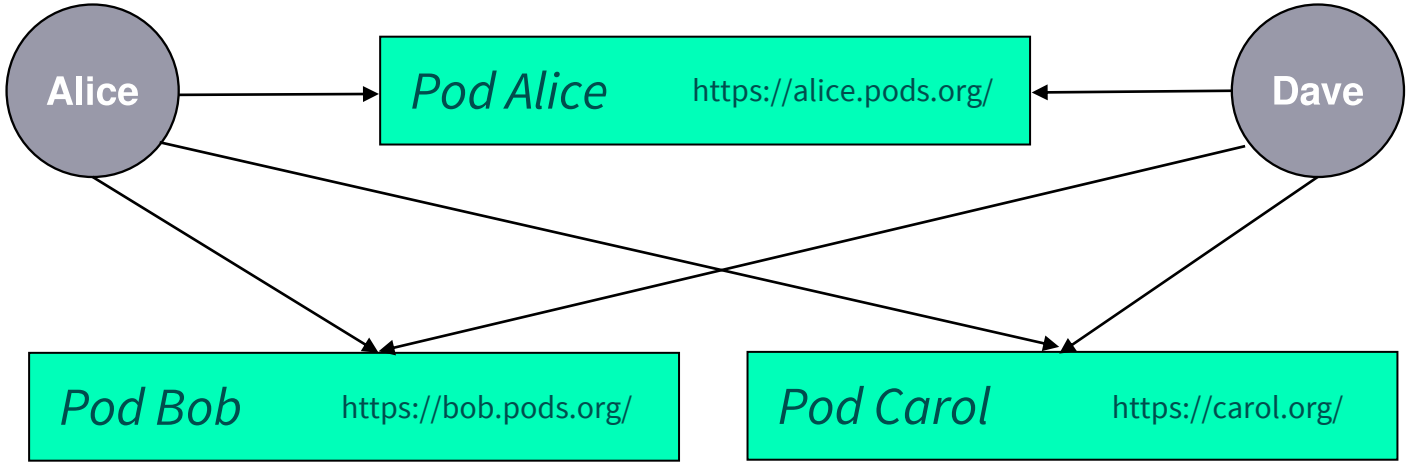
Towards Querying in Decentralized Environments with Privacy-Preserving Aggregation

Ruben Taelman, **Simon Steyskal**, and Sabrina Kirrane – **QuWeDa@ISWC 2020**

SIEMENS

Solid: a decentralized web-based ecosystem



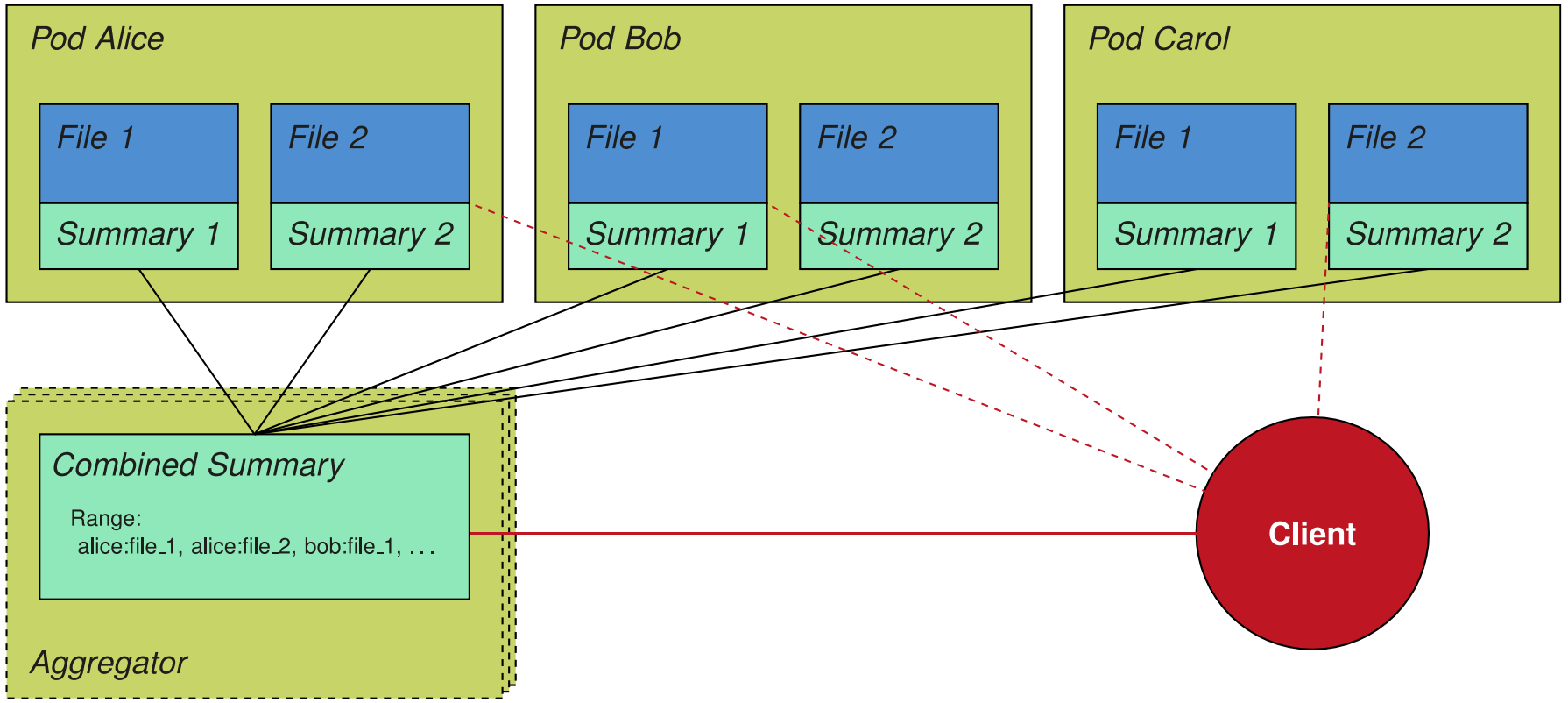
Decouples data from applications

Pods allow for more control over personal data

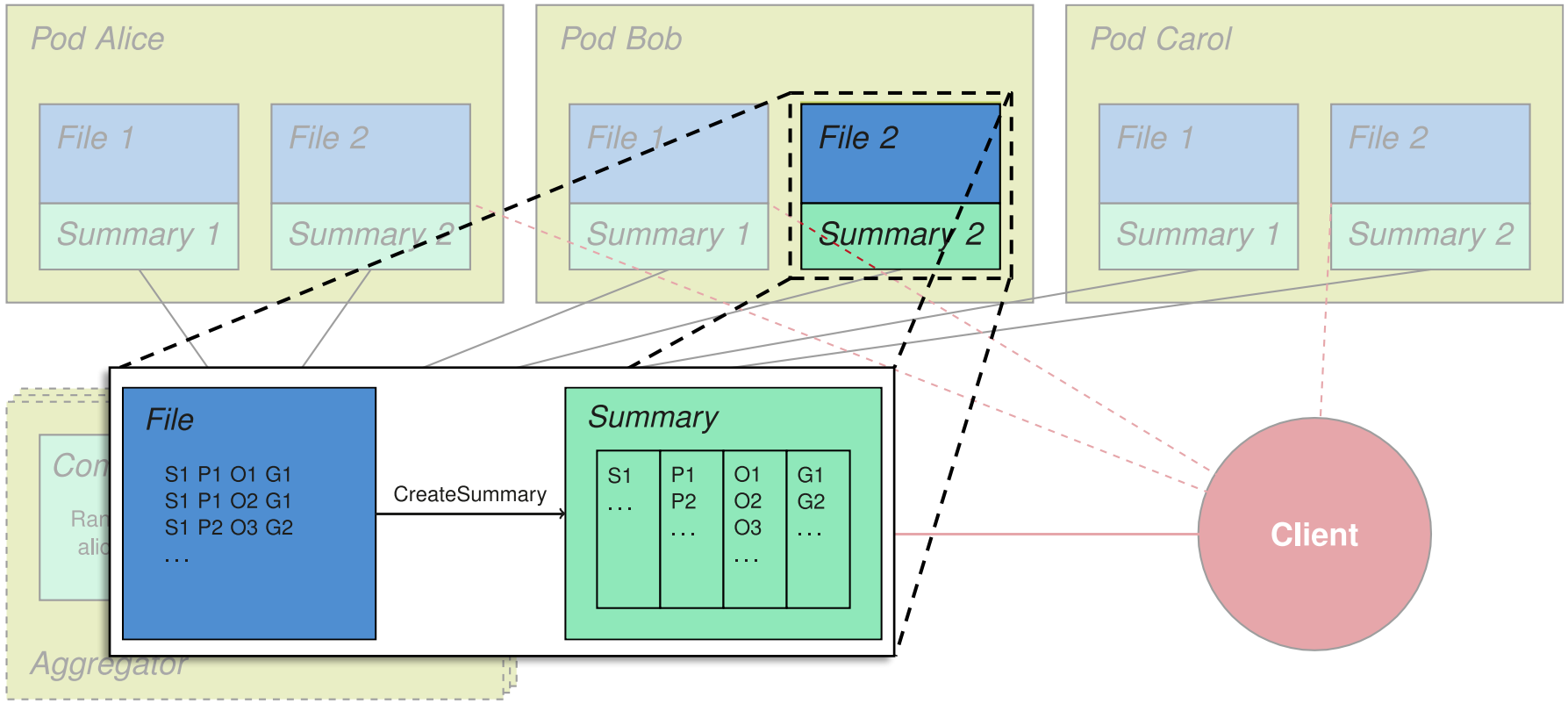
Based on open Web standards

Querying over personal (non-public) data stores must be both efficient and privacy preserving

Efficiency: Using Data Summaries for Efficient Querying



Efficiency: Using Data Summaries for Efficient Querying



Note: The summary values are not necessarily an exact representation of the way the summary is stored, these values are merely an indication of what information is used to construct the summary.

Technical Requirements

No data leaking. 1

- Access restricted data must not be available to those who are not authorised to access it.

Privacy-preserving summary creation. 2

- It must be possible to add values to summaries by access key and file URI.

Summary combinations. 3

- It must be possible to combine two summaries, where the combined summary is identical to a summary where all of the entries were added directly.

Authorised membership checking. 4

- Probabilistic membership checking must be possible for a given value, access key and file URI. False positives are allowed, but true negatives are required.

Query Execution with Access control. 5

- It must be possible for the pod to limit query results based on a set of access policies.

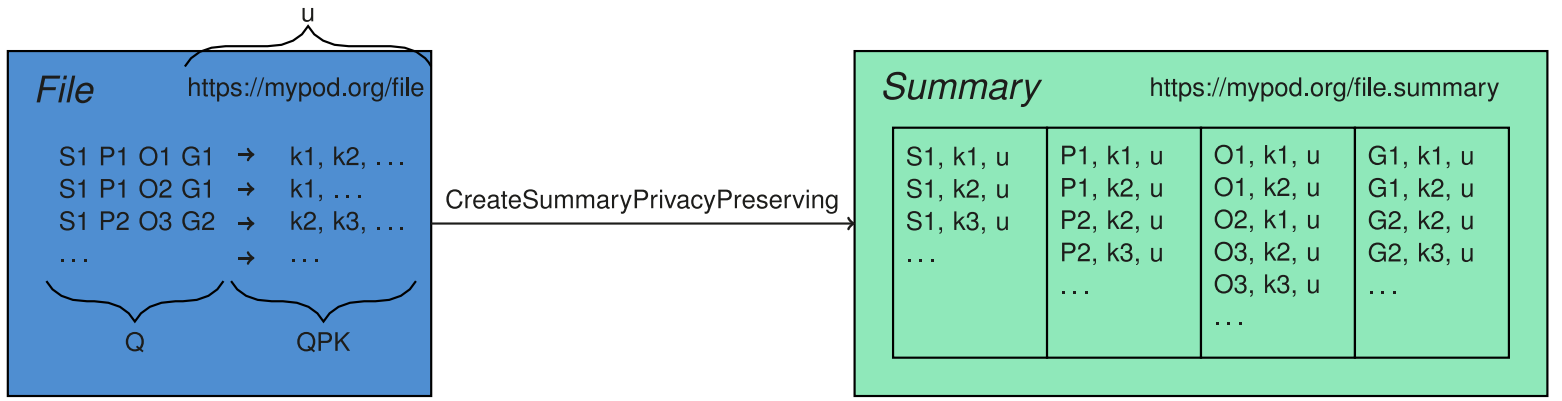
No Data Leaking: Access Key Creation Algorithm

```
FUNCTION CreateAccessKeys(Q, P)
  INPUT:
    Q: set of quads, P: set of policies
  OUTPUT:
    QPK: hashmap of quads to policies and keys
QPK = new Map()
FOREACH q in Q
  FOREACH p in P
    k = GenerateKey(q,p)
    QPK = AddKey(QPK, q, p, k)
RETURN QPK
```

- **Assumption**: pod owners already have a set of access control policies that govern access to quads stored in theirs pods.
- **Task**: Map access keys to quads based on existing access policies

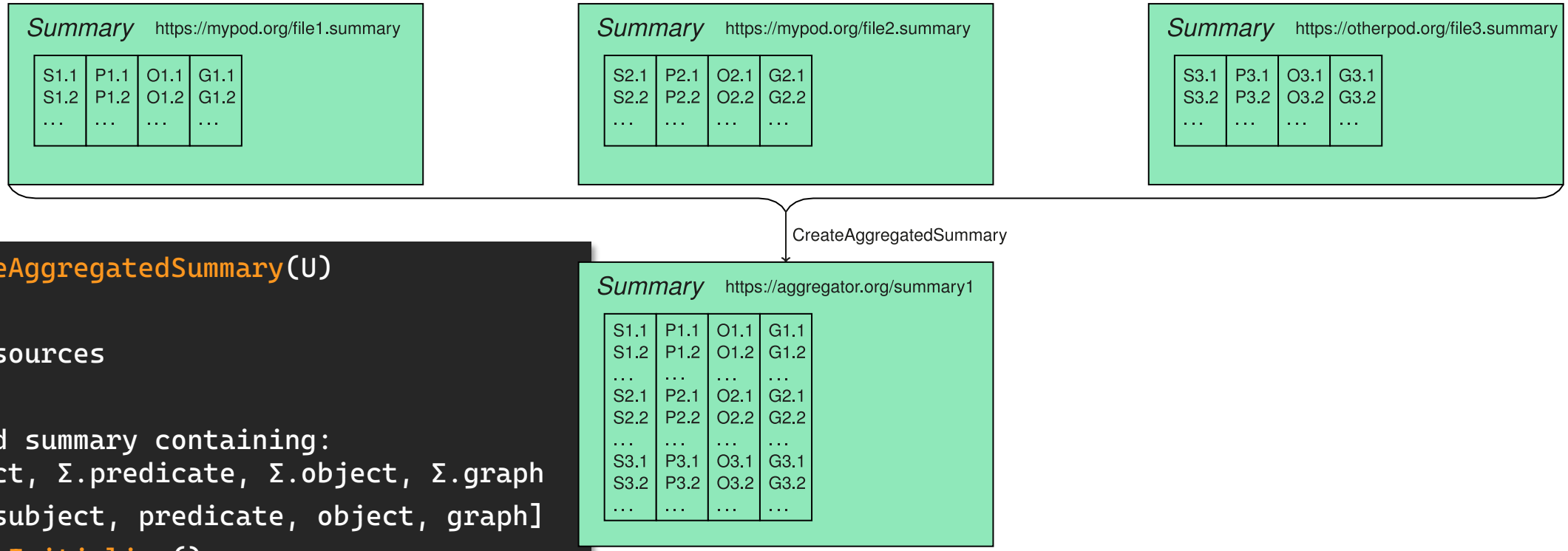
- **Many-to-many mapping** between
  - quads and policies
- **One-to-one mapping** between
  - access policies that are used for policy enforcement at query time, and access keys that are used to create privacy-preserving summaries that are needed to optimise federated querying.

Secure Access: Creating privacy-preserving summaries



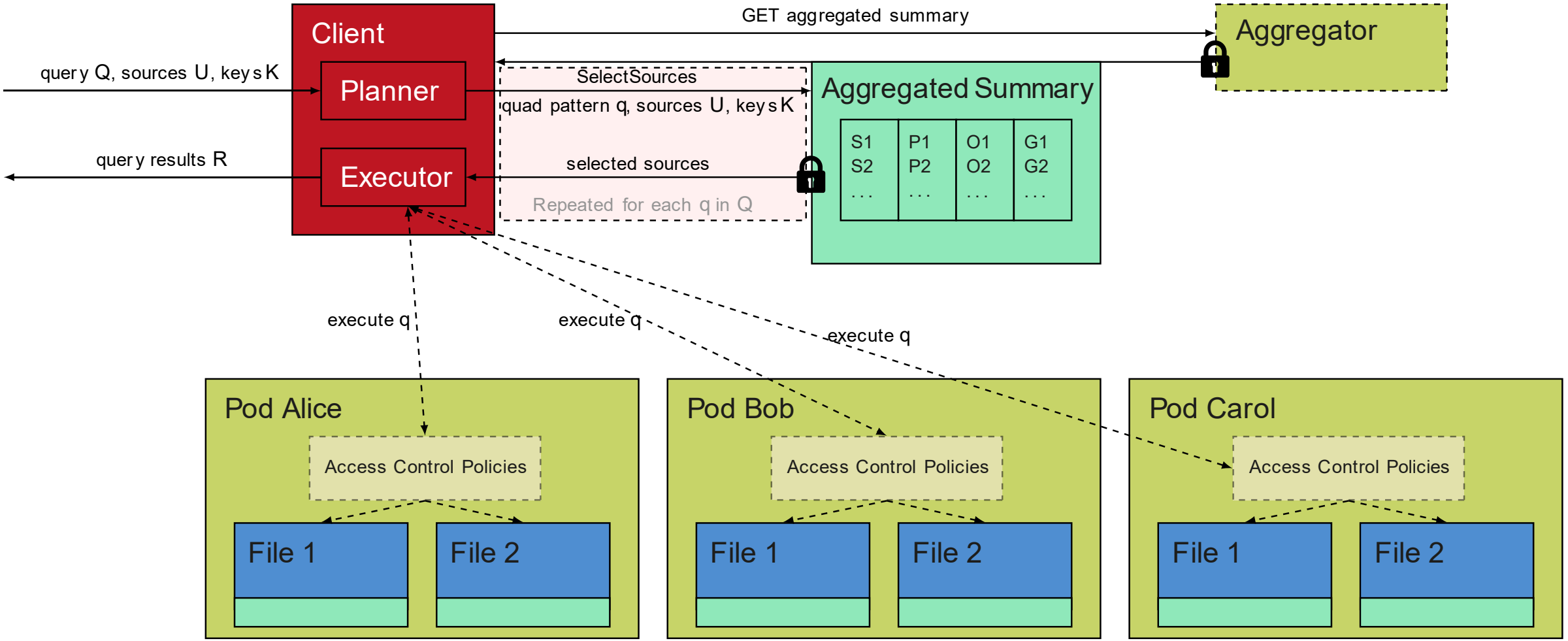

```
FUNCTION CreateSummaryPrivacyPreserving (Q, u, QPK)
  INPUT:
    Q: set of quads, u: URI of the file, QPK: hashmap relating quads to policies and keys
  OUTPUT:
    Σ: summary containing: Σ.subject, Σ.predicate, Σ.object, Σ.graph
FOREACH c in [subject, predicate, object, graph]
  Σ.c = SummaryInitialize()
FOREACH q in Q
  k = QPK(q).k
  FOREACH c in [subject, predicate, object, graph]
    Σ.c = SummaryAdd(Σ.c, q.c, k, u)
RETURN Σ
```

Summary Combinations: Combining summaries to reduce workload

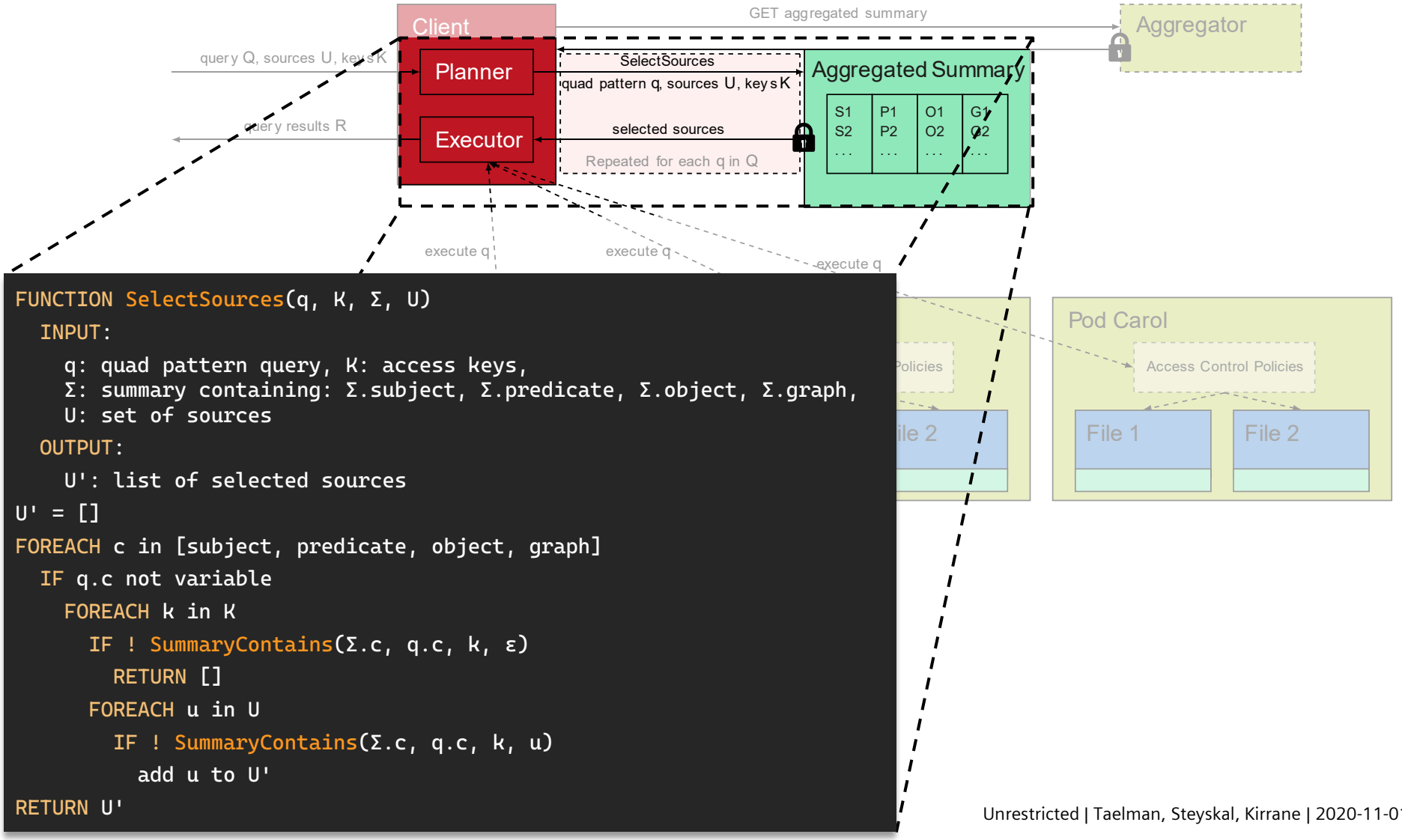



```
FUNCTION CreateAggregatedSummary(U)
  INPUT:
    U: set of sources
  OUTPUT:
    Σ: combined summary containing:
       Σ.subject, Σ.predicate, Σ.object, Σ.graph
FOREACH c in [subject, predicate, object, graph]
  Σ.c = SummaryInitialize()
FOREACH u in U
  Σ' = get summaries from u
  FOREACH c in [subject, predicate, object, graph]
    Σ.c = SummaryCombine(Σ.c, Σ'.c)
RETURN Σ
```

Big Picture: Query Execution over Privacy-Preserving Summaries

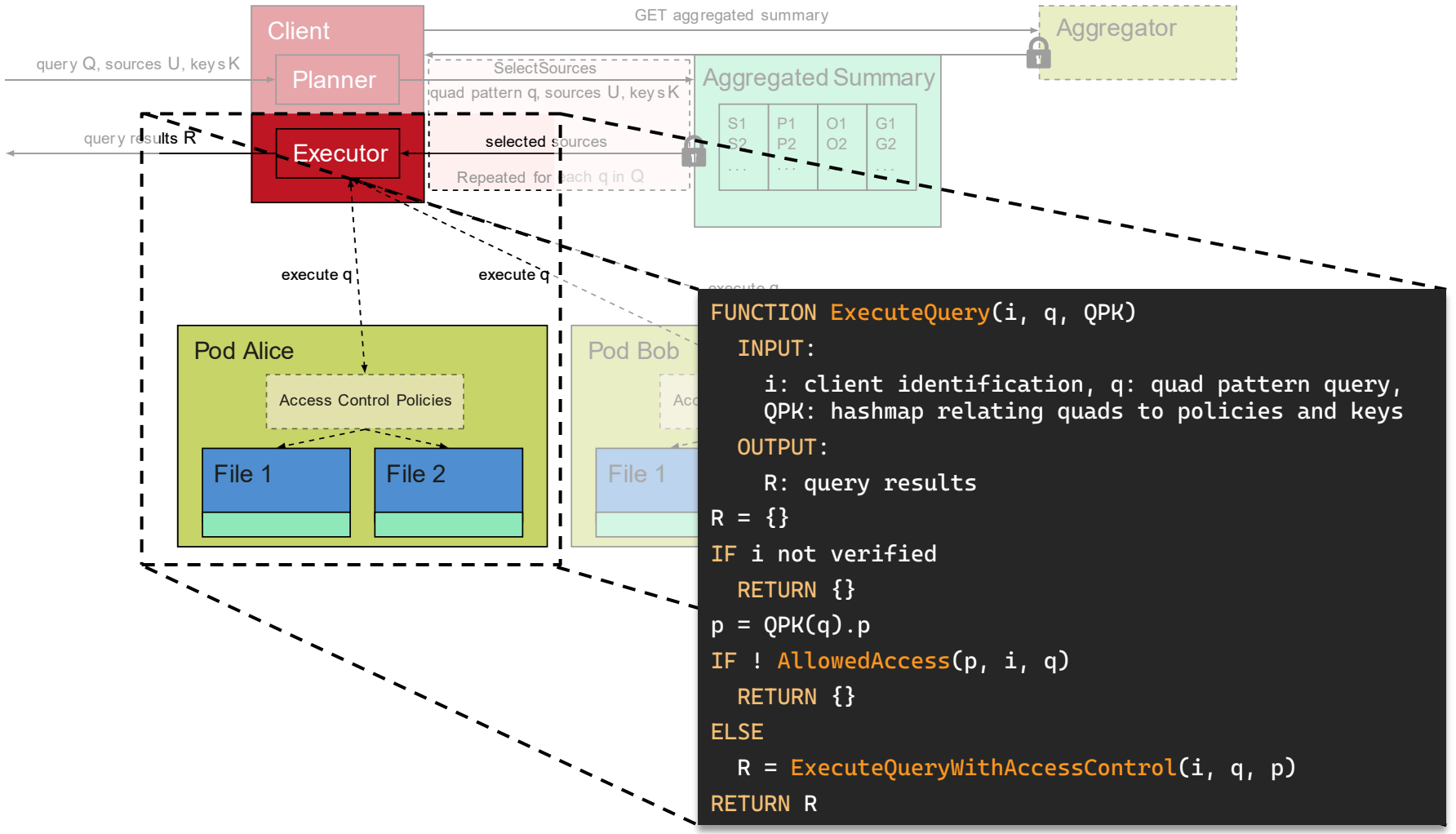


Authorised Membership Checking: Query Execution over Privacy-Preserving Summaries




```
FUNCTION SelectSources(q, K, Σ, U)
  INPUT:
    q: quad pattern query, K: access keys,
    Σ: summary containing: Σ.subject, Σ.predicate, Σ.object, Σ.graph,
    U: set of sources
  OUTPUT:
    U': list of selected sources
U' = []
FOREACH c in [subject, predicate, object, graph]
  IF q.c not variable
    FOREACH k in K
      IF ! SummaryContains(Σ.c, q.c, k, ε)
        RETURN []
      FOREACH u in U
        IF ! SummaryContains(Σ.c, q.c, k, u)
          add u to U'
RETURN U'
```

Query Execution with Access Control: Query Execution over Privacy-Preserving Summaries




```
FUNCTION ExecuteQuery(i, q, QPK)
  INPUT:
    i: client identification, q: quad pattern query,
    QPK: hashmap relating quads to policies and keys
  OUTPUT:
    R: query results
R = {}
IF i not verified
  RETURN {}
p = QPK(q).p
IF ! AllowedAccess(p, i, q)
  RETURN {}
ELSE
  R = ExecuteQueryWithAccessControl(i, q, p)
RETURN R
```

Challenges and Opportunities

Access Policy Specification

- We assume that pod owners need to be able to specify access control policies that can be enforced both from an indexing and also a query processing perspective
- Our initial proposal makes use of simple symmetric keys, but more complex scenarios both attribute-based encryption and/or key derivation algorithms possible
- When to comes to policy management, there is a need to ensure that
    i. access keys are tightly bound to access policies,
    ii. said keys are distributed to authorised individuals

Summary Generation and Maintenance

- The requirements for enabling federated querying in an efficient manner through privacy-preserving aggregators are mainly driven by the summarisation technology.
- In this context symmetric keys are used to create privacy-preserving summaries that do not leak access restricted data.
- We consider AMFs, such as Bloom filters, as being one possible candidate for such summaries that meet the **privacy-preserving summary creation** and **summary combinations** requirements.

Challenges and Opportunities

Source Selection

- In the proposed framework, a client-side query engine can make use of the aggregator's summary to perform source selection, in order to reduce the number of sources that are being consulted by this engine.

- From a source selection perspective, we address the **authorised membership checking** requirement.
  - summaries allow source selection based on quad patterns instead of full SPARQL queries,

  => source selection can be pushed down into the query plan, which allows quad patterns in the query to be executed over a different range of sources.

- A hybrid approach where source selection happens both before and during query execution could be investigated.

Query Execution with Access control

- Once the query engine has identified the data sources that could potentially contribute results to their query, the query engine needs to authenticate the user to the server(s) and execute the query or parts thereof.

- The server is responsible for enforcing access control, and executing the query or parts thereof.
  - **query execution with access control** requirements.

- Enforcement of authorisations (i.e., policies) which govern ***who* can do *what* with *which* resources under *what* conditions.**

- We envision a mechanism that translates access policies (i.e. sets of authorisations) into constraints (e.g., data shapes like SHACL) which requests and respective query results can then be validated against.

Q&A

SIEMENS

Contact

Simon Steyskal
Research Scientist
Siemens AG / T RDA BAM CON-AT
Austria

E-mail simon.steyskal@siemens.com

Ruben Taelman
Postdoctoral Researcher
UGent / IDLab
Belgium

E-mail ruben.taelman@ugent.be

Sabrina Kirrane
Assistant Professor / Researcher
WU Wien / Institute for InfoSys & New Media
Austria

E-mail sabrina.kirrane@wu.ac.at