SEAL: Capability-Based Access Control for Data-Analytic Scenarios

Hamed Rasifard, Rahul Gopinath, Michael Backes, Hamed Nemati | 28th ACM Symposium on Access Control Models and Technologies | June 7-9

✉️ hamed.rasifard@cispa.de
Trust Issues in Big-Data Sharing: Data Owners vs. Data Analysts
Trust Issues in Big-Data Sharing: Data Owners vs. Data Analysts

- Big-data era

High Volume

High Velocity

High Variety
Trust Issues in Big-Data Sharing: Data Owners vs. Data Analysts

- Big-data era

High Volume
- Data owners collaborate with data analysts to extract data-driven insights

High Velocity

High Variety
Trust Issues in Big-Data Sharing: Data Owners vs. Data Analysts

• Big-data era

High Volume
• Data owners collaborate with data analysts to extract data-driven insights

High Velocity

High Variety

• Data-sharing concerns
  - Data owners: data privacy and security
  - Data analysts: data quality and reliability
Data Sharing: Privacy Challenges
Data Sharing: Privacy Challenges

Data utility

VS

Privacy
Data Sharing: Privacy Challenges

- Data utility
- Privacy
- Scalability
- Performance
Data Sharing: Privacy Challenges

- Data utility
- Privacy
- Scalability
- Performance
- Regulatory compliance
Data Sharing: Privacy Challenges

- Data utility
- Privacy
- Scalability
- Performance

- Regulatory compliance
- Lack of data-owner control over data usage
Data Sharing: Privacy Challenges

- Data utility
- Privacy
- Scalability
- Performance

- Regulatory compliance
- Lack of data-owner control over data usage
- Emerging Threats and Attacks
Data Sharing: Access-Control Challenges
Data Sharing: Access-Control Challenges

Fine-grained access control
Data Sharing: Access-Control Challenges

- Fine-grained access control
- Dynamic data access
Data Sharing: Access-Control Challenges

- Fine-grained access control
- Dynamic data access
- Data context and granularity
Data Sharing: Access-Control Challenges

- Fine-grained access control
- Dynamic data access
- Data context and granularity
- Integrating access-control systems with privacy-preserving techniques
Our Solution: Bringing Computation to Data
Our Solution: Bringing Computation to Data

Data security
Our Solution: Bringing Computation to Data

- Data security
- Data privacy
Our Solution: Bringing Computation to Data

- Data security
- Data privacy
- Scalability and Efficiency
Our Solution: Bringing Computation to Data

Data security
Data privacy
Scalability and Efficiency

Required network bandwidth
Our Solution: Bringing Computation to Data

- **Data security**
- **Data privacy**
- **Scalability and Efficiency**

- **Required network bandwidth**

- **Compliance with data governance and regulations**
Our Solution: Bringing Computation to Data

- Data security
- Data privacy
- Scalability and Efficiency

- Required network bandwidth

- Compliance with data governance and regulations

- Challenges:
  - Supporting fine-grained and dynamic access control
  - Supporting complex orders of computations
  - Maintaining data-owner control through all steps of computations
Our Solution: Bringing Computation to Data

- **Data security**
- **Data privacy**
- **Scalability and Efficiency**

- Required network bandwidth

- **Thumbs up**: Compliance with data governance and regulations

- **Challenges**:
  - Supporting fine-grained and dynamic access control
  - Supporting complex orders of computations
  - Maintaining data-owner control through all steps of computations

**SEAL: Capability-based Access-control Framework**
Capability-based Access Control
Capability-based Access Control

• Provides fined-grained access control
• Support the least-privilege principle
• A capability is an unforgeable token
• Access rights is granted based-on possessing of capabilities
Capability-based Access Control

• Provides fine-grained access control
• Support the least-privilege principle
• A capability is an unforgeable token
• Access rights is granted based-on possessing of capabilities

• Capability-Object Model*
  - Combines capabilities and objects to enforce access control
  - Objects represent system resources or entities that are protected by the capability-object model

Capability-based Access Control

- Provides fined-grained access control
- Support the least-privilege principle
- A capability is an unforgeable token
- Access rights is granted based-on possessing of capabilities

**Capability-Object Model**
- Combines capabilities and objects to enforce access control
- Objects represent system resources or entities that are protected by the capability-object model

Capability-based Access Control

- Provides finely-grained access control
- Supports the least-privilege principle
- A capability is an unforgeable token
- Access rights are granted based on possessing of capabilities

**Capability-Object Model**

- Combines **capabilities** and **objects** to enforce access control
- Objects represent system resources or entities that are protected by the capability-object model

---

Capability-based Access Control

• Provides fined-grained access control
• Support the least-privilege principle
• A capability is an unforgeable token
• Access rights is granted based-on possessing of capabilities

• Capability-Object Model*
  − Combines capabilities and objects to enforce access control
  − Objects represent system resources or entities that are protected by the capability-object model

SEAL: Capability Model

• Capability types
  - User capability ↔ Forwarding facet
  - System capability ↔ Revoking facet
SEAL: Capability Model

• Capability types
  - User capability $\equiv$ Forwarding facet
  - System capability $\equiv$ Revoking facet

• System-Capability Tree

![System-Capability Tree Image]
SEAL: Capability Model

• Capability types
  - User capability $\equiv$ Forwarding facet
  - System capability $\equiv$ Revoking facet

• System-Capability Tree
  - Tracking delegations

User Capability $\rightarrow$ System Capability

System-Capability Tree with re-delegation
SEAL: Capability Model

- Capability types
  - User capability ➞ Forwarding facet
  - System capability ➞ Revoking facet

- System-Capability Tree
  - Tracking delegations
  - Fast revocation

Re-delegation
SEAL: Capability Model

• Capability types
  - User capability  \(\equiv\)  Forwarding facet
  - System capability  \(\equiv\)  Revoking facet

• System-Capability Tree
  - Tracking delegations
  - Fast revocation
SEAL: Stateful System Model
SEAL: Stateful System Model

- A finite state machine represent possible orders of computations
- SEAL extends Rei policy language
- A data owners defined the state machine as a policy set
SEAL: Stateful System Model

- A **finite state machine** represent possible orders of computations
- SEAL extends Rei policy language
- A data owners defined the state machine as a policy set

![Finite State Machine Diagram]

- **Start** → **S₁** → **S₂** → **End₁** (Path a)
- **Start** → **S₁** → **S₃** → **End₁** (Path b)
- **Start** → **S₁** → **S₄** → **End₂** (Path c)
- **Start** → **End₂** (Path f)
- **Start** → **End₂** (Path g)
- **Start** → **S₄** → **End₁** (Path e)

Data owner → SEAL → Policy
SEAL: Stateful System Model

- A **finite state machine** represent possible orders of computations
- SEAL extends Rei policy language
- A data owners defined the state machine as a policy set

(Data owner)

Policy

No Sensitive Data

![Diagram of SEAL state machine](image)
SEAL: Stateful System Model

- A **finite state machine** represent possible orders of computations
- SEAL extends Rei policy language
- A data owners defined the state machine as a policy set

![Finite State Machine Diagram]

Data owner

Policy

SEAL

---

8
SEAL: Security Labels Tracking

• SEAL tracks security labels
  - Computation level (transition tracing)
  - Data level (taint tracking: **High** vs. **Low**)

"High" and "Low" are used to denote different levels of security labels. **High** typically represents a higher level of security, while **Low** represents a lower level.
SEAL: Security Labels Tracking

• SEAL tracks security labels
  - Computation level (transition tracing)
  - Data level (taint tracking: **High** vs. **Low**)

• For example:
  - Current state = \( S_3 \)
SEAL: Security Labels Tracking

• SEAL tracks security labels
  − Computation level (transition tracing)
  − Data level (taint tracking: **High** vs. **Low**)

• For example:
  − Current state = $S_3$
    − Computation trace = \{s, a, b\}
SEAL: Security Labels Tracking

- SEAL tracks security labels
  - Computation level (transition tracing)
  - Data level (taint tracking: **High** vs. **Low**)

- For example:
  - Current state = \(S_3\)
    - Computation trace = \(\{s, a, b\}\)
    - Current data taint = \(\{High\}\)
SEAL: Permissions
SEAL: Permissions

• A capability contains a set of Permissions

• Permission = transition +
  - data_predicate(security labels) +
  - computation_predicate(security labels)
SEAL: Permissions

- A capability contains a set of Permissions
- **Permission** = transition +
  data_predicate(security labels) +
  computation_predicate(security labels)
- For example:
  
  \[
  P_1: \{s, \text{High} \vee \text{Low}\} \\
  P_2: \{a, \text{LOW}\} \\
  P_3: \{a, \text{High} \vee \text{Low}\} 
  \]
• A capability contains a set of Permissions

• Permission = transition +
  data_predicate(security labels) +
  computation_predicate(security labels)

• For example:
  \( P_1: \{s, \text{High} \lor \text{Low}\} \)
  \( P_2: \{a, \text{LOW}\} \)
  \( P_3: \{a, \text{High} \lor \text{Low}\} \)

\[ C \ni P_2 \in C \land P_3 \notin C \]
**SEAL: Permissions**

• A capability contains a set of Permissions

• **Permission** = transition +
  
  data_predicate(security labels) +
  computation_predicate(security labels)

• For example:

  \[ P_1: \{ s, \text{High} \lor \text{Low} \} \]
  \[ P_2: \{ a, \text{LOW} \} \]
  \[ P_3: \{ a, \text{High} \lor \text{Low} \} \]

\[ C \ni P_3 \in C \]

Analyst
Case Study: Statistical Analysis
Case Study: Statistical Analysis

- Selecting a subset of data records and count them
- The *Publish_Result* action adds noise to the result
Case Study: Model Training with Taint Tracking

- SEAL can track the taint of every bit during a computation
- Data owners can leverage the provided taint-tracking mechanism
SEAL: Implementation

• A proof-of-concept implementation
• Secure program execution: Capsicum framework
• Taint-tracking:
  - Data flow: Python object proxies for direct taint propagation
  - Control flow: Statically instruments the source code to keep track of indirect taint propagation due to control flow
• Libraries
  - Transfer libraries to LLVM-Intermediate representation (IR) using Numba
  - Static taint tracking using PhASAR
SEAL: Evaluation

- We evaluated scenarios on three real-world datasets *
  - *Adult* dataset (32,561 entries)
  - *Incident-Report* dataset (141,713 entries)
  - *Household-Power-Consumption* dataset (2,075,258 entries)
SEAL: Evaluation

- We evaluated scenarios on three real-world datasets *
  - *Adult* dataset (32,561 entries)
  - *Incident-Report* dataset (141,713 entries)
  - *Household-Power-Consumption* dataset (2,075,258 entries)

* from UCI Machine-Learning Repository
Key Takeaways
Key Takeaways

• SEAL resolves the trust issue between data owners and analytics
**Key Takeaways**

- SEAL resolves the trust issue between data owners and analytics
- SEAL is a fine-grained access-control framework for data-analytics scenarios
  - Capability-object model
  - Stateful system model
  - Security label tracking
Key Takeaways

• SEAL resolves the trust issue between data owners and analytics
• SEAL is a fine-grained access-control framework for data-analytics scenarios
  − Capability-object model
  − Stateful system model
  − Security label tracking
• SEAL can be employed in the real-world scenarios with a reasonable overhead
Back-up Slides
SEAL: Threat Model
SEAL: Threat Model

- System Security
  - trusted: the framework’s hosting machine + Capsicum
**SEAL: Threat Model**

- System Security
  - **trusted**: the framework’s hosting machine + Capsicum
  - **assumed**: analysts act as adversaries + secured connections + network-based attacks are prevented
SEAL: Threat Model

• System Security
  - trusted: the framework’s hosting machine + Capsicum
  - assumed: analysts act as adversaries + secured connections + network-based attacks are prevented

• Data Privacy regarding machine learning (Following Nasr et al.*)
SEAL: Threat Model

• System Security
  - **trusted**: the framework’s hosting machine + Capsicum
  - **assumed**: analysts act as adversaries + secured connections + network-based attacks are prevented

• Data Privacy regarding machine learning (Following Nasr et al.∗)
  - **weak** adversaries: can train models and evaluate their data with trained models
SEAL: Threat Model

• System Security
  - **trusted**: the framework’s hosting machine + Capsicum
  - **assumed**: analysts act as adversaries + secured connections + network-based attacks are prevented

• Data Privacy regarding machine learning (Following Nasr et al.)*
  - **weak** adversaries: can train models and evaluate their data with trained models
  - **medium** adversaries: weak adversaries + can request models
SEAL: Threat Model

- **System Security**
  - **trusted**: the framework’s hosting machine + Capsicum
  - **assumed**: analysts act as adversaries + secured connections + network-based attacks are prevented

- **Data Privacy regarding machine learning** (Following Nasr et al.*)
  - **weak** adversaries: can train models and evaluate their data with trained models
  - **medium** adversaries: weak adversaries + can request models
  - **strong** adversaries: medium adversaries + can apply their datasets during training models

---

* Nasr et al., “Adversary Instantiation: Lower Bounds for Differentially Private Machine Learning”. In 2021 IEEE Symposium on Security and Privacy (SP)
Seal: Approach
Seal: Approach

- Based on capability-object model
  - tracking capabilities
  - revoking capability hierarchies

- Stateful system model
  - defining possible orders of computations

- Security labels tracking
  - data level
  - computation level
• Operates in two phases
  - initialisation phase (steps A - D)
  - execution phase (steps 1 - 13)
SEAL: Components
SEAL: Components

- Access Controller
  - orchestrates operations
- Capability Manager
SEAL: Components

- **Access Controller**
  - orchestrates operations

- **Capability Manager**
  - handles delegating/revoking capabilities
  - verifies capabilities
SEAL: Components

- **Access Controller**
  - orchestrates operations

- **Capability Manager**
  - handles delegating/revoking capabilities
  - verifies capabilities

- **Policy Manager**
  - checks requests and keeps their trace

- **File Manager**
SEAL: Components

- **Access Controller**
  - orchestrates operations

- **Capability Manager**
  - handles delegating/revoking capabilities
  - verifies capabilities

- **Policy Manager**
  - checks requests and keeps their trace

- **File Manager**
  - creates file handlers (using Capsicum)

- **Execution Manager**
SEAL: Components

- **Access Controller**
  - orchestrates operations

- **Capability Manager**
  - handles delegating/revoking capabilities
  - verifies capabilities

- **Policy Manager**
  - checks requests and keeps their trace

- **File Manager**
  - creates file handlers (using Capsicum)

- **Execution Manager**
  - execute computations (inside Capsicum sandboxes)
SEAL: Security Policies

• A system’s state transforms based on policies
• Extended Rei policy language
SEAL: Security Policies

- A system’s state transforms based on policies
- Extended Rei policy language
  - Rei consists of constructs: *rights*, *prohibitions*, *obligations*
SEAL: Security Policies

• A system’s state transforms based on policies
• Extended Rei policy language
  – Rei consists of constructs: *rights*, *prohibitions*, *obligations*

• Added Two policy constructs
**SEAL: Security Policies**

- A system’s state transforms based on policies
- Extended Rei policy language
  - Rei consists of constructs: *rights*, *prohibitions*, *obligations*
- Added Two policy constructs
  - *StateObject*: defines a system’s state
SEAL: Security Policies

• A system’s state transforms based on policies
• Extended Rei policy language
  − Rei consists of constructs: rights, prohibitions, obligations

• Added Two policy constructs
  − StateObject: defines a system’s state
  − ACTION: defines a possible computation
SEAL: Security Policies

- A system’s state transforms based on policies
- Extended Rei policy language
  - Rei consists of constructs: *rights*, *prohibitions*, *obligations*
- Added Two policy constructs
  - *StateObject*: defines a system’s state
  - *ACTION*: defines a possible computation
- Rights define state transitions
SEAL: Security Policies

- A system’s state transforms based on policies
- Extended Rei policy language
  - Rei consists of constructs: *rights*, *prohibitions*, *obligations*
- Added Two policy constructs
  - **StateObject**: defines a system’s state
  - **ACTION**: defines a possible computation
- Rights define state transitions
- A capability includes rights
Security Policies - An Example
Security Policies - An Example

ACTION(Query_Data, query_data_function,
  Paramset(query_data_parameters,
    params(param(any-of-these, Listkv_String),
      param(all-of-these, Listkv_String))),
  Require(taint-tracking))
Security Policies - An Example

\[
\text{ACTION}(\text{Query\_Data}, \text{query\_data\_function}, \text{Paramset}(\text{query\_data\_parameters}, \text{params}(\text{param}(\text{any\_of\_these}, \text{Listkv\_String}), \text{param}(\text{all\_of\_these}, \text{Listkv\_String}))), \text{Require}(\text{taint\_tracking}))
\]

\[
\text{RIGHT}(\text{data\_query}, \text{Query\_Data}, \text{StateObject}(\text{START}), \text{StateObject}(\text{Queried\_Data}), \text{Obligation}())
\]
Case Study: First Scenario

Diagram:
- Start
- Select_Data → Selected_Data → Count → Computed_Data → Publish_Result
- Go_To
Case Study: First Scenario

- Statistical Analysis
- Selecting a subset of data records and count them
Case Study: First Scenario

- Statistical Analysis
- Selecting a subset of data records and count them
- The *Publish_Result* action adds noise to the result
Case Study: Second Scenario
Case Study: Second Scenario

- Differentially Private Machine Learning
- Reduce an analyst’s budget based on the types of adversaries
Case Study: Second Scenario

- Differentially Private Machine Learning
- Reduce an analyst’s budget based on the types of adversaries
  weak adversaries
Case Study: Second Scenario

- Differentially Private Machine Learning
- Reduce an analyst’s budget based on the types of adversaries
  - weak adversaries
  - medium adversaries
Case Study: Second Scenario

- Differentially Private Machine Learning
- Reduce an analyst’s budget based on the types of adversaries:
  - weak adversaries
  - medium adversaries
  - strong adversaries
Case Study: Third Scenario

- Processing data with analysts’ programs
- The *Publish_Result* action adds noise to the result
Fourth Scenario: Model Training with Taint Tracking

- SEAL can track the taint of every bit during a computation
- Data owners can leverage the provided taint-tracking mechanism
- SEAL Can evaluate Rights based on the data taints
• We evaluated on three real-world datasets *
  - **Adult** dataset (32,561 entries)
  - **Incident-Report** dataset (141,713 entries)
  - **Household-Power-Consumption** dataset (2,075,258 entries)