Structural Cloud Audits that Protect Private Information

Hongda Xiao, Bryan Ford, Joan Feigenbaum
Department of Computer Science
Yale University

Cloud Computing Security Workshop – November 8, 2013

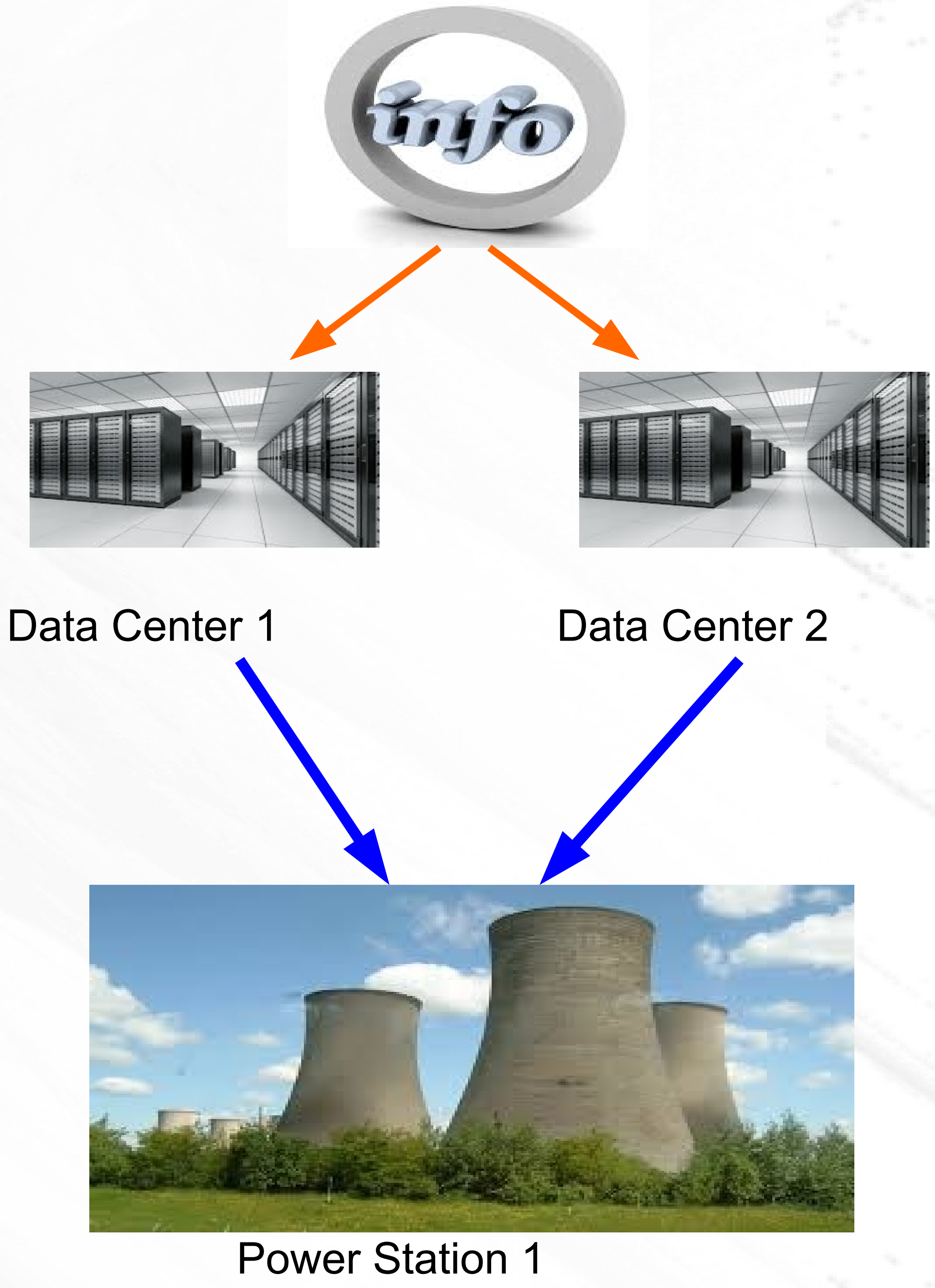
Motivation

- Cloud computing and cloud storage now plays a central role in the daily lives of individuals and businesses.
    - Over a billion people use Gmail and Facebook to create, share, and store personal data
    - 20% of all organizations use the commercially available cloud-storage services provided both by established vendors and by cloud-storage start-ups
- Reliability of cloud-service providers grows in importance.

Motivation

- Cloud-service providers use redundancy to achieve reliability
- But redundancy can fail due to Common Dependencies

[Ford, *Icebergs in the Clouds,* HotCloud '12]



Motivation

- This is a real problem
  - e.g. a lightning storm in northern Virginia took out both the main power supply and the backup generator that powered all of Amazon EC2's data centers in the region

- We need a systematic way to discover and quantify vulnerabilities resulting from common dependencies

Motivation

- Zhai et al. proposed Structural Reliability Auditing (SRA)
  - collect comprehensive information from infrastructure providers
  - construct a service-wide fault tree
  - identify critical components, estimate likelihood of service outage

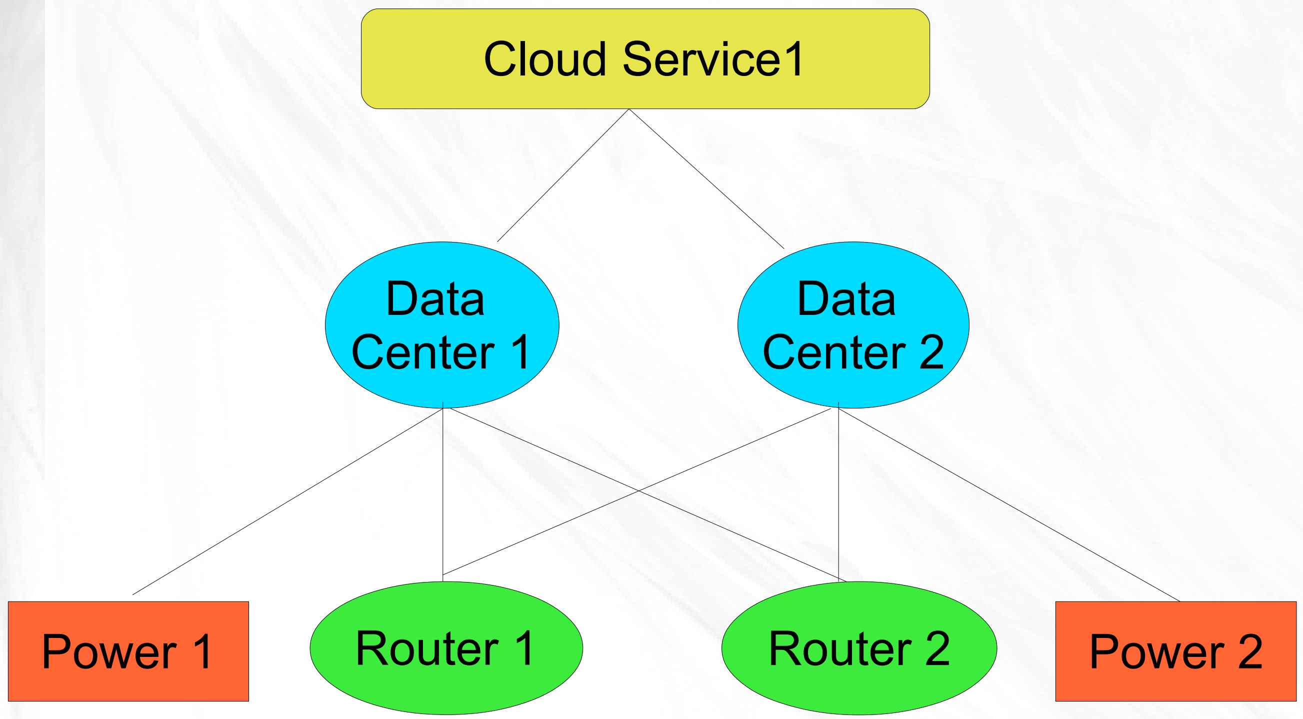
- A potential barrier to adoption of SRA is the sensitive nature of both its input and its output.
  - cloud service providers and infrastructure providers may not be willing to disclose the required information

Objective

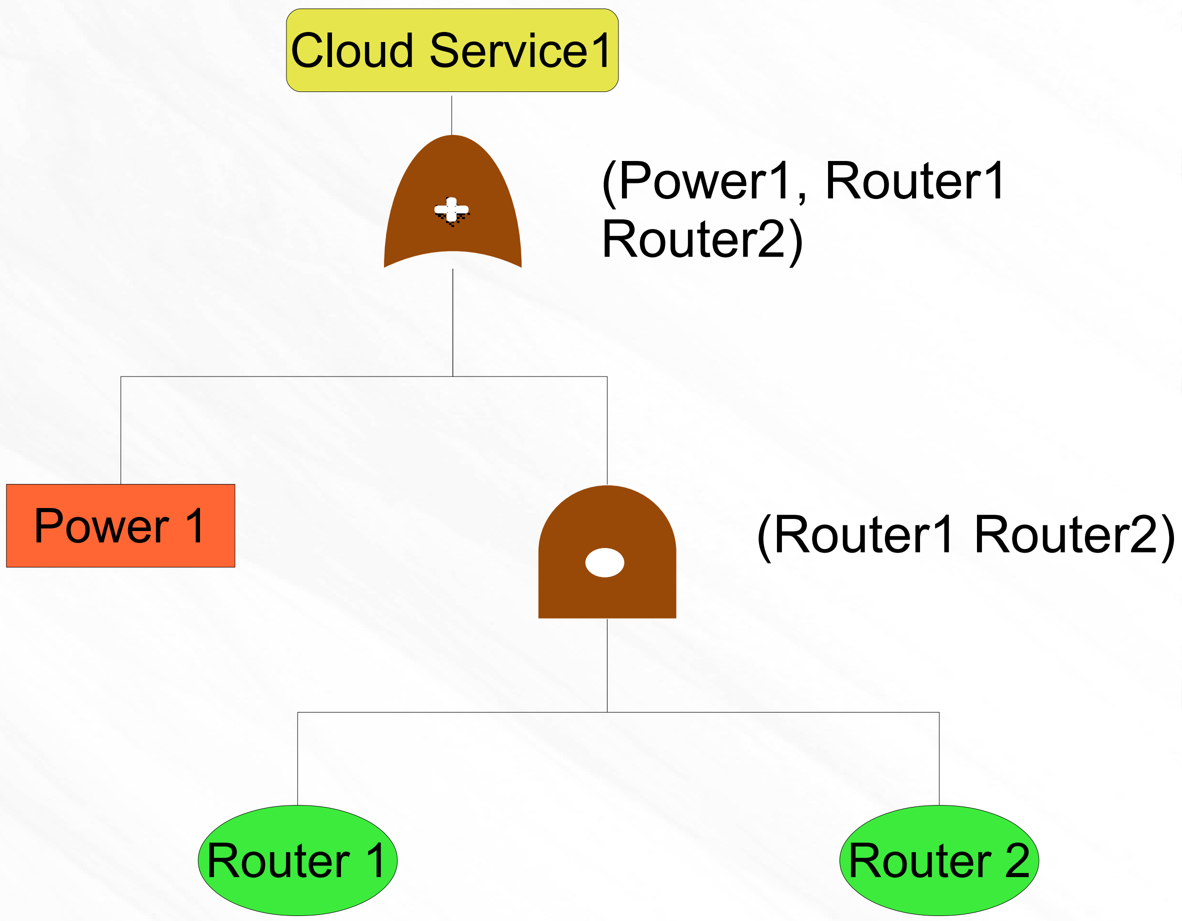
- Privacy-Preserving SRA (P-SRA): investigate the use of secure multi-party computation (SMPC) to perform SRA in a privacy preserving manner
  - Perform SMPC on complex, linked data structures of cloud topology, which has not often been explored yet

Basic Idea

Step 1: Build a *structural model* of cloud infrastructure of interest



Step 2: Perform *fault tree analysis* to detect hidden failure risks



[Zhai et al., *Auditing the Structural Reliability of the Clouds*, Yale TR-1479]

Challenges

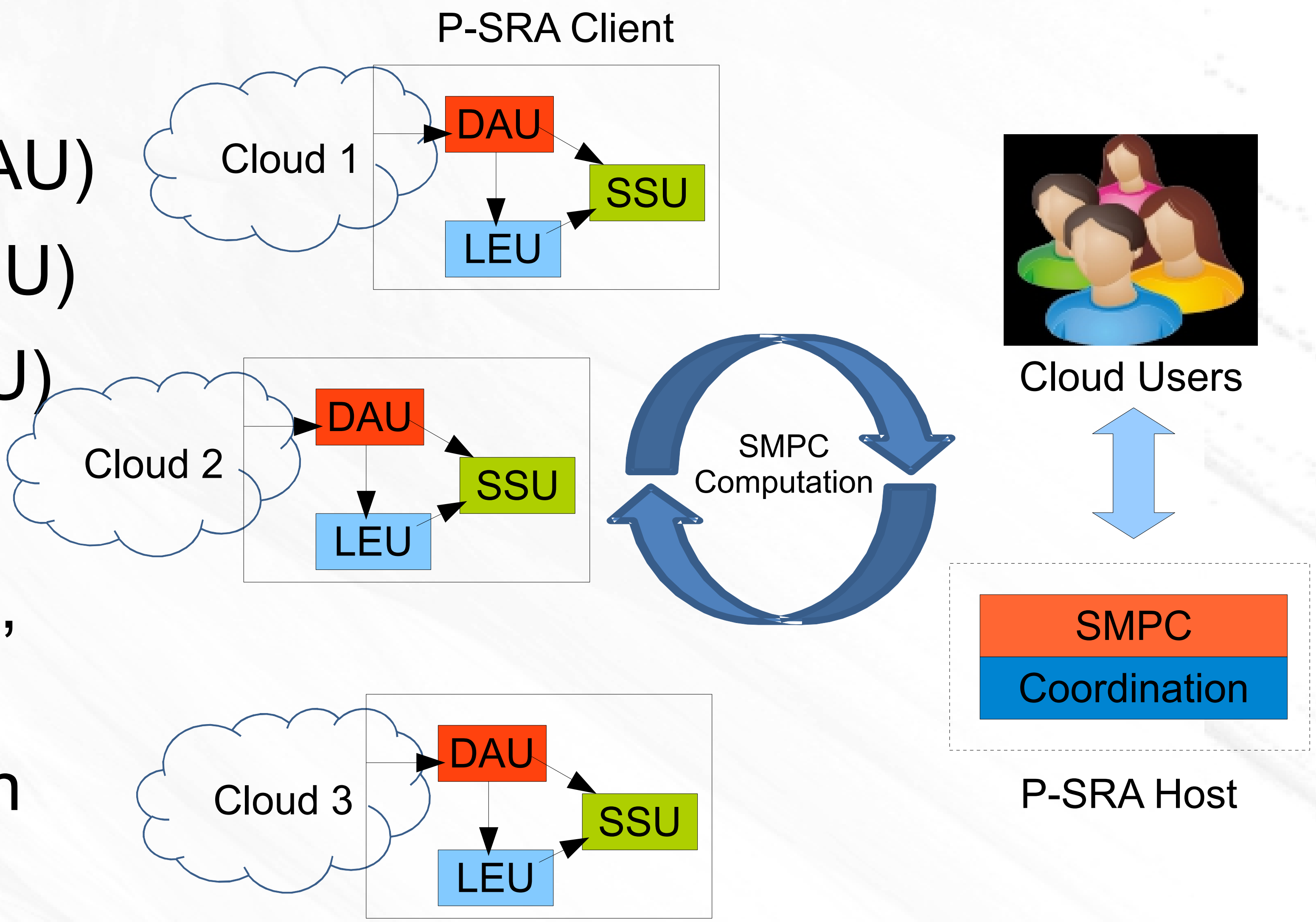
- Private Data Acquisition
  - How to collect complex, linked data of cloud topology without compromising the privacy of the cloud and infrastructure providers?
- Privacy-Preserving Analysis
  - How to identify common dependencies and correlated failure risk without requiring providers to disclose confidential information?
- Efficiency
  - SMPC is NOT very efficient especially when the size of inputs are large

Our Solutions

- Private Data Acquisition
  - Leverage secret sharing techniques in SMPC
  - Specify valid output protecting privacy
- Privacy-Preserving Analysis
  - Specialized graph representation techniques to build fault tree in a privacy preserving manner
- Efficiency
  - Novel data partitioning techniques to effectively reduce the input size of SMPC and leave most of the computations locally

System Design Overview

- P-SRA Client
  - Data Acquisition Unit (DAU)
  - Local Execution Unit (LEU)
  - Secret Sharing Unit (SSU)
- P-SRA Host
  - Represents Cloud Users, Reliability Auditors
  - Does SMPC coordination



Cloud Provider

- Install and control a P-SRA Client
- Input their private infrastructure information, which is considered private
- Semi-honest Threat Model
    - The Cloud Providers are honest but curious

P-SRA Client

- Fully controlled by Cloud Providers
- Data Acquisition Unit
    - Collects component and dependency information
- Local Execution Unit
    - Perform local structural reliability analysis
- Secret Sharing Unit
    - Perform SMPC with P-SRA Host

P-SRA Host

- SMPC module
  - Perform SMPC with each P-SRA client installed by cloud providers
- Coordination module
  - Coordinate the communication between P-SRA Clients and P-SRA Host
- Semi-honest Model
  - The P-SRA Host is honest but curious

Outline of How the System Works

- Step 1: Privacy-preserving dependency acquisition
- Step 2: Subgraph abstraction to reduce problem size
- Step 3: SMPC protocol execution and local computation
- Step 4: Privacy-preserving output delivery

Privacy-preserving dependency acquisition

- The DAU of each cloud-service provider collects information about the components and dependencies of this provider
    - network dependencies
    - hardware dependencies
    - software dependencies
    - failure probability estimates for components
- Store the information in a local database for use by P-SRA's other modules.

Subgraph Abstraction

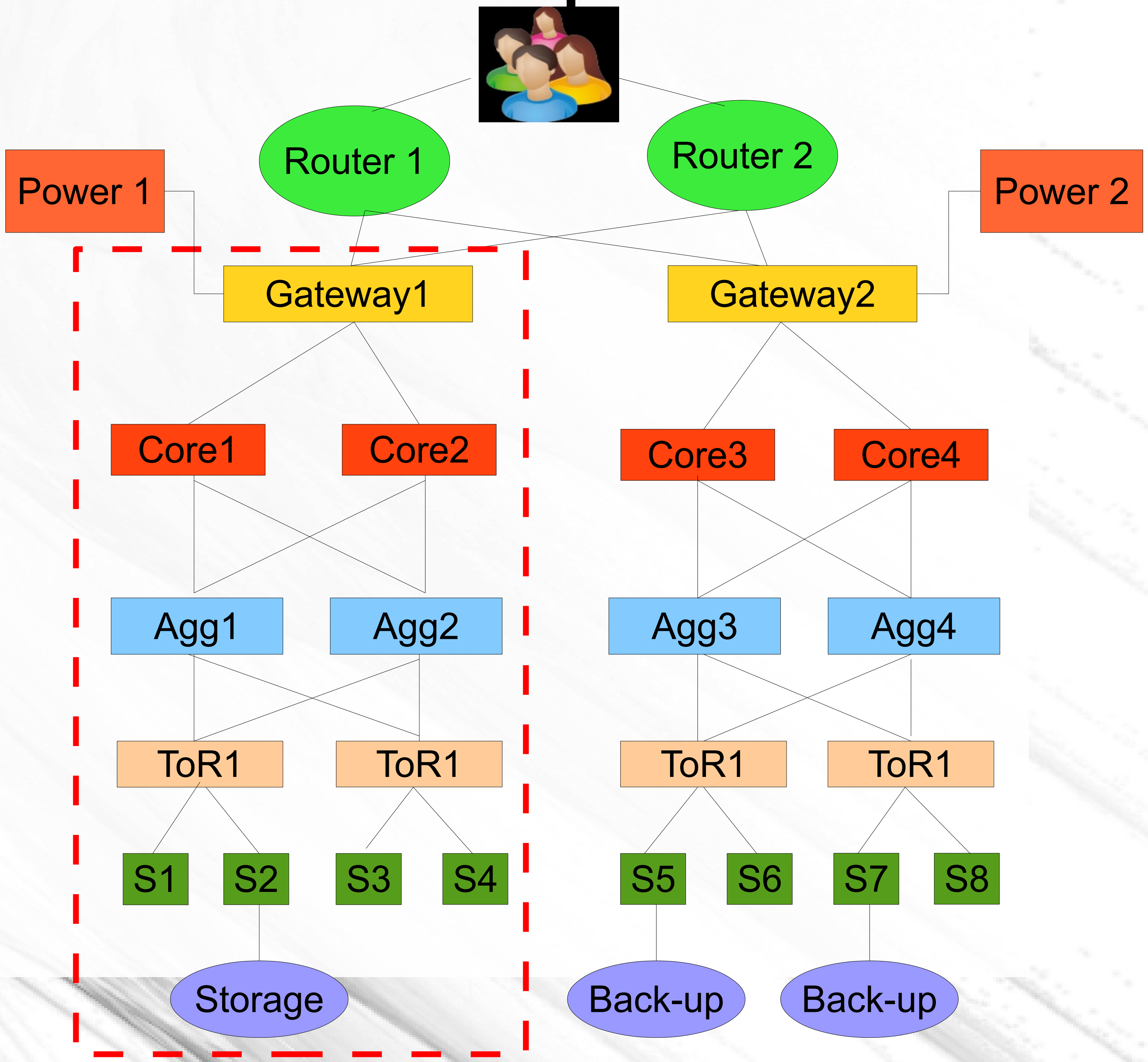
- The Client's SSU abstracts the dependency information of private components as a set of macro-components, which are the actual inputs of the SMPC
- Key step to reduce the input size of SMPC
- The choice of abstraction policy is flexible as long as satisfying the proper criterions
- Can be generalized to other SMPC problem on complex and linked data structure

Subgraph Abstraction Policy

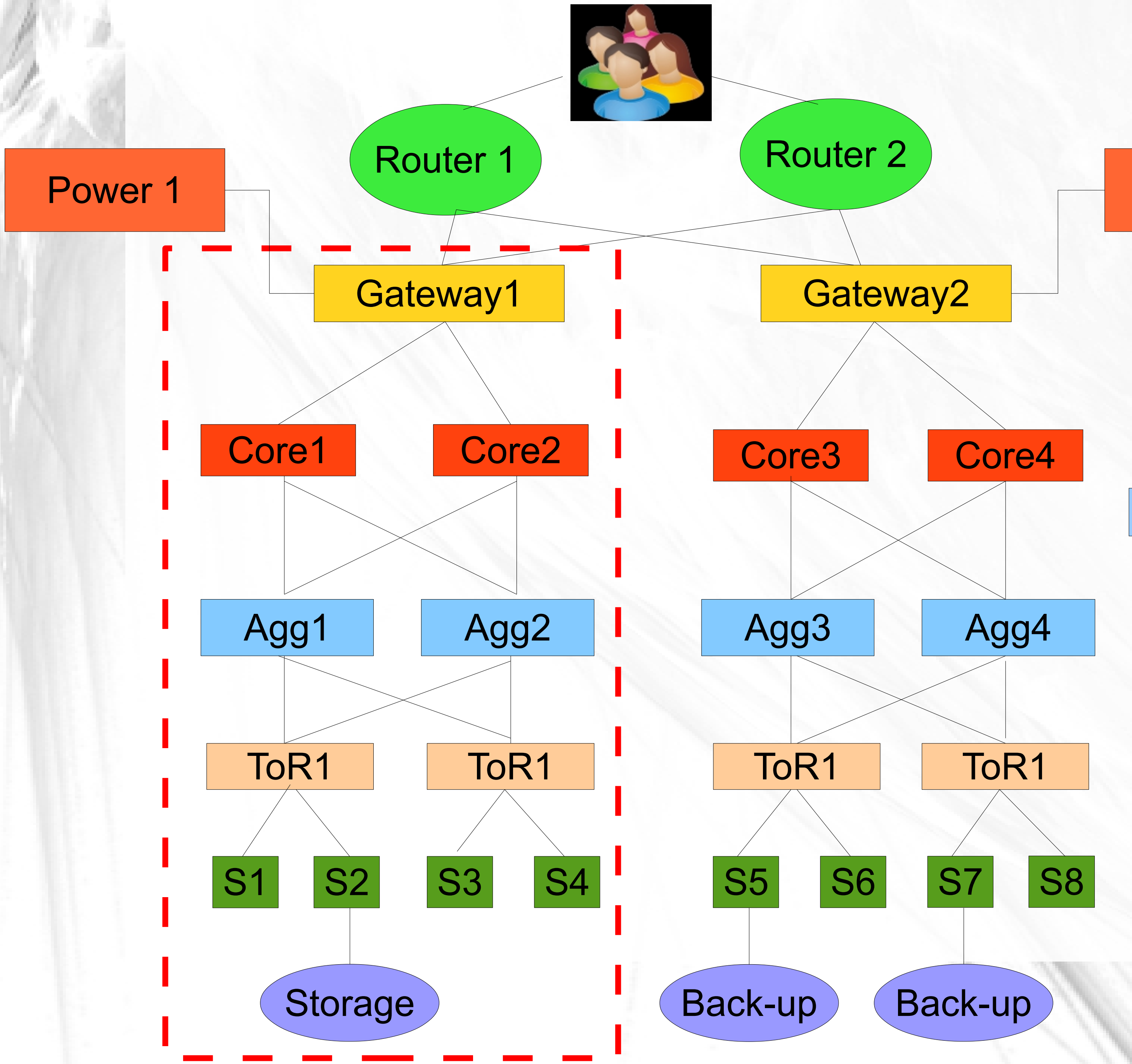
- A subgraph H of the full dependency graph G of a cloud-service provider S should have two properties in order to be eligible for abstraction as a macro-component
    - all components in H must be used only by S
    - for any two components v and w in H, the dependency information of v with respect to components outside of H is identical to that of w
- SSU collapses H to a single node to transfer G to a smaller graph G'

Subgraph Abstraction: Example

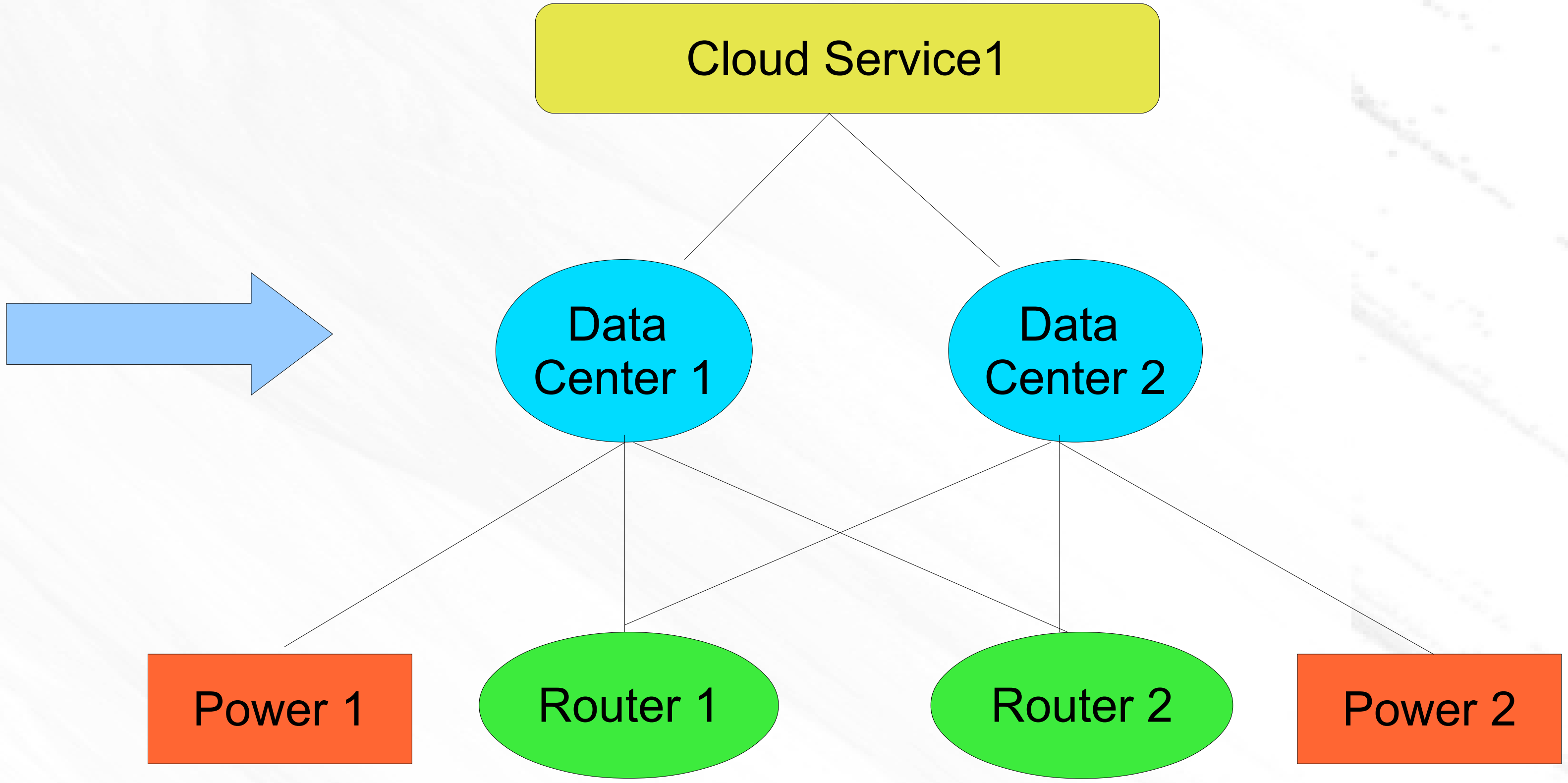
- Dependency Graph of a Simple Data Center
  - A Storage Service
  - Two Data Centers, one for service and the other for back-up
- Red Frame is the data center 1, which satisfies the two properties



Subgraph Abstraction: Example



Red frame on the left is data center 1, which is abstracted as Data Center 1 on the right



SMPC and Local Computation

- SMPC
  - Perform SMPC to identify common dependency and reliability analysis across cloud providers
  - SSUs of P-SRA Clients work with SMPC of P-SRA Host
- Local Computation
  - SSU passes the dependency informaiton within macro-components to LEU
  - LEU performs structural reliability analysis locally

SMPC Protocol

- Fault-tree construction
- Generate input for the SMPC
- Identify common dependencies
- Calculate failure sets

Fault Tree Analysis

- FTA is a deductive reasoning technique
    - Occurence of top event is a boolean combination of occurence of lower level events
- Fault Tree is a Directed Acyclic Graph (DAG)
    - Node: gate or event
    - Link: dependency information
- Failure Set is a set of components whose simultaneous failure results in cloud service outage

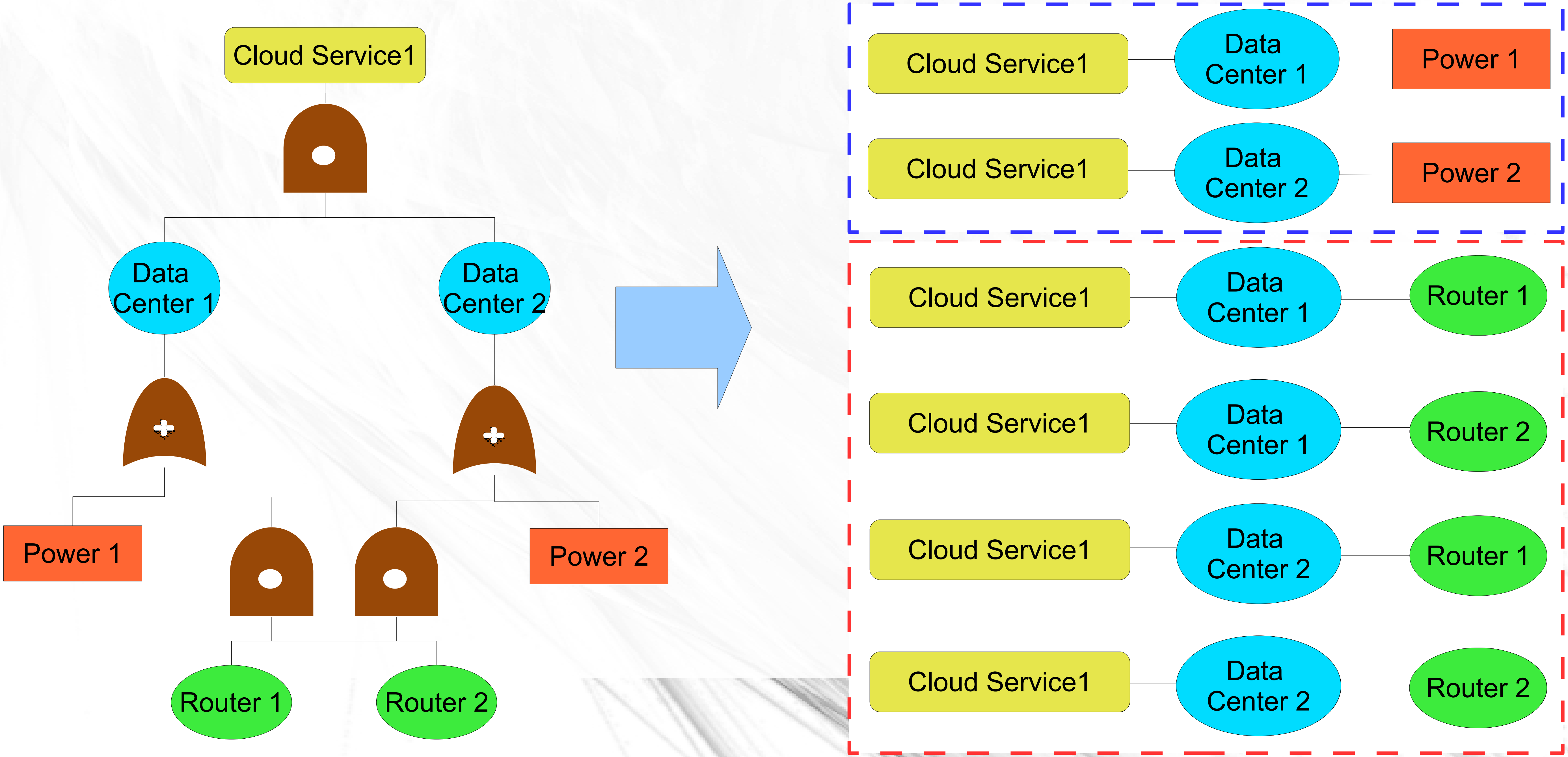
SMPC Fault Tree Construction

- Challenge
    - SMPC cannot readily handle conditionals, which are necessary in traditional ways of processing Fault Trees
- Solution
    - Rewrite the fault tree as topology paths form with types
    - Eliminates use of conditionals

Topology Paths with Types

- Extract all paths through dependency DAG
  - root node → intermediate nodes → leaf node
  - Unpacks the DAG for "circuit" processing
  - Can be exponentially larger than DAG in worst case :(
- Types of topology paths
  - The SSU builds a disjunction of conjunctions of disjunctions data structure by assigning each path a type

Topology Paths with Types: Example



Local Execution Protocol

- Generate fault tree for components within macro-components
- Compute the failure sets of each macro-component

Generate input for the SMPC

- SSUs pad the fault tree in order to avoid leaking structural informatoin such as the size of the cloud infrastructure
  - Add dummy nodes with zero ID into each topology path
  - Add zero paths into the fault tree with randomly assigned types
  - Zero ID nodes do not affect the result

Identify common dependencies

- SSUs and P-SRA Host cooperate to identify common dependency
    - doing multiple (privacy-preserving) set intersections, followed by one (privacy-preserving) union
- Strict security requires doing it without conditional statements
    - Transfer conditional statements into arithmetic computation

Identify common dependencies

Algorithm 1: Common-Dependency Finder

Input: Fault tree T_i, $i = 1\ to\ N$, where N is the number of participating cloud-service providers

Output: Common Dependency

foreach $T_I\ and\ T_J, I \neq J$ **do**

    private mask.clear();

    foreach $node_i \in T_I\ and\ node_j \in T_J$ **do**

        $private\ mask[i][j] = (node_i.ID == node_j.ID)$;

    private CommonDep.clear();

    foreach $node_i \in T_i\ and\ node_j \in T_j$ **do**

        $private\ CommonDep[i] = mask[i][j] \times node_j.ID + CommonDep[i]$;

    private CommonDependent.append(CommonDep);

return private CommonDependent;

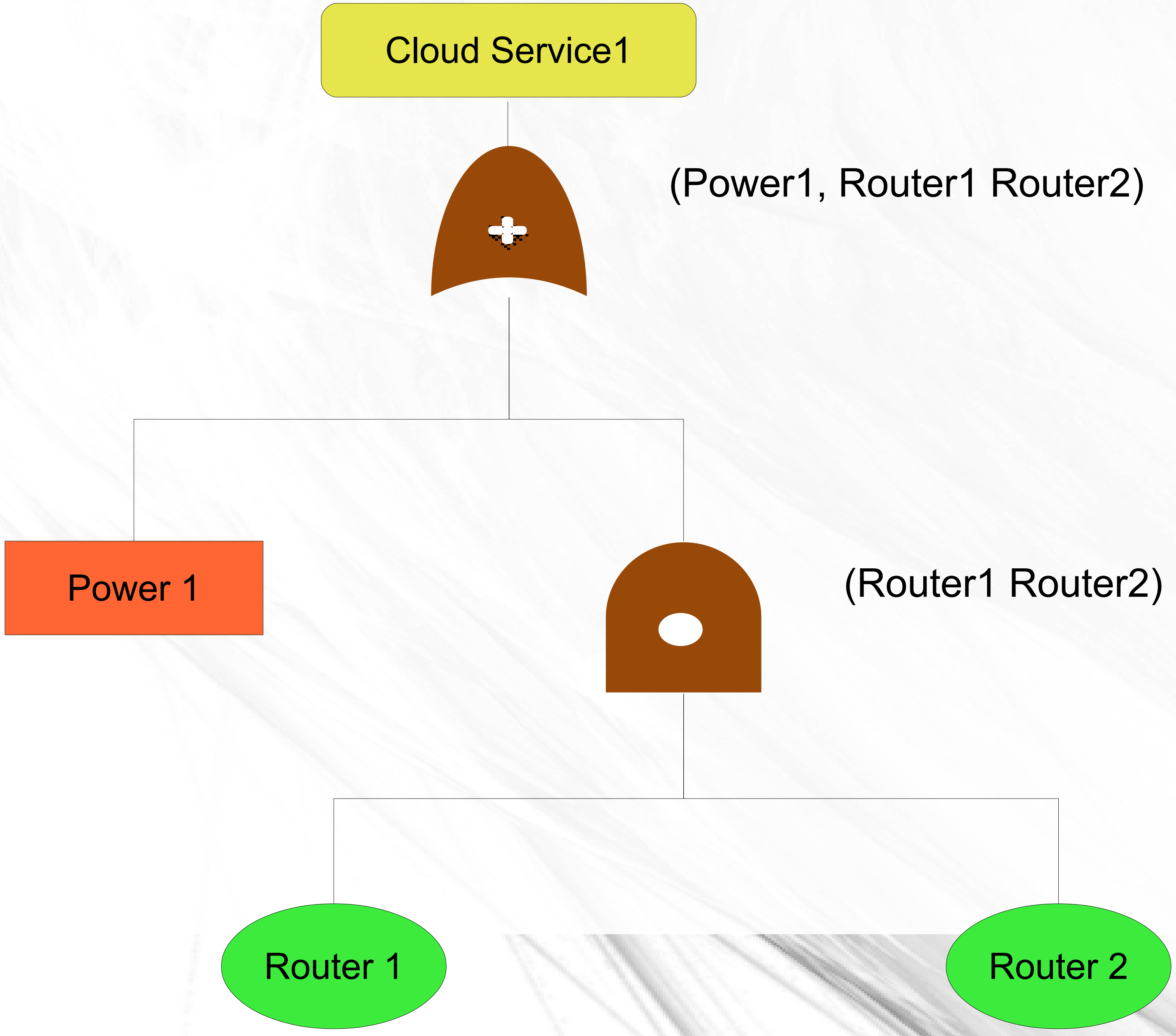
Privacy Preserving Fault Tree Analysis: Calculate failure sets

- Minimal FSes algorithm
    - Find minimal FSes
    - Exponential complexity
- Heuristic failure-sampling algorithm
    - Faster
    - Not necessarily the minimal FSes

Minimal FSes Algorithm

- The algorithm traverses the Fault Tree
- Basic events generate FSes containing only themselves, while non-basic events produce FSes based on the FSes of their child events and their gate types.
- For an OR gate, any FS of one of the input nodes is an FS of the OR.
- For an AND gate, take cartesian product of the sets of FSes of the input nodes then combine each element of the cartesian product into a single FS by taking a union.

Minimal FSes Algorithm: Example



Minimal FSes Algorithm

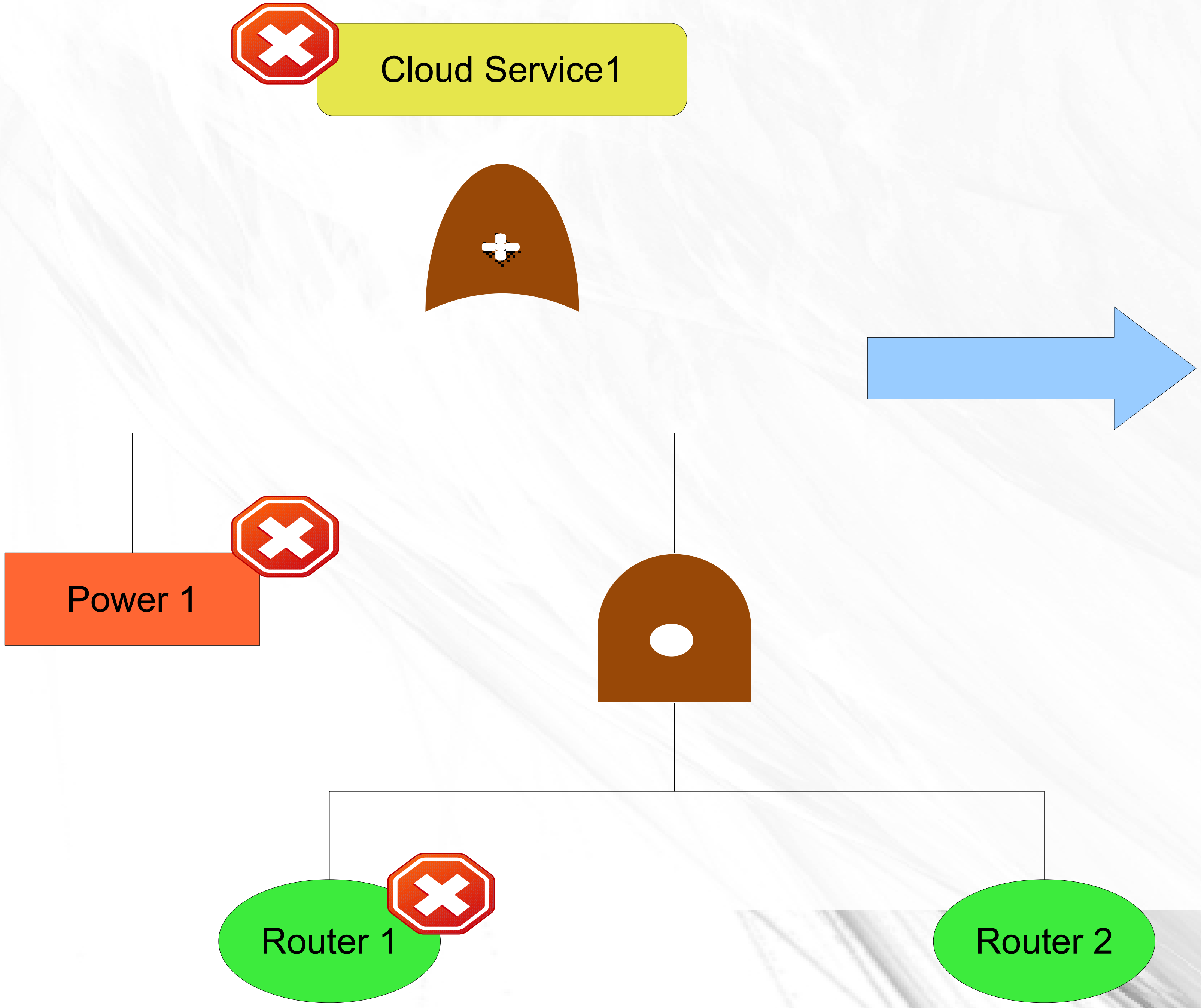
Algorithm 2: Minimal-FS algorithm

```
Input: Global Fault tree T
Output: MinimalFS
foreach private path_i ∈ T do
    foreach private node_j ∈ private path_i do
        private path_i.FS.apped(node_j);
        /* each path corresponds to an OR gate with
           input as the nodes along the path */
foreach AndGate_i ∈ T do
    AndGate_i.FS.clear();
    foreach path_j ∈ AndGate_i do
        AndGate_i.FS ← AndGate_i.FS × path_j.FS;
        /* process the AndGate for each type of
           topology paths */
        /* FS of AndGate_i is the Cartesian Product of
           AndGate_i.FS and path_j.FS. */
private minimalFS.clear();
foreach AndGate_i ∈ T do
    minimalFS.append(AndGate_i.FS);
    /* process the OR gate connecting to the And
       Gates */
/* reduce redundant items in minimumFS and assign the
   result to minimalFS, and then simplify minimalFS.
   */
minimalFS ← reduce_redundancy(minimalFS);
minimalFS ← simplify(minimalFS);
return minimalFS;
```

Failure Sampling Algorithm

- Randomly assigns fail or no fail to the basic events of the Fault Tree
- Compute whether the top event fails
- If the top event fails, the failed basic events consist of a FS

Failure Sampling Algorithm: Example



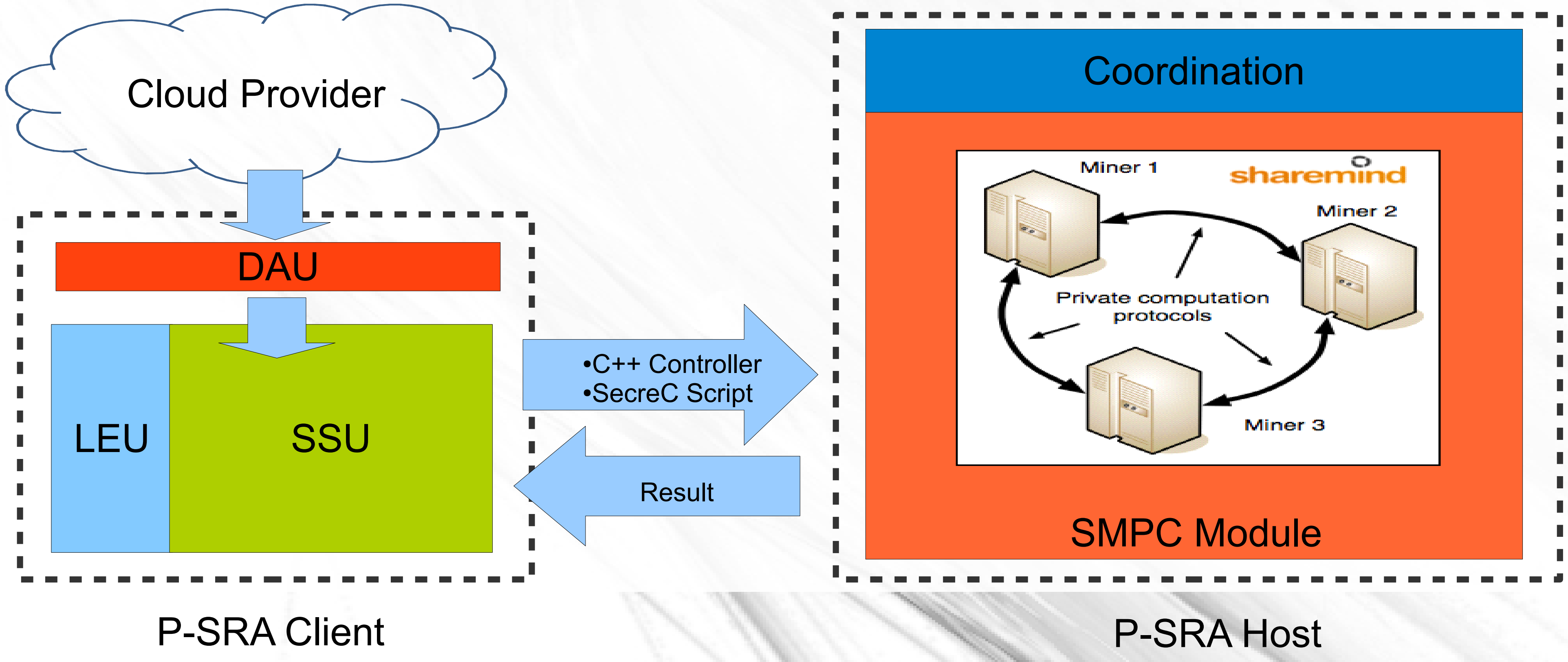
(Power1Router1) is a failure set, but not minimal

Privacy-preserving Output Delivery

- Output for Cloud-Service Providers
    - Common dependency
    - Partial failure sets
- Output for Cloud-Service Users
    - Common-dependency ratio
    - Overall failure probabilities of cloud services
    - Top-ranked failure sets

Implementation

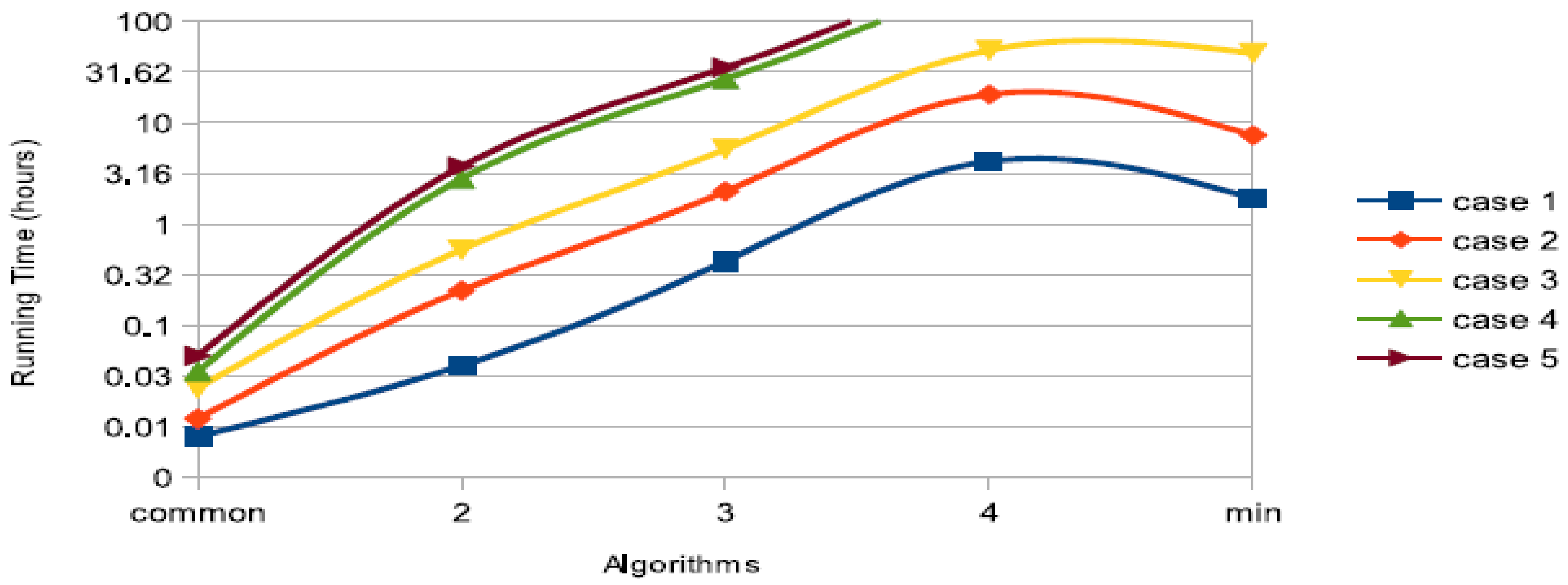
- Sharemind SecreC
  - C-like SMPC programming language
  - Specialized assembly to execute the code



Simulation: SMPC

	Case 1	Case 2	Case 3	Case 4	Case 5
# of cloud providers	2	2	3	3	2
# of data center	1	3	8	10	3
# of internet router	3	5	10	15	5
# of power stations	1	2	3	5	2
ratio of common dep.	0.8	0.2	0.2	0.2	0.2
ratio of padding	0.0	0.0	0.0	0.0	0.5

Table 1: Configuration of Test Data Sets



Simulation: Local Execution

Table 2: Performance of the LEU of a P-SRA client

Configuration	Case 1	Case 2	Case 3	Case 4	Case 5
# of switch ports	4	8	16	24	48
# of core routers	4	16	64	144	576
# of agg switches	8	32	128	288	1152
# of ToR switches	8	32	128	288	1152
# of servers	16	128	1024	3456	13824
Total # of components	40	216	1360	4200	16752
Running time (minutes)					
FS round 10^3	< 0.7	< 0.7	< 0.7	< 0.7	< 0.7
FS round 10^4	0.7	0.7	1.7	2.3	6.9
FS round 10^5	0.8	0.9	5.3	28.1	6.9
FS round 10^6	1.7	4.5	65.0	243.5	462.9
FS round 10^7	28.3	56.6	512.1	NA	NA
Minimal FS	0.8	14.8	309.7	NA	NA

Conclusion

- We designed P-SRA, a private, structural-reliability auditor for cloud services based on SMPC, and prototyped it using the Sharemind SecreC platform
- We explored the use of data partitioning and subgraph abstraction SMPC on large graphs, with promising results.
- Our preliminary experiments indicate that P-SRA could be a practical, off-line service, at least for small-scale cloud services or for ones that permit significant subgraph abstraction.

Thank you
Any Questions?