\mathtt{deny}) \equiv \mathsf{false}$$

$$\mathsf{GoC}(\mathtt{conflict}) \equiv \mathsf{true} \qquad \mathsf{GoC}(\mathtt{undef}) \equiv \mathsf{false}$$

$$\mathsf{GoC}(\mathtt{grant\ if}\ cond) \equiv cond \qquad \mathsf{GoC}(\mathtt{deny\ if}\ cond) \equiv \mathsf{false}$$

$$\begin{aligned}\mathsf{GoC}(\mathsf{case}\,\{\,[g_1 : p_1] \ldots [g_{n-1} : p_{n-1}]\,[\mathsf{true}\!: p_n]\,\}) \equiv\ & (\mathsf{R}(g_1)\ \&\&\ \mathsf{GoC}(p_1)) \mid\mid \cdots \\ & \cdots \mid\mid (\mathsf{R}(g_{n-1})\ \&\&\ \mathsf{GoC}(p_{n-1})) \mid\mid (\mathsf{R}(\mathsf{true})\ \&\&\ \mathsf{GoC}(p_n))\end{aligned}$$

$$\boxed{pol \equiv (\mathtt{grant\ if}\ \mathsf{GoC}(pol))\ \mathtt{join}\ (\mathtt{deny\ if}\ \mathsf{DoC}(pol))}$$

Reachability of Guards as Circuits

$$
\begin{aligned}
\mathsf{R}(g_1) &\equiv \quad \mathsf{T}(g_1) \\
\mathsf{R}(g_i) &\equiv \neg\mathsf{T}(g_1)\ \&\&\ \ldots\ \&\&\ \neg\mathsf{T}(g_{i-1})\ \&\&\ \mathsf{T}(g_i), \quad 1 < i < n \\
\mathsf{R}(\mathsf{true}) &\equiv \neg\mathsf{T}(g_1)\ \&\&\ \ldots\ \&\&\ \neg\mathsf{T}(g_{n-1})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{T}(\mathsf{true}) &\equiv \mathsf{true} \\
\mathsf{T}(g_1\ \&\&\ g_2) &\equiv \mathsf{T}(g_1)\ \&\&\ \mathsf{T}(g_2)
\end{aligned}
$$

$$
\mathsf{T}(pol\ \mathtt{eval}\ dec) \equiv \begin{cases}
\mathsf{GoC}(pol)\ \&\&\ \mathsf{DoC}(pol) & \text{if } dec \text{ equals } \mathtt{conflict} \\
\neg\mathsf{GoC}(pol)\ \&\&\ \mathsf{DoC}(pol) & \text{if } dec \text{ equals } \mathtt{deny} \\
\mathsf{GoC}(pol)\ \&\&\ \neg\mathsf{DoC}(pol) & \text{if } dec \text{ equals } \mathtt{grant} \\
\neg\mathsf{GoC}(pol)\ \&\&\ \neg\mathsf{DoC}(pol) & \text{if } dec \text{ equals } \mathtt{undef}
\end{cases}
$$

Verification: Satisfiability Modulo Theories

Example theory: $\forall agents: 0 \leqslant agent.reputation \leqslant 1$

Satisfiability modulo theories: formula *and* theory are true

$$\phi \wedge \bigwedge_{\psi \in \mathcal{T}} \psi$$

Example: does circuit *c* represent policy *p*?

1. Generate *GoC(p)* and *DoC(p)*
2. *!GoC(p) && !DoC(p) unsatisfiable?* Then *p* has no gaps.
3. *GoC(p) && DoC(p) unsatisfiable?* Then *p* has no conflicts.
4. *GoC(p) and c logically equivalent?*

Accept circuit *c* only if all checks 2.-4. are positive.

Example: Change Management

Want assurance that updated policy *pol'* not more **permissive** than original *pol*

e.g. *pol'* doesn't grant whenever *pol* denied or had a gap.

Can assure this by showing unsatisfiability of

$$\mathsf{T}(pol' \; \texttt{eval grant}) \;\&\&\; (\mathsf{T}(pol \; \texttt{eval undef}) \;||\; \mathsf{T}(pol \; \texttt{eval deny}))$$

FROST Policy Evaluation

- policy evaluation cannot get stuck - *if all attributes evaluate to meaningful values*
  - but FROST can also securely deal with **incomplete** attribute information (*Kleene's 3-valued logic, etc.*)
- **Q**. are all evaluation paths of a policy **reachable**?

Dead-Code Analysis: remove unreachable paths

Assume: P can only eval to `grant` or `undef`

Q cannot eval to `undef`

$\Rightarrow$ *P join Q* simplifies to…?

```
case {
  [(P eval undef): Q]
  [(Q eval undef): P]
  [(P eval conflict): conflict]
  [(Q eval conflict): conflict]
  [((P eval deny) && (Q eval grant)): conflict]
  [((P eval grant) && (Q eval deny)): conflict]
  [true: P]
}
```

Dead-Code Analysis: remove unreachable paths

Assume: P can only eval to `grant` or `undef`

Q cannot eval to `undef`

$\Rightarrow$ *P join Q* simplifies to…?

```
case {
  [(P eval undef): Q]
  [(Q eval conflict): conflict]
  [((P eval grant) && (Q eval deny)): conflict]
  [true: P]
}
```

```
case {
  [(P eval undef): Q]
  [(Q eval undef): P]
  [(P eval conflict): conflict]
  [(Q eval conflict): conflict]
  [((P eval deny) && (Q eval grant)): conflict]
  [((P eval grant) && (Q eval deny)): conflict]
  [true: P]
}
```

Dead-Code Removal: Algorithm for Case-Policy

```
REM = ∅;
for (i = 1 to n − 1) {
  if (SAT_T (R(g_i)) returns "unsatisfiable") {
    remove case g_i : p_i from case-policy;
    recompute R(g_j) for all i < j ⩽ n;
    REM = REM ∪ {i};
  }
  if (REM = {1, ..., n − 1}) {
    return p_n;
  }
// R(g_i) is "satisfiable" for all i ∈ {1, ..., n − 1} \ REM
if (SAT_T (R(true)) returns "unsatisfiable") {
  remove case true: p_n;
  m = max({1, ..., n − 1} \ REM);
  if (|{1, ..., n − 1} \ REM| == 1) {
    return p_m;
  } else {
    g_m = true;
    return computed case-policy;
  }
}
}
```

Two Types of Obligations in Access Control

1. Obligations on system, e.g. log granted/performed access
2. Obligations on actors, e.g. to make a payment for access

Enforcing fulfillment of obligations:

- easier for 1.
- may benefit from escrow service for 2.

Obligations in FROST

Grants and denials may incur obligations:

$$rule ::= \texttt{grant}\,\{obl^*\}\ \texttt{if}\ cond \mid \texttt{deny}\,\{obl^*\}\ \texttt{if}\ cond$$

Idea: only collect obligations that are consistent with computed policy decision

Flexible Delegation: FROST Design Principles

- Delegator can override delegatee’s decision
- Delegatees can write & administer policies
- Resource owner can set delegation depth
- Delegation chain has composition idioms
- Delegation composition must be verifiable
- Any request maps to unique delegation chain

Delegation: Example Composition Idiom

Conservative delegation chain:

$$p_0 \gg (p_1 \gg (\cdots \gg p_n) \ldots)$$

Definition of *P* >> *Q*

```
case {
   [P eval conflict : deny]
   [P eval undef : Q]
   [true : P]
}
```

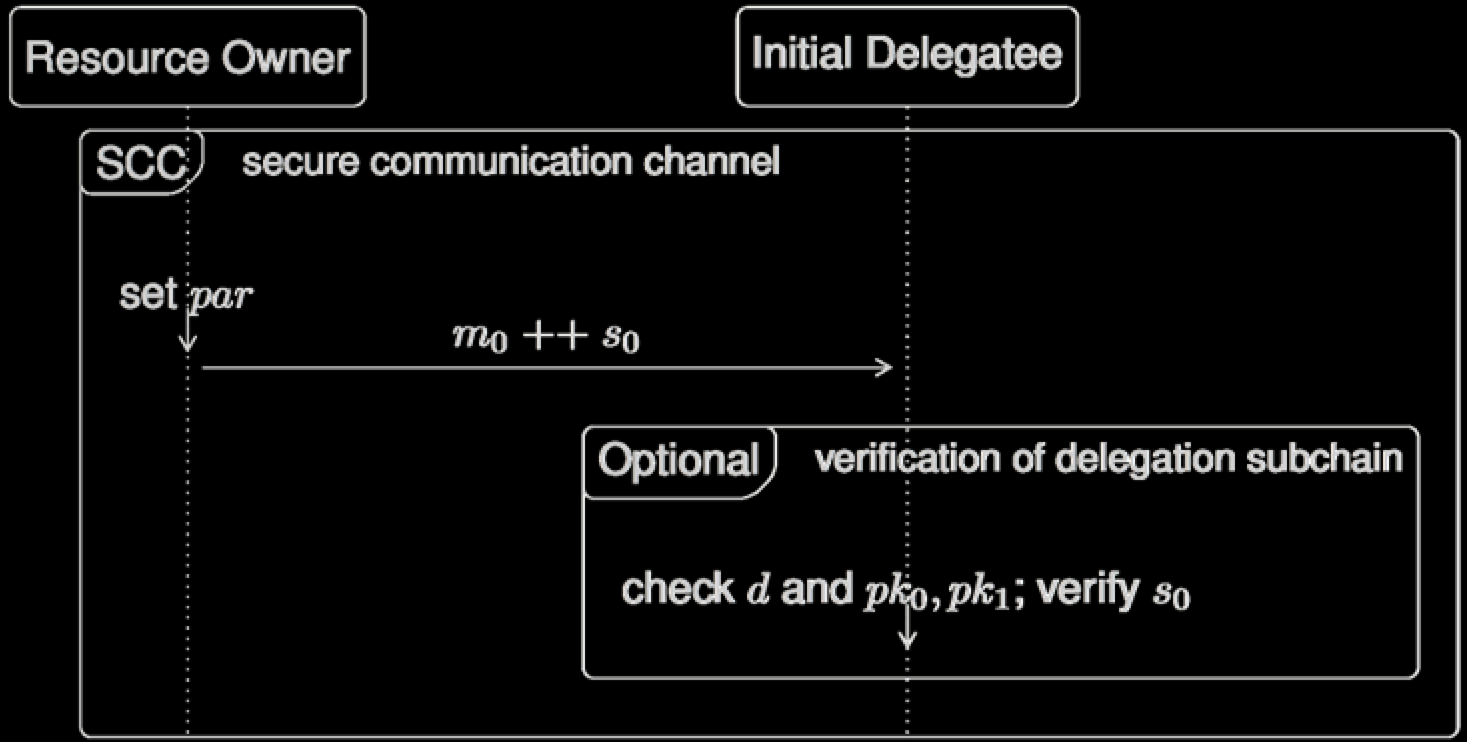
Initialization of Delegation Chain: Message

$$m_0 = par \mathbin{+\!\!+} pk_0 \mathbin{+\!\!+} p_0 \mathbin{+\!\!+} pk_1$$

$$s_0 = \mathrm{sign}(sk_0,\ H(m_0))$$

- *par*: system parameters, e.g. delegation depth *d*
- pk_0: public key of resource owner
- pk_1: public key of first delegatee
- sk_0: secret key corresponding to pk_0

Initialization of Delegation Chain: Protocol

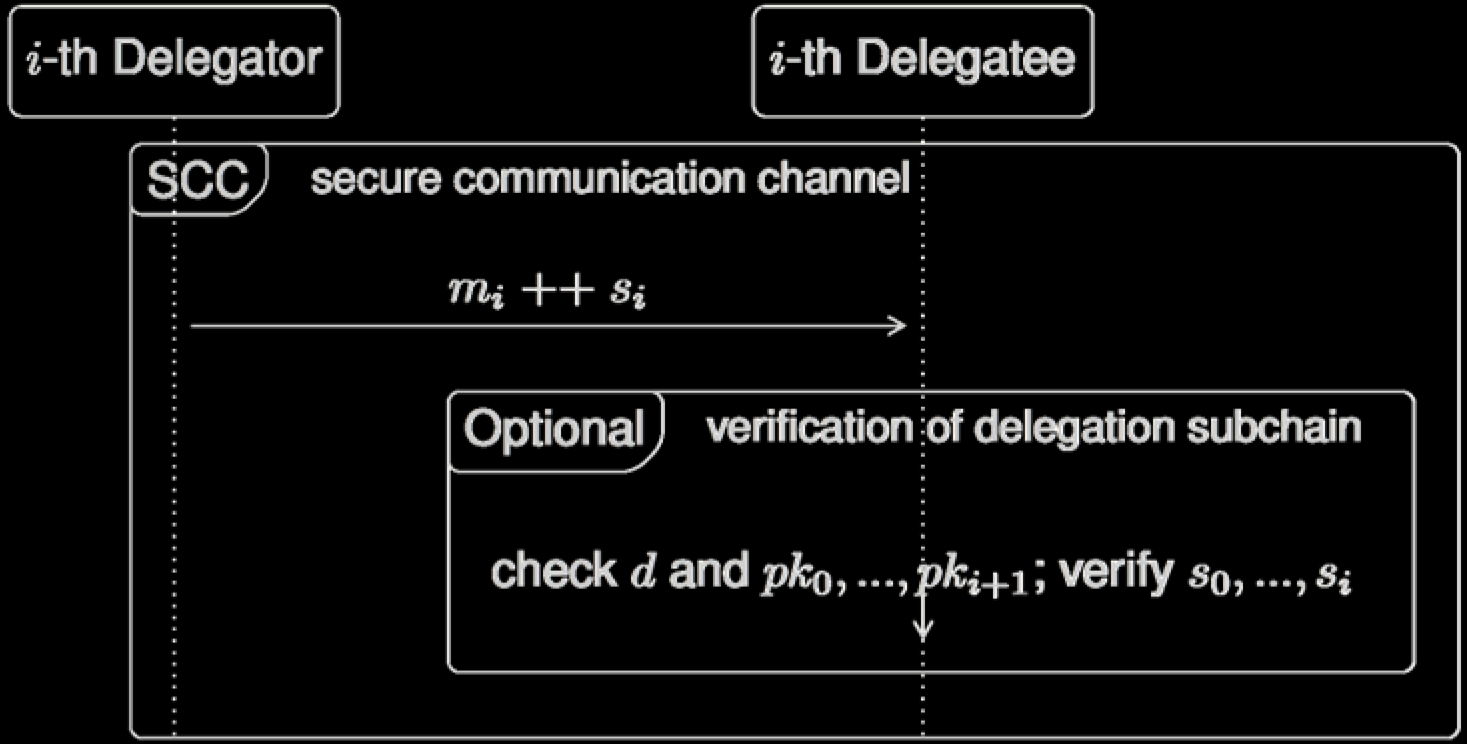


Extending Delegation Chain: Message

- m_i contains all policies, signatures from previous messages on chain ⇒ agent *i* can verify validity of the chain she sees

$$
\begin{aligned}
m_i = par \mathbin{+\!\!+} pk_0 \mathbin{+\!\!+} p_0 \mathbin{+\!\!+} pk_1 \mathbin{+\!\!+} s_0 \mathbin{+\!\!+} \cdots \\
\cdots \mathbin{+\!\!+} pk_{i-1} \mathbin{+\!\!+} p_{i-1} \mathbin{+\!\!+} pk_i \mathbin{+\!\!+} s_{i-1} \mathbin{+\!\!+} pk_i \mathbin{+\!\!+} p_i \mathbin{+\!\!+} pk_{i+1} \\
s_i = \mathsf{sign}(sk_i, H(m_i))
\end{aligned}
$$

Extending Delegation Chain: Protocol

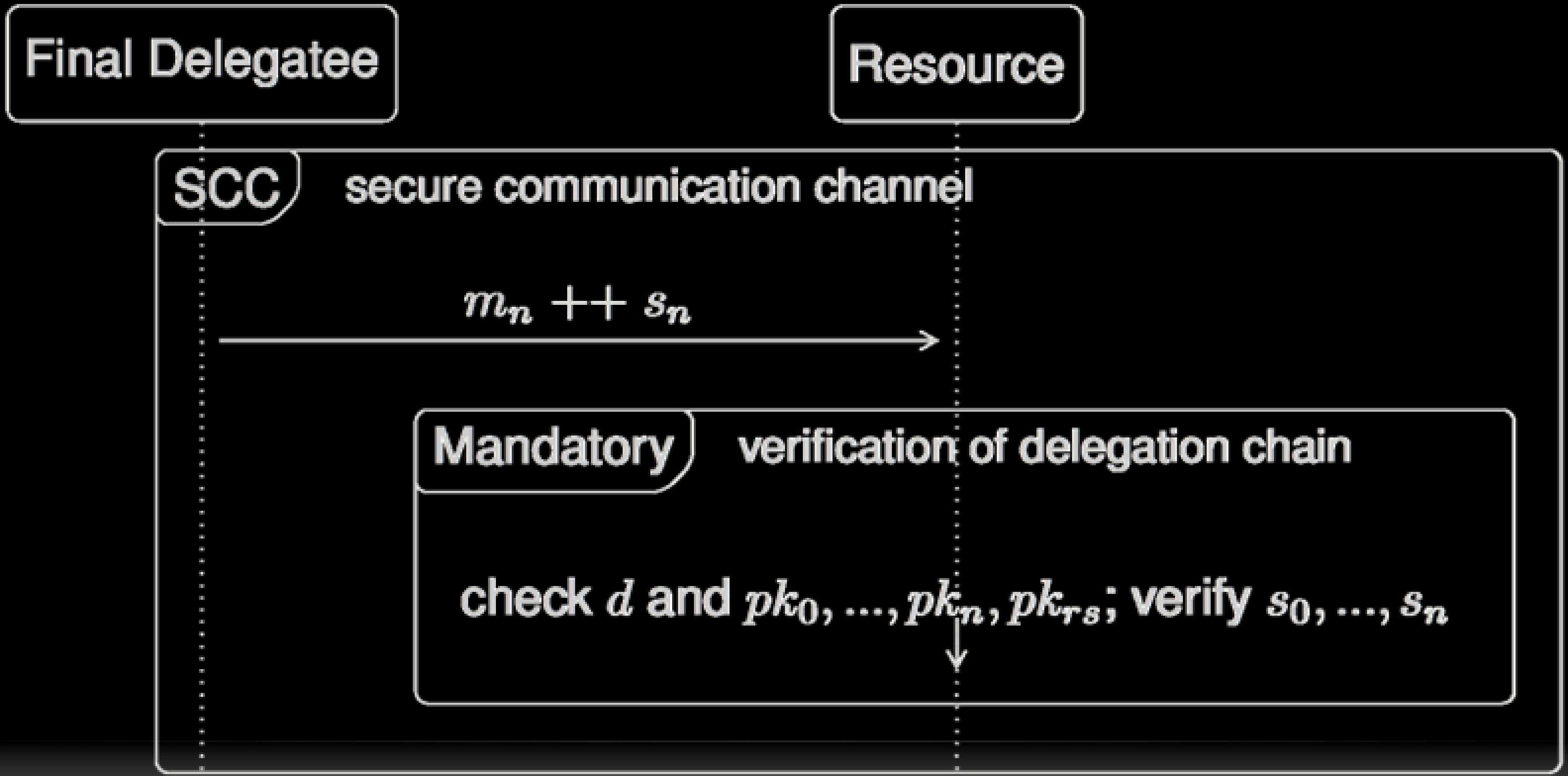


Completing Delegation Chain: Message

- similar to m_i but last agent n sends message to resource rs:

$$
\begin{aligned}
m_n = par \mathbin{+\!\!+} pk_0 \mathbin{+\!\!+} p_0 \mathbin{+\!\!+} pk_1 \mathbin{+\!\!+} s_0 \mathbin{+\!\!+} \cdots \\
\cdots \mathbin{+\!\!+} pk_{n-1} \mathbin{+\!\!+} p_{n-1} \mathbin{+\!\!+} pk_n \mathbin{+\!\!+} s_{n-1} \mathbin{+\!\!+} pk_n \mathbin{+\!\!+} p_n \mathbin{+\!\!+} pk_{rs} \\
s_n = \mathsf{sign}(sk_n,\, H(m_n))
\end{aligned}
$$

Completing Delegation Chain: Protocol

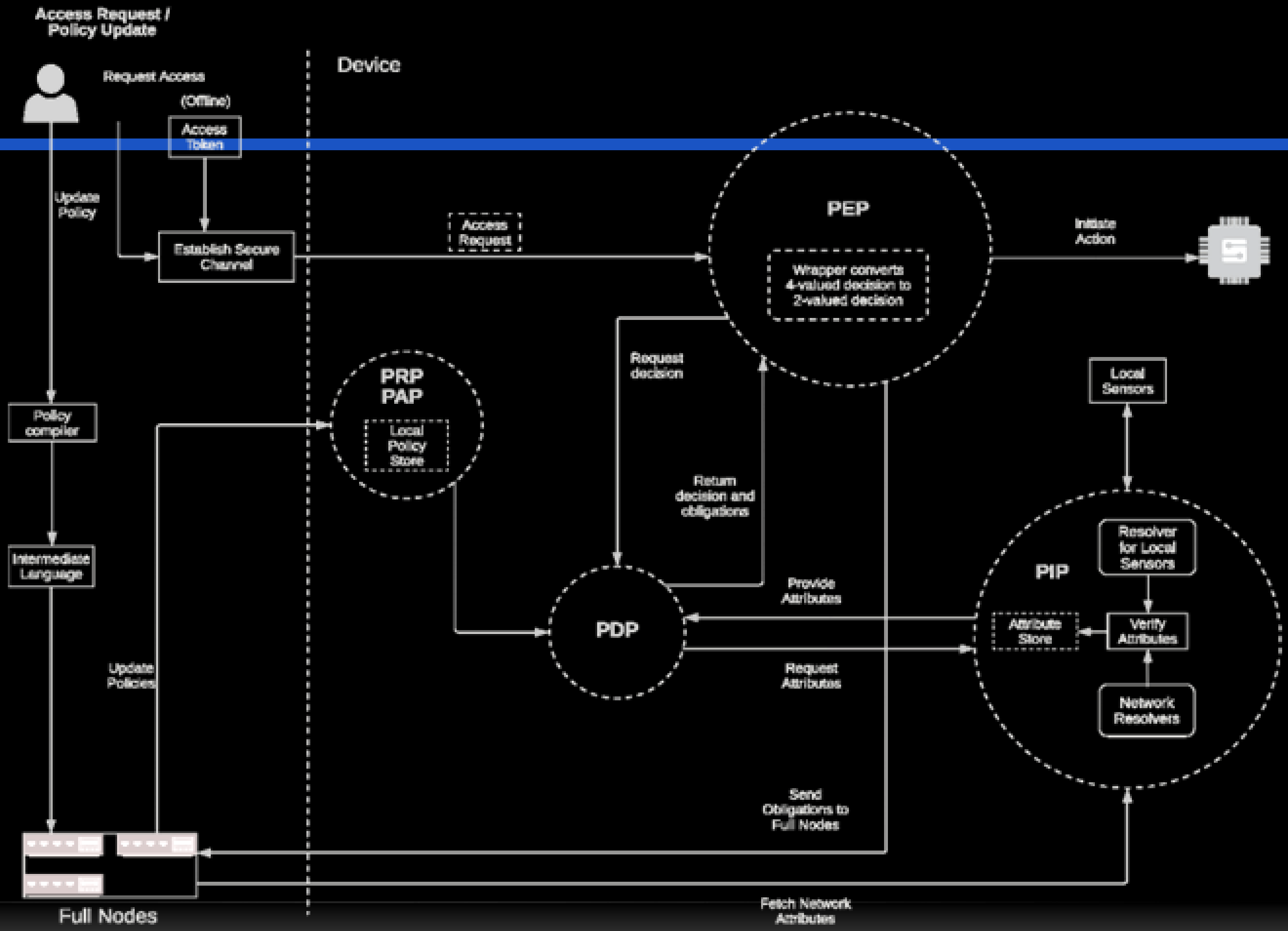


More on Delegation: See FROST Yellow Paper

- data structure for *policy delegation trees*
- *policy composition* on delegation chains
- *change management* on delegation chains
- *policy privacy* on delegation chains
- delegation and *cryptographic access tokens*

Architecture

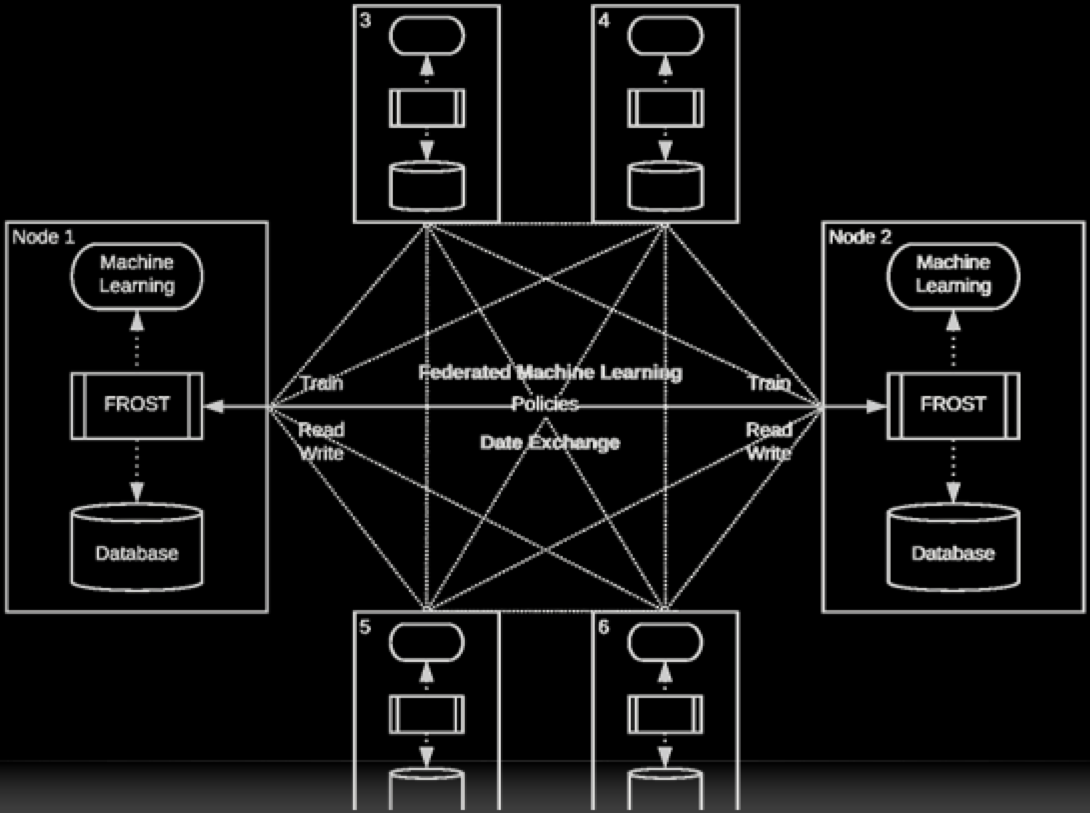
FROST Technology is not wedded to particular networks or data backends



Mitigating Potential Attack Vectors

- Choice of Cryptographic Primitives
- Risk-Aware Access Control
- Adversarial Withholding of Attribute Info
- Trusted Policy Lifecycle
- Policy Malware
- Exploiting Gaps Between Abstraction Layers

A Use Case: Training Access for Federated ML



“Access” to FROST Yellow Paper

Please get in touch with our Director of Marketing, Jesse Steele, at jesse.steele@xain.io, if you want a copy of that Yellow Paper prior to its public release.


XAIN.

The FROST Language

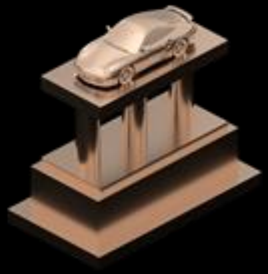
A trusted and user-centric access control language: Enabling delegation of fine-grained policies in shared ecosystems

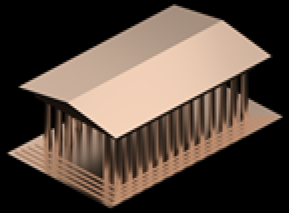
Kwok Cheung
Michael R. A. Huth
Laurence M. Kirk
Leif-Nissen Lundbæk
Rodolphe Marques
Jan Petsche

Yellow Paper: Version 0.9

Thank you for your kind attention!

Our History





2014 - 2017

Scientific start at Oxford University in 2014 and incorporation in Berlin 2017.

Jun 2017

Integration in manufacturing process for fine-grained policy delegations.

Jan 2018

Successful integration of first embedded Blockchain in a vehicle with Porsche.

May 2018

Founding of the XAIN Foundation to open-source the codebase for members.

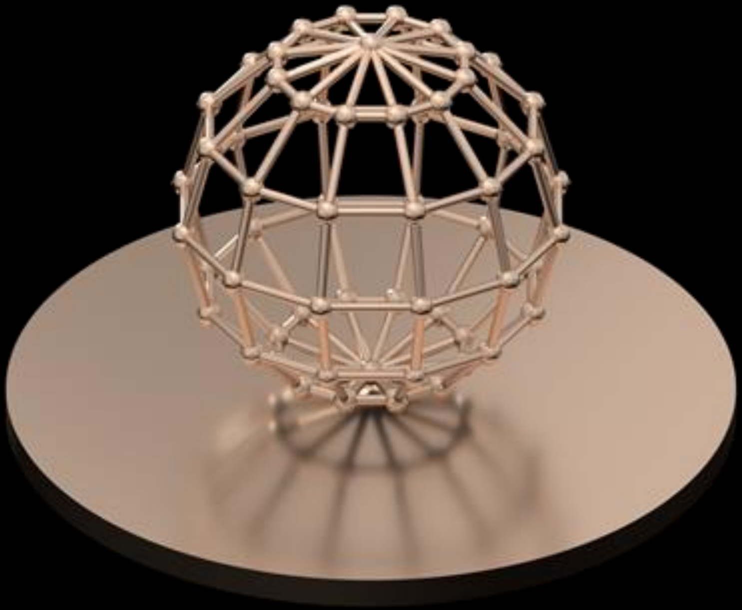
Our Clients

PORSCHE

VOLKSWAGEN
AKTIENGESELLSCHAFT

DAIMLER

Ideate. Co-create. Innovate.



40% of value capture depends on interoperability of systems

McKinsey Global Institute

XAIN.

visit www.xain.io