



SecLab

COMPUTER SECURITY LABORATORY

19<sup>th</sup> Sep., 2024

# STYX: Collaborative and Private Data Processing Using TEE-Enforced Sticky Policy

Shixuan Zhao

PhD Candidate @ SecLab

CSE, The Ohio State University

zhao.3289@osu.edu



THE OHIO STATE UNIVERSITY

COLLEGE OF ENGINEERING

# Acknowledgement



This project is funded by NSF, under the CDCC Frontier grant.



CENTER FOR  
DISTRIBUTED CONFIDENTIAL COMPUTING

# Policy-Attached Data: Challenges

## Conventionally...

- Embedded in code

## Sticky Policy

- Sticky policy: policy attached to data
- Flexible policy customised for every data



## ⚠️ Problems

- Data-in-use protection?
- Data lifecycle protection?
- Dynamic collaboration?

# Policy-Attached Data: Challenges

## Data-in-Use Protection

- Identity-Based Encryption [1]
- Attribute-Based Encryption [2][3]
- Proxy Re-Encryption [4]



## ⚠ Problem

They all have to trust the application!

[1] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In Annual international cryptology conference, pages 213–229. Springer, 2001.

[2] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006, pages 89– 98. ACM, 2006.

[3] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute- based encryption. In 2007 IEEE symposium on security and privacy (SP'07), pages 321–334. IEEE, 2007.

[4] M. Green and G. Ateniese. Identity-based proxy re-encryption. In Applied Cryptography and Network Security: 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007. Proceedings 5, pages 288– 306. Springer, 2007.

# Policy-Attached Data: Challenges

## Data Lifecycle Protection

- Handles derived data
- How to protect the output?
- What is allowed in the output?
- What policy should be attached to the output?

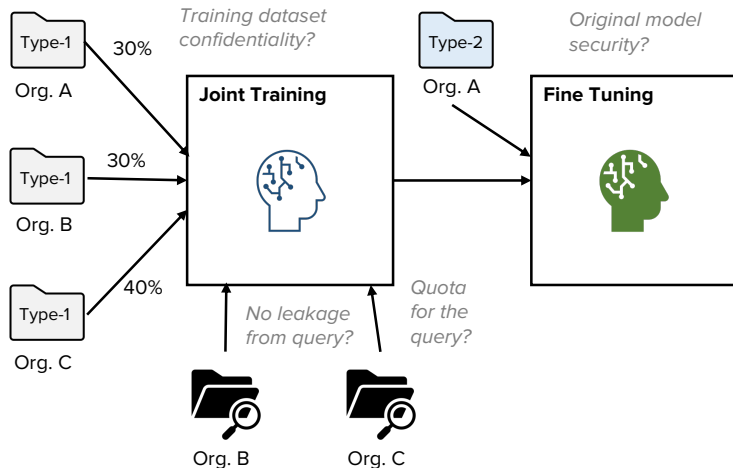


## ⚠ Problem

How to achieve the above stuff?

# Motivating Example

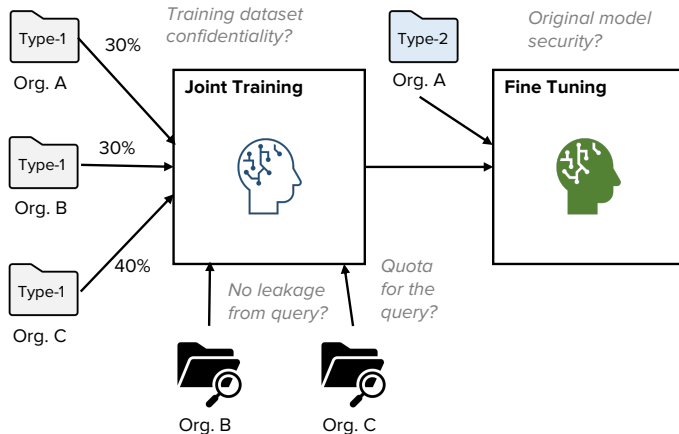
Hospitals train a model to classify cancer



# Motivating Example: Data-in-Use Protection

## Training

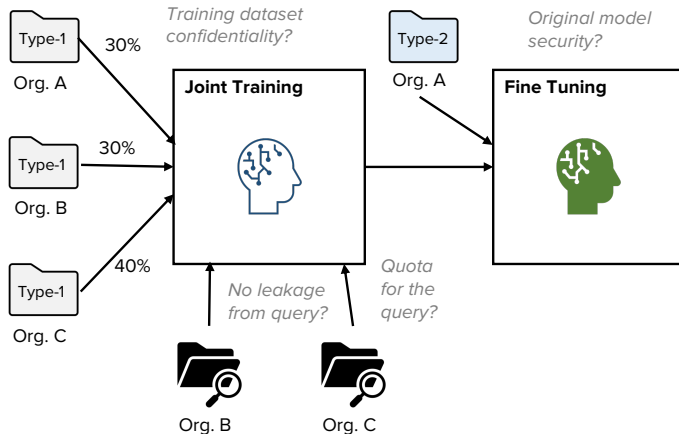
- How to make sure my dataset won't be over a limit? (e.g. 30%)
- How to guarantee the **confidentiality** of my training dataset?



# Motivating Example: Data Lifecycle Protection

## Model Usage

- Jointly owned
- How to limit **fair-use**? (e.g. Quota according to training set contribution)
- How to safeguard the model?

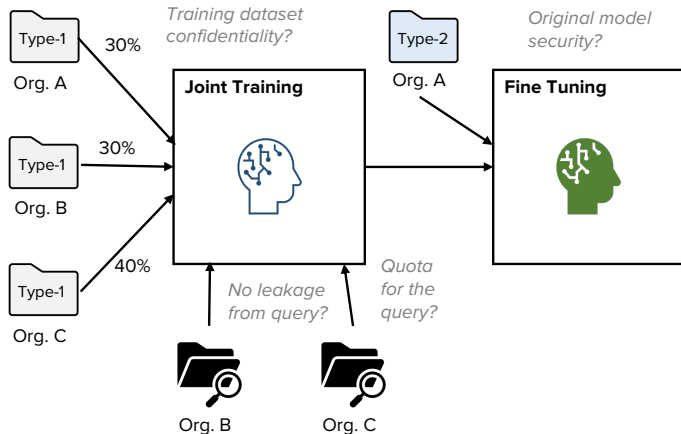




# Motivating Example: Data Lifecycle Protection

## Derived Model

- Is it even allowed?
- Exclusively owned with jointly owned bits
- How to ensure the security of the original model?



# Problem

## ⚠ Problems

- Data-in-use protection?
- Data lifecycle protection?
- Dynamic collaboration?



# Problem

## ⚠ Problems

- Data-in-use protection?
- Data lifecycle protection?
- Dynamic collaboration?

## Solutions

- TEE hardware-enforced trust and isolation
- Cryptographically-encrypted transmission
- Sandboxed access
- Policy enforced for both input and output



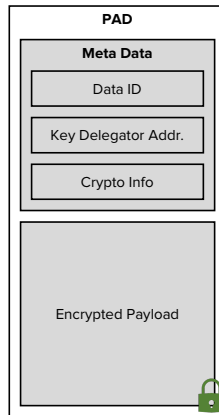
# What does a PAD look like?

## Plaintext Contents

- Information to get the key
- How to decrypt

## Ciphertext Contents

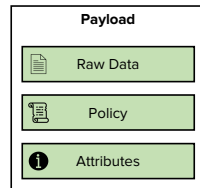
- Everything else!
- Data & policy
- Data attributes used by policy



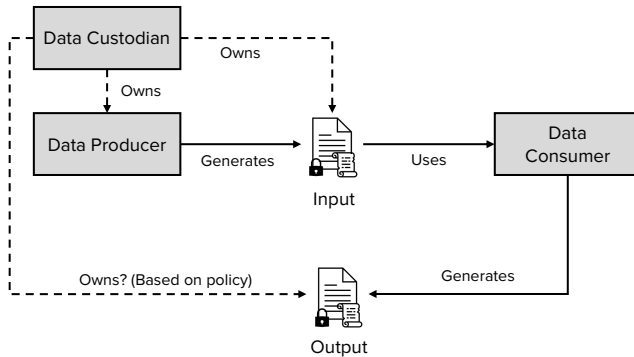
Data ID Example:  
9968d394-303b-42eb-bd99-ebc8054beacb

Key Delegator Address Example:  
<https://key-delegator.xxx.com/xxx>

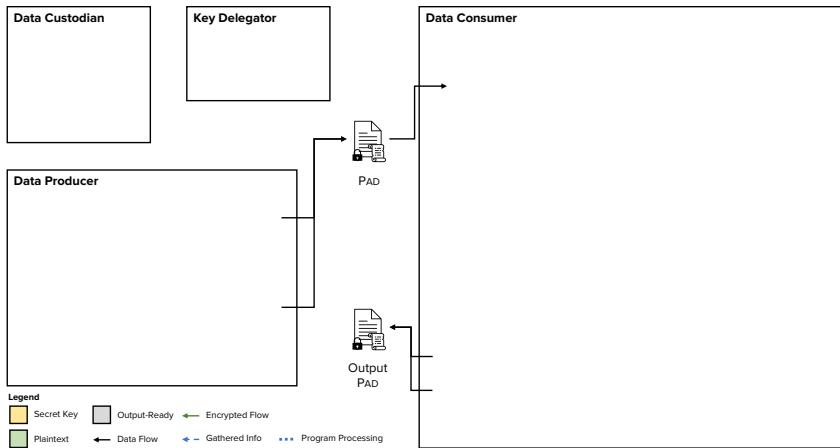
Crypto Info Example:  
AES-GCM 128



# Modelling the Workflow



# Modelling the Workflow



**Data Custodian**

- Key (Secret Key)
- Policy (Plaintext)

**Key Delegator**

- Key (Secret Key)

**Data Producer**

- New Key (Secret Key)
- Policy (Plaintext)
- Raw Data (Plaintext)
- Attributes (Plaintext)
- Meta Data (Output-Ready)
- Encrypted Payload (Output-Ready)

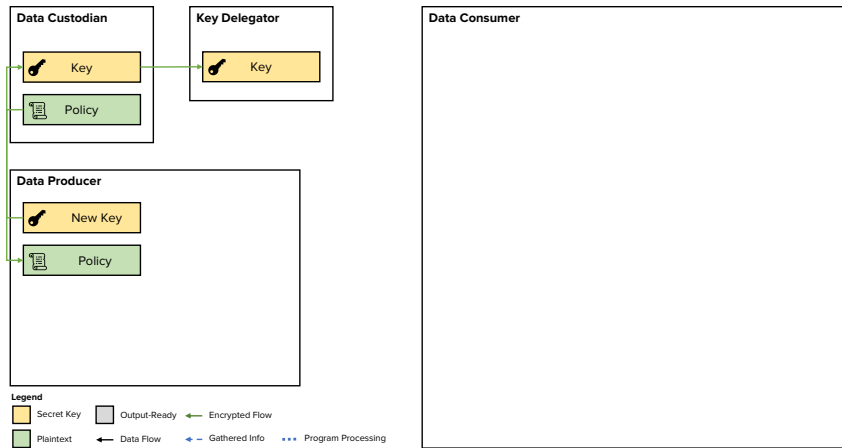
**Data Consumer**

- Sandbox**
  - Consumer Program (Program Processing)
  - Output Data (Plaintext)
  - Attributes (Plaintext)
  - Proposed Policy (Plaintext)
- Dataset**
  - Meta Data (Output-Ready)
  - Encrypted Payload (Output-Ready)
  - Raw Data (Plaintext)
  - Attributes (Plaintext)
  - Policy (Plaintext)
- Policy Engine** (Program Processing)
- Original Key (Secret Key)
- New Key (Secret Key)
- Output Data (Plaintext)
- Attributes (Plaintext)
- Checked Policy (Plaintext)

**Legend**

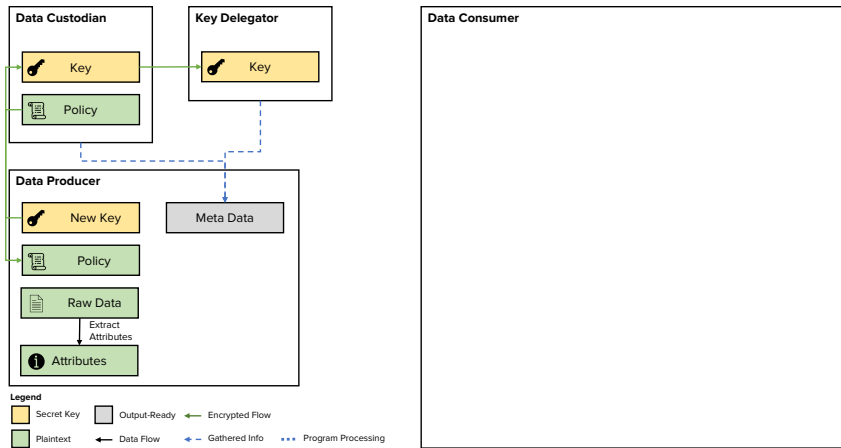
- Secret Key (Yellow box)
- Output-Ready (Grey box)
- Plaintext (Green box)
- Encrypted Flow (Green arrow)
- Data Flow (Black arrow)
- Gathered Info (Blue dashed arrow)
- Program Processing (Blue dotted arrow)

# Workflow: Step-by-Step

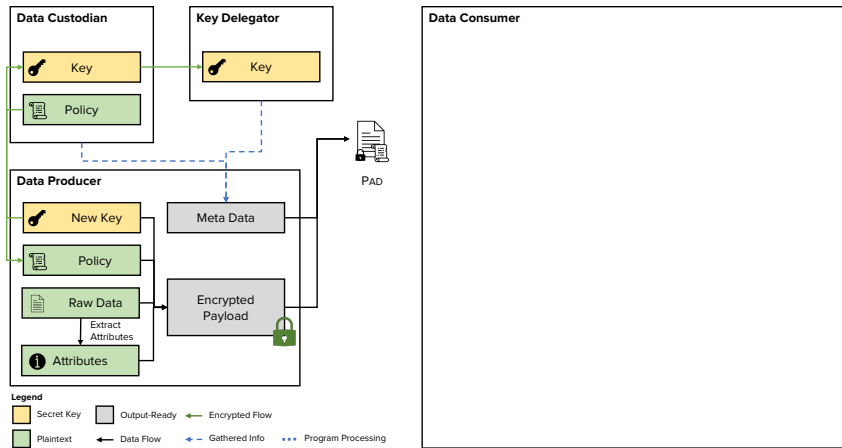




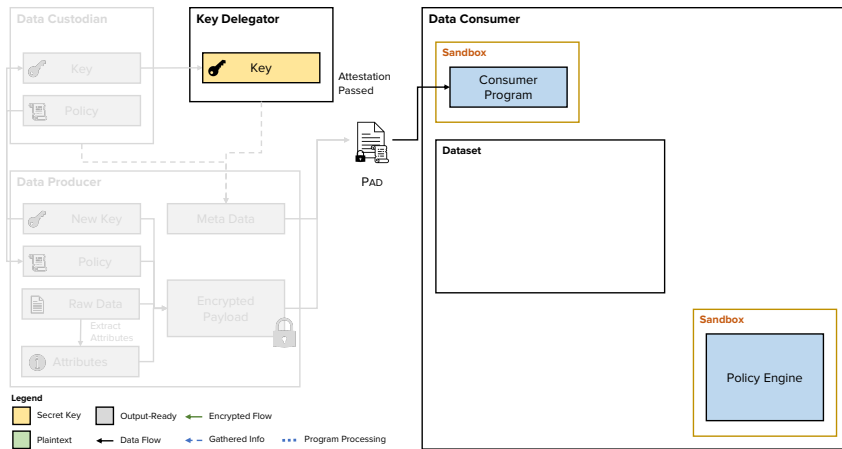
# Workflow: Step-by-Step



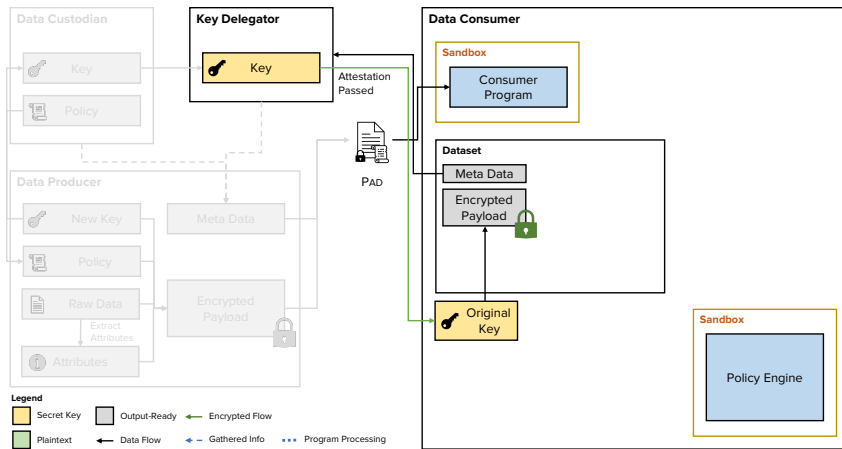
# Workflow: Step-by-Step



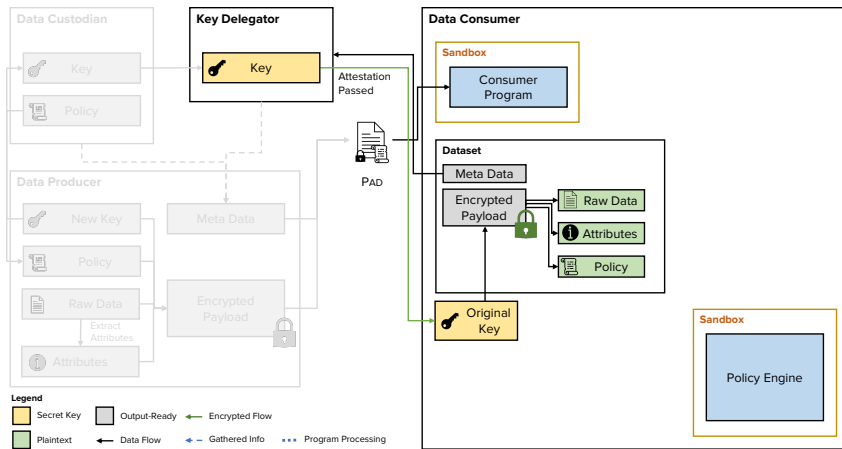
# Workflow: Step-by-Step



# Workflow: Step-by-Step



# Workflow: Step-by-Step



The diagram illustrates the architecture and data flow between four main components: Data Custodian, Key Delegator, Data Producer, and Data Consumer.

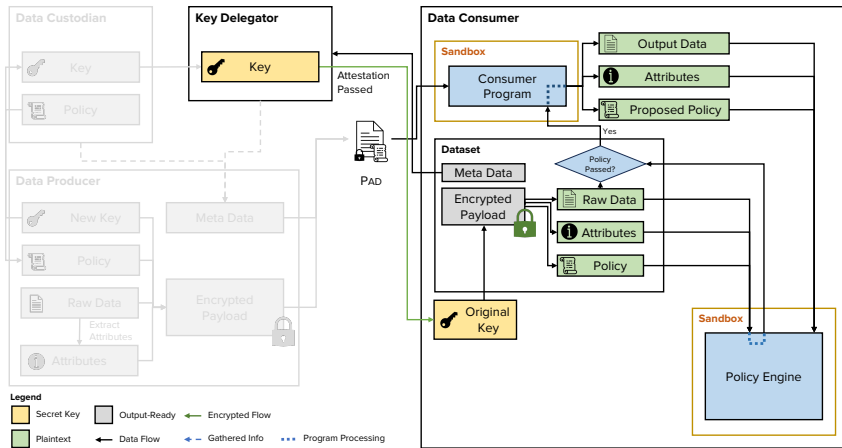
- Data Custodian:** Contains a **Key** (Secret Key) and a **Policy** (Output-Ready).
- Key Delegator:** Contains a **Key** (Secret Key). It receives the Key from the Data Custodian and provides an **Attestation Passed** signal to the Data Consumer.
- Data Producer:**
  - Inputs: **New Key** (Secret Key), **Policy** (Output-Ready), **Raw Data** (Plaintext), and **Attributes** (Plaintext).
  - Process: **Raw Data** is processed to **Extract Attributes**, which are then combined with **Raw Data** to create an **Encrypted Payload** (Output-Ready).
  - Output: The **Encrypted Payload** is sent to the Data Consumer.
- Data Consumer:**
  - Sandbox:** Contains a **Consumer Program** (Output-Ready).
  - Dataset:**
    - Inputs: **Meta Data** (Output-Ready), **Encrypted Payload** (Output-Ready), and **Original Key** (Secret Key).
    - Process: The **Encrypted Payload** is decrypted using the **Original Key** to produce **Raw Data** (Plaintext), **Attributes** (Plaintext), and **Policy** (Plaintext).
    - Policy Engine:** Receives **Raw Data**, **Attributes**, and **Policy** from the Dataset. It performs **Program Processing** (indicated by blue dots) and outputs **Gathered Info** (blue dashed line) back to the Dataset.
    - Policy Passed?** Decision: Receives **Gathered Info** and outputs **Yes** to the **Consumer Program** in the Sandbox.
  - PAD (Policy Attestation Document):** Receives the **Attestation Passed** signal from the Key Delegator and provides input to the **Consumer Program** in the Sandbox.

**Legend:**

- Secret Key:** Yellow box with a key icon.
- Output-Ready:** Grey box with a document icon.
- Plaintext:** Green box with a document icon.
- Data Flow:** Solid black arrow.
- Encrypted Flow:** Green arrow.
- Gathered Info:** Blue dashed arrow.
- Program Processing:** Blue dots.



# Workflow: Step-by-Step





**Data Custodian**

- Key
- Policy

**Key Delegator**

- Key

**Data Producer**

- New Key
- Policy
- Raw Data
- Attributes
- Meta Data
- Encrypted Payload

**Data Consumer**

- Sandbox**
  - Consumer Program
  - Dataset
    - Meta Data
    - Encrypted Payload
    - Raw Data
    - Attributes
    - Policy
  - Policy Engine
- Program Processing**
  - Encrypted Payload
  - Original Key
  - Meta Data
  - Encrypted Payload
  - New Key
  - Output Data
  - Attributes
  - Checked Policy
- Output Data
- Attributes
- Proposed Policy
- Policy Passed?
- Raw Data
- Attributes
- Policy
- Output PAD
- Encrypted Payload

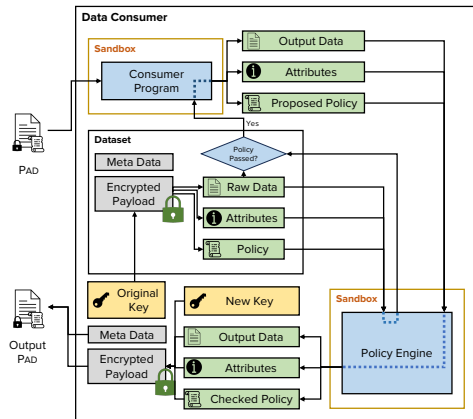
**Legend**

- Secret Key
- Output-Ready
- Encrypted Flow
- Plaintext
- Data Flow
- Gathered Info
- Program Processing

# Consumer: From Model to Architecture

## Model

- Sandboxing
- Policy engine
- I/O protection
- Trust



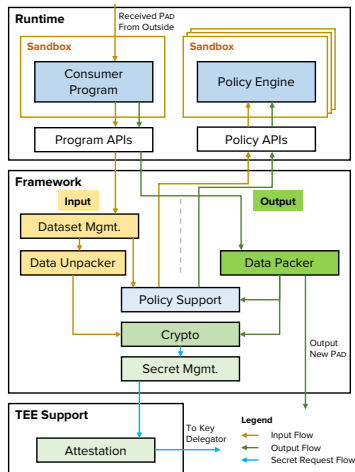
# Consumer: From Model to Architecture

## ⚠️ A middleware

A middleware with a runtime to achieve the security guarantee

## Architecture

- Sandboxing via runtime
- Policy engine using runtime
- I/O protection by middleware
- Trust with remote attestation



# Implementation

## Framework

- Architecture-independent
- APIs for different TEEs & runtimes

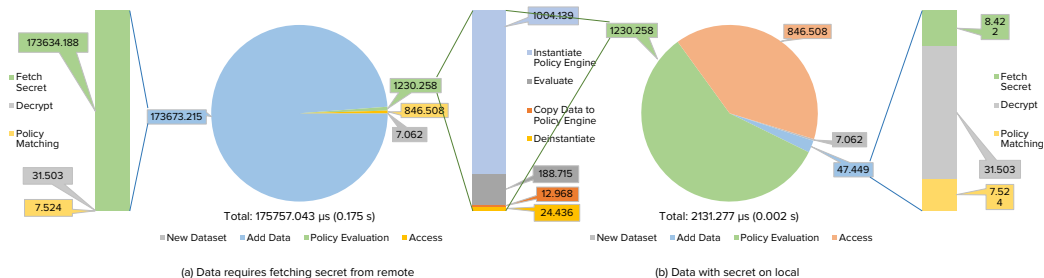


## Prototype

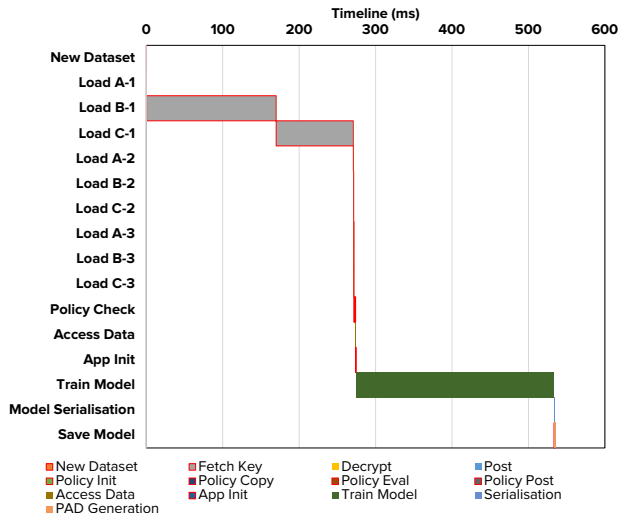
- Intel SGX as TEE
- WebAssembly as runtime



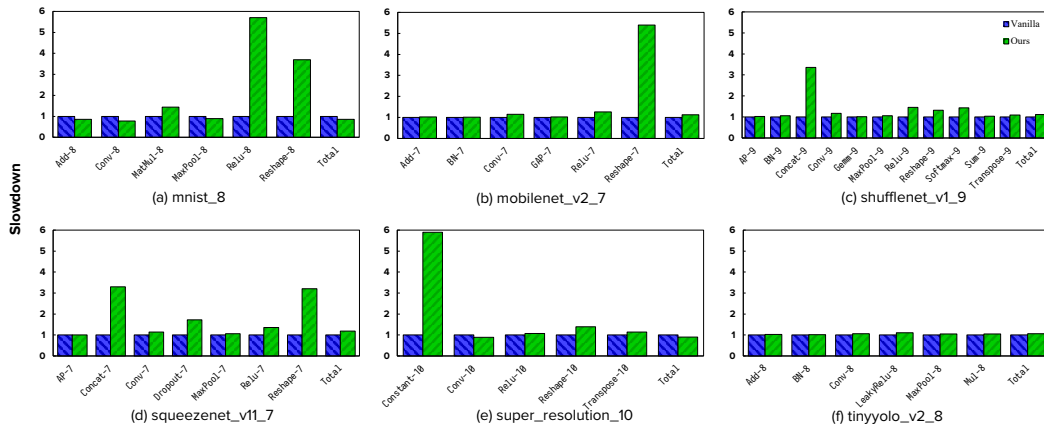
# Data Access Overhead



# SVM Training



# libonnx Benchmarks with Native Libs



# Other Use Cases

## Joint Data Analysis

- Cross-bank fraud analysis
- Output must be limited to original bank

## Private Database Query

- Paid query to proprietary databases
- Query must not be leaked

## Smart Home IoT

- Connected security services (e.g. ADT)
- Home insurance
- Access limited to data type/specific timing



# Limitations and Future Works

## Hardware Accelerators

- GPU TEEs (e.g., NVIDIA H100)
- How to apply the method to GPU?
- Trusted I/O

## Trade-Offs in Runtime Sandboxing

- Other methods (e.g., SFI)
- AoT & JIT compilation
- Native libs

## Formal Verification of Middleware

- Full stack verification required
- Verified runtime (e.g., MesaPy)

## Source Code

Will be available after published

## Q&A

### SecLab @ OSU

<https://go.osu.edu/seclab>

### Shixuan Zhao's Homepage

<https://nskernell.org>