IPSN 2024

CPS-IoT Week 2024
Hong Kong, China, May 13-16

dTEE

A Declarative Approach to Secure IoT Applications Using TrustZone

Tong Sun[1], Borui Li[2], Yixiao Teng[1], Yi Gao[1], and Wei Dong[1]

[1]The State Key Laboratory of Blockchain and Data Security

College of Computer Science, Zhejiang University, China

[2]School of Computer Science and Engineering, Southeast University, China

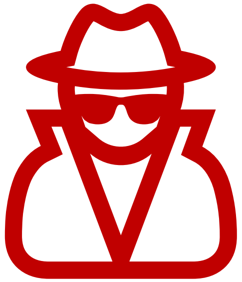
浙江大學 区块链与数据安全全国重点实验室

STATE KEY LABORATORY OF BLOCKCHAIN AND DATA SECURITY
ZHEJIANG UNIVERSITY

東南大學
SOUTHEAST UNIVERSITY

Background

- **Internet of Things (IoT) devices are widely deployed in safety-critical scenarios**



If the device's **rich execution environment** (e.g., operating system) has been compromised

- **Sensitive operations** will be manuplicated
- **Sensitive data** will be modified or leaked

Background

- Vendors leverage the hardware-assisted Trusted Execution Environments (TEEs) to enhance the ability of devices against attacks

IoT

ARM TrustZone

PC

Intel SGX

AMD SEV

Background

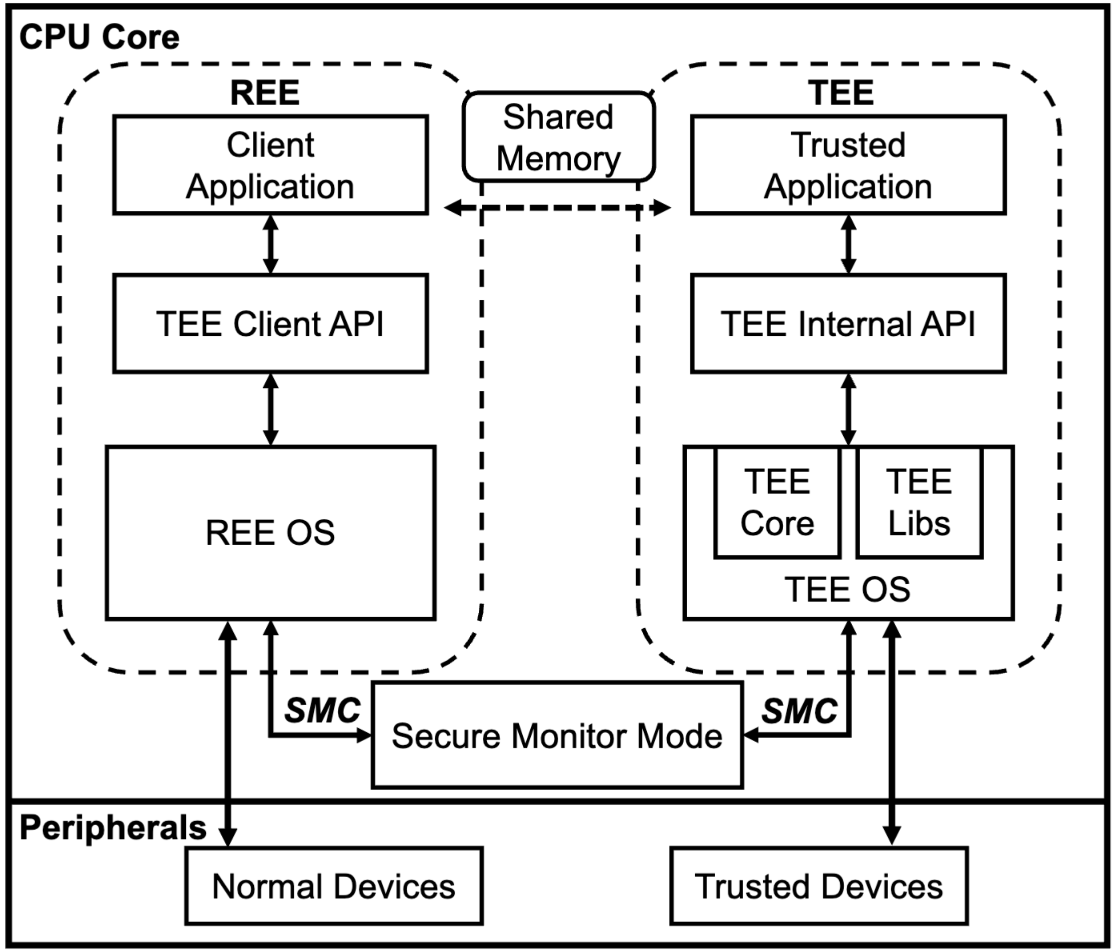
What is TEE?

- **CPU/Memory/Peripherals are isolated into two worlds**
  - Secure World (TEE)
  - Non-secure World (REE)
- **REE OS (Linux, Zephyr, Contiki, …)**
- **TEE OS (OP-TEE, Trusty, …)**

How to use it?

- **Separates an app into two parts**
  - A trusted application (TA)
  - A client application (CA)

TEE-based IoT Platform



Develop TEE-based Apps

- Developers need an in-depth knowledge of TEE APIs to carefully design the control-flow between TA and CA

- Developers should also take care of the data-flow to prevent the disclosure of security-sensitive variables

Unfortunately, securing an existing non-secure IoT app with TEE is not easy.

Previous work

- **Automatic Code Partitioning**
  - [ICSE’16] Automated Partitioning
  - [ATC’17] Glamdring

- **Automatic Code Transformation**
  - [ICDE’21] Twine
  - [ICDCS’22] WATZ

[ICSE’16] Automated Partitioning of Android Applications for Trusted Execution Environments.
[ATC’17] Glamdring: Automatic Application Partitioning for Intel SGX.
[ICDE’21] Twine: An Embedded Trusted Runtime for WebAssembly.
[ICDCS’22] WATZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone

Previous work

- **Automatic code partitioning [ICSE’16,ATC’17]**
  - They encapsulate sensitive data into TA and generate glue code between TA and CA
  - They are suitable for simple apps but fail in the following two scenarios

1. Complicated trusted logic

Developers may not be satisfied with only protecting the data but also try to add customized logic

2. Secure peripheral interactions

The most noticeable distinction between IoT and desktop apps is the interaction of sensing and actuating peripherals

[ICSE’16] Automated Partitioning of Android Applications for Trusted Execution Environments.
[ATC’17] Glamdring: Automatic Application Partitioning for Intel SGX.

Previous work

- **Automatic Code Transformation [ICDE’21, ICDCS’22]**
  - They port the WebAssembly (WASM) runtime in the TEE and compile the whole program into WASM bytecode
  - Two main limitations

1. Secure peripheral interactions

They cannot support peripherals because the WASM runtime lacks of I/O modules

2. Large memory usage

- Protect whole program
- WASM runtime occupies ~50% secure memory

[ICDE’21] Twine: An Embedded Trusted Runtime for WebAssembly.
[ICDCS’22] WATZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone.

Contributions

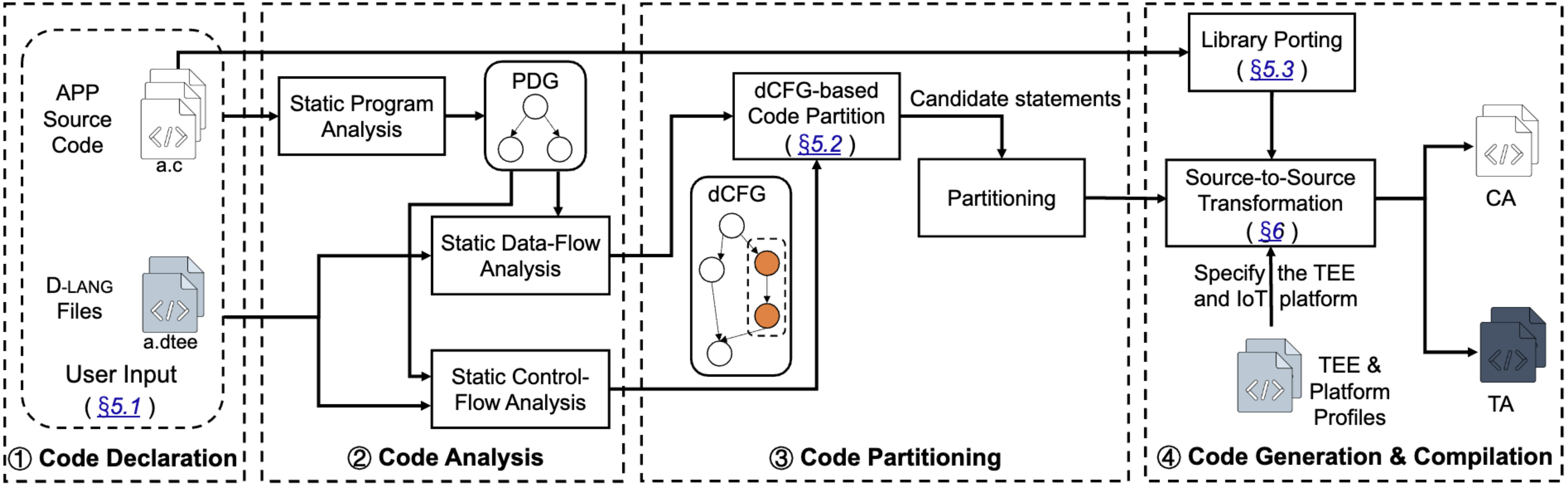
- **dTEE is a novel system to accelerate the development of trusted IoT applications using a declarative approach**
  - D-Lang language to support expressive application development.
- **We propose a graph optimization algorithm to maximize the efficiency of the TEE-based applications**
- **We implement dTEE and extensively evaluate its expressiveness, efficiency, and overhead**

Design Goals

- **Ease of programming**
  - We propose a declarative language D-lang on top of the well-known SQL language
- **Application independent**
  - D-lang is to be developed such that existing apps need no source code modifications
- **Efficiency**
  - Reduce the associated overhead resulting from the enhancement of app security
- **Soundness**
  - e.g., meticulous sanitization of the transformation interfaces

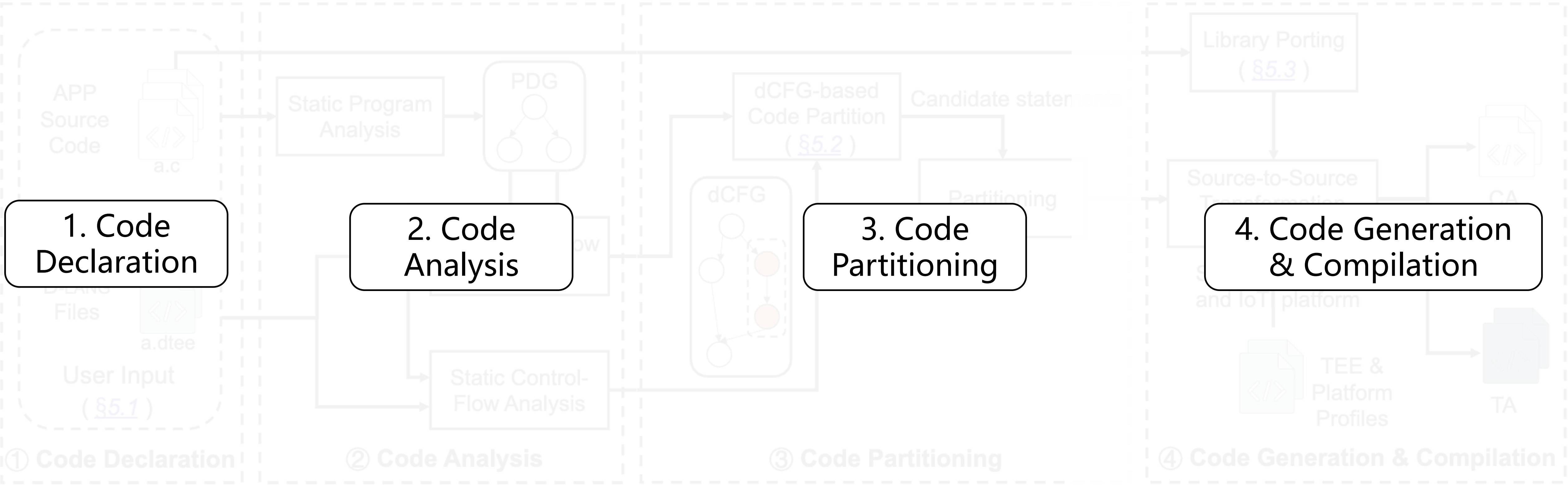
dTEE Workflow

- **Four stages**



dTEE Workflow

- **Four stages**



1. Code Declaration

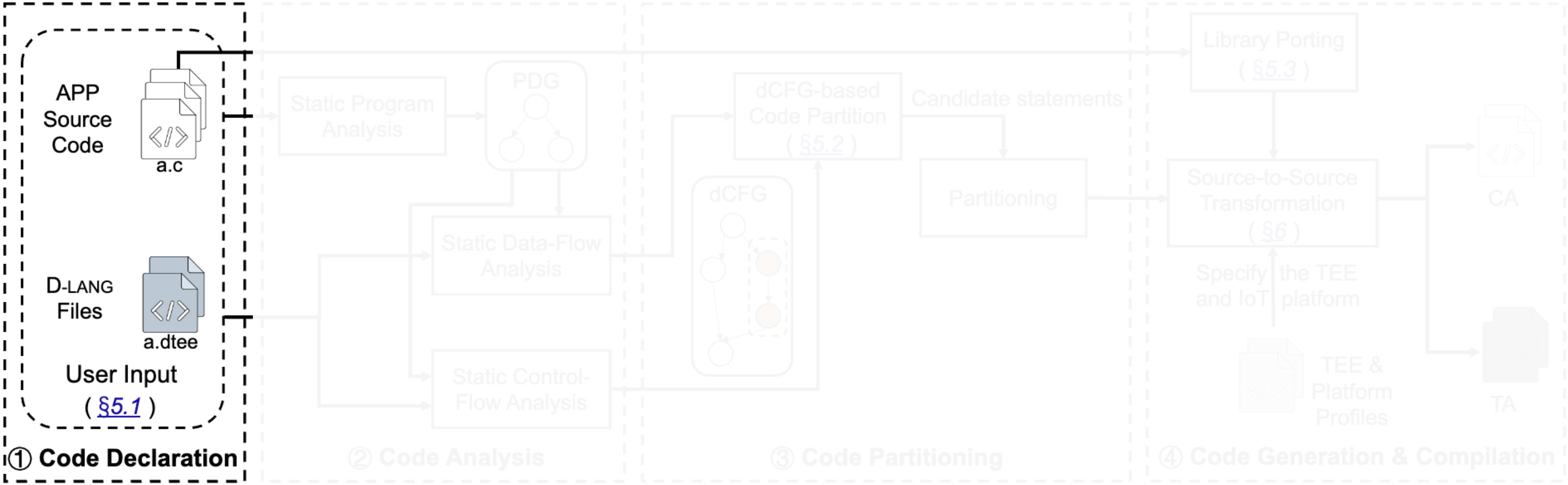
2. Code Analysis

3. Code Partitioning

4. Code Generation & Compilation

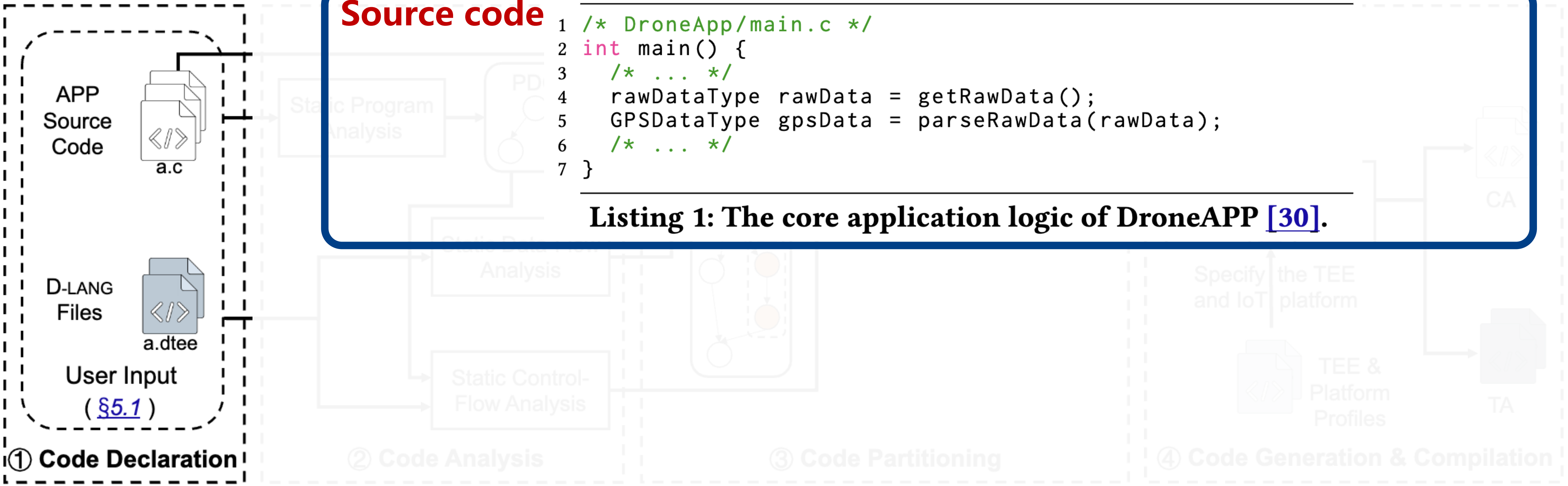
dTEE Workflow

- 1. Code declaration



dTEE Workflow

- 1. Code declaration



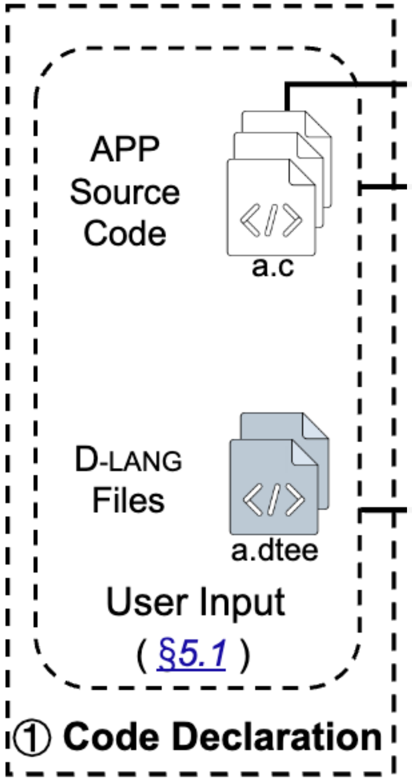
Source code

```
/* DroneApp/main.c */
int main() {
  /* ... */
  rawDataType rawData = getRawData();
  GPSDataType gpsData = parseRawData(rawData);
  /* ... */
}
```

Listing 1: The core application logic of DroneAPP [30].

dTEE Workflow

- 1. Code declaration



Source code

```
/* DroneApp/main.c */
int main() {
  /* ... */
  rawDataType rawData = getRawData();
  GPSDataType gpsData = parseRawData(rawData);
  /* ... */
}
```

Listing 1: The core application logic of DroneAPP [30].

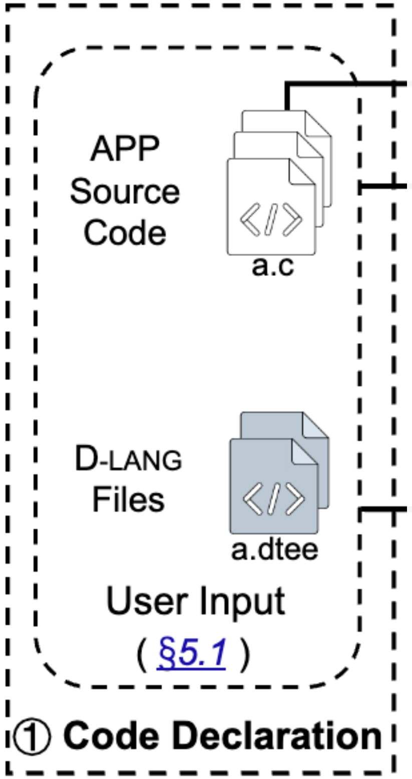
Case 1: Users require that the integrity of GPS data must be protected

D-Lang code

```
/* Case1/drone.dtee */
FROM DroneAPP/main.c FUNC main {
  TZ_DATA_INTG gpsData;
}
```

dTEE Workflow

- 1. Code declaration



Source code

```c
/* DroneApp/main.c */
int main() {
  /* ... */
  rawDataType rawData = getRawData();
  GPSDataType gpsData = parseRawData(rawData);
  /* ... */
}
```

Insert code

Listing 1: The core application logic of DroneAPP [30].

Case 2: Users want to sign the GPS data in the TEE

D-Lang code

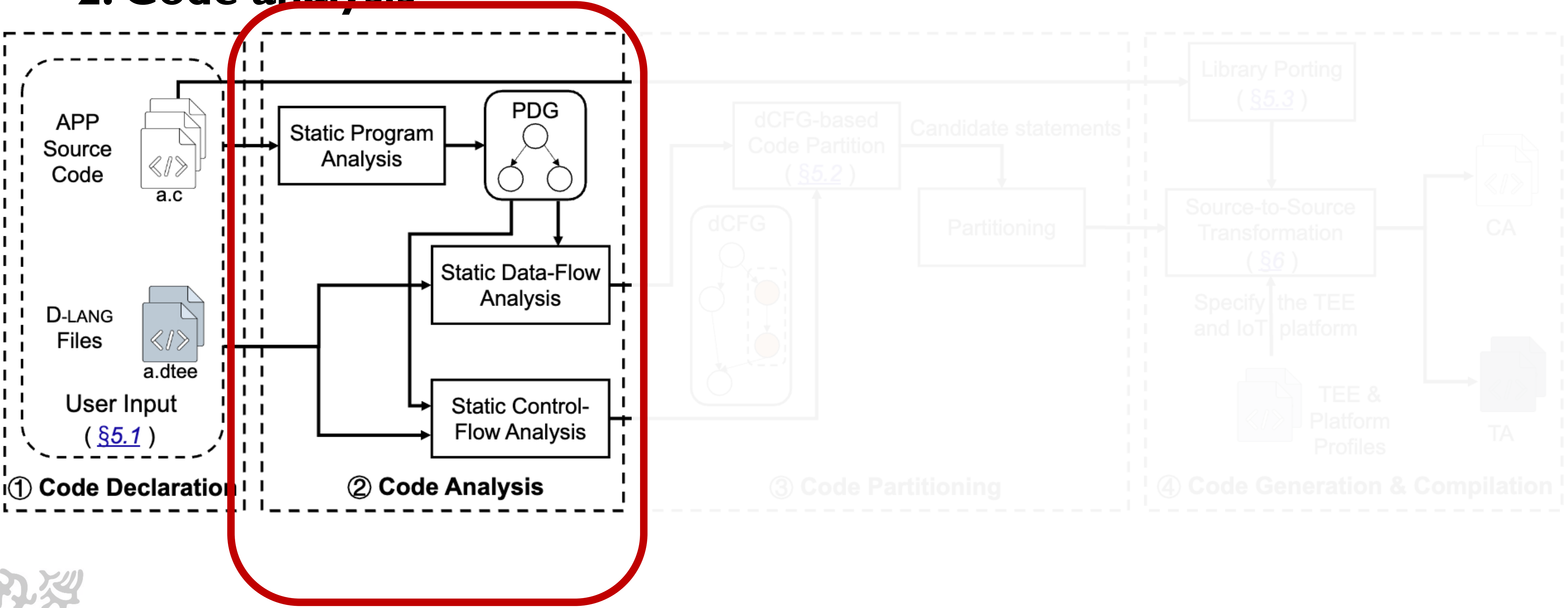
```
/* Case2/drone.dtee */
FROM DroneAPP/main.c FUNC main {
  TZ_DATA_CONF gpsData;
  INSERT_AFTER [ANNT]
    STRING_TZ *private_key;
    TZ_genKey(RSA_KEYPAIR, 2048, private_key, NULL);
    char *signed_gpsData;
    TZ_sign_rsa_sha1(signed_gpsData, private_key, &
      ↪ gpsData, ...);
  END_INSERT
}
```

D-Lang

- **General**
  - Configure the TEE environment
- **Declarative Development of Trusted Logic**
  - Provide built-in functions
- **Tiered Protection**
  - Provide tiered degrees for temporary data protection
  - Provide permanent data protection
  - Provide functions protection

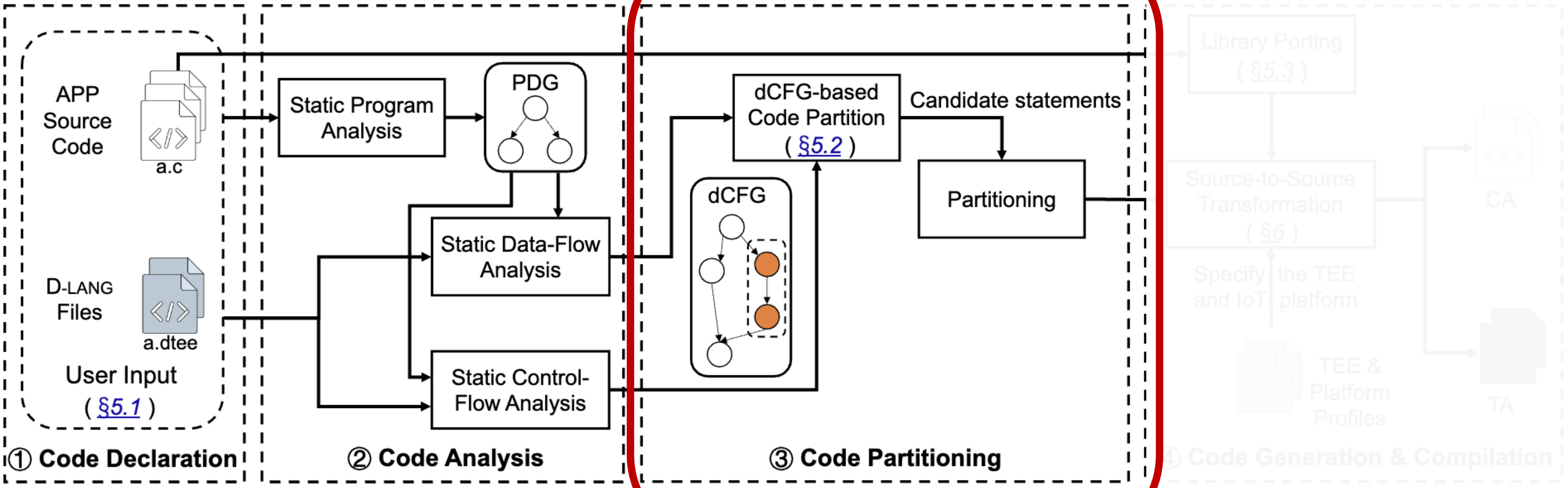
dTEE Workflow

- 2. Code analysis



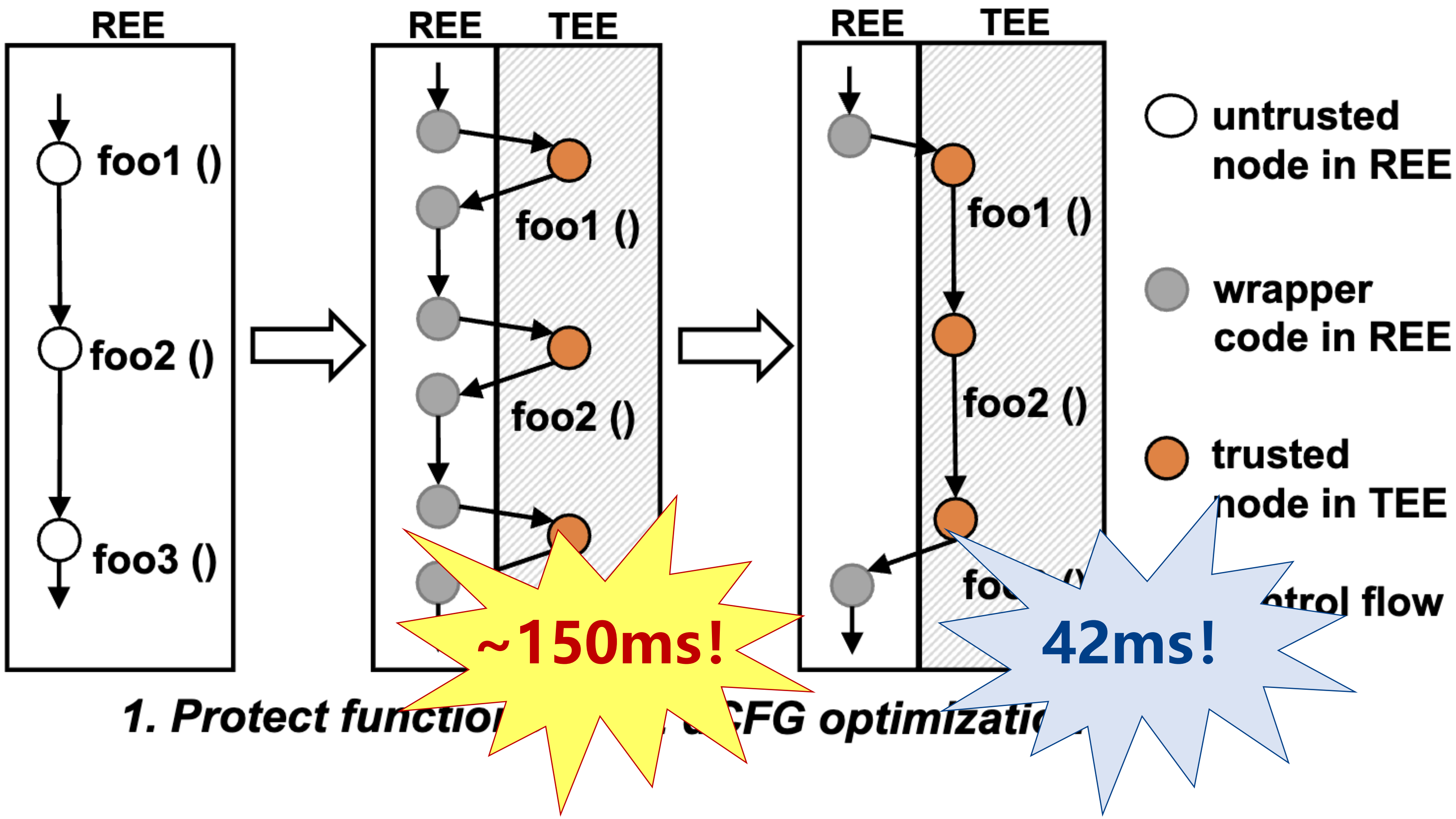
dTEE Workflow

- 3. Code Partitioning



issue#1

- REE and TEE world switching has huge overhead



dCFG-based Code partitioning

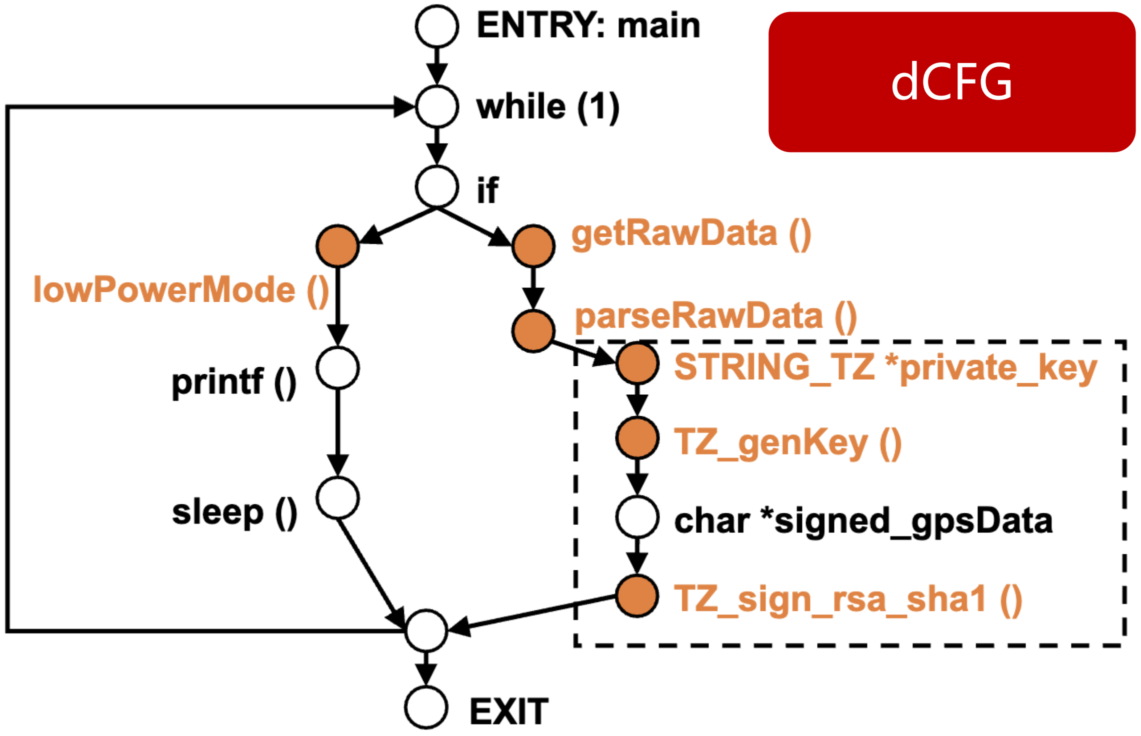
- **dCFG Construction**

```c
/* DroneApp/main.c */
int main() {
  /* ... */ // Initialization
  while(1) { // Primary logic
    if (mode == "Low Power") {
        lowPowerMode();
        printf("Save energy...");
        sleep(1000);
    } else {
        rawDataType rawData = getRawData();
        GPSDataType gpsData = parseRawData(rawData);
        /* ... */
    }
  }
}
```

Drone app source code

```
/* Case2/drone.dtee */
FROM DroneAPP/main.c FUNC main {
  TZ_DATA_CONF gpsData;
  INSERT_AFTER [ANNT]
    STRING_TZ *private_key;
    TZ_genKey(RSA_KEYPAIR, 2048, private_key, NULL);
    char *signed_gpsData;
    TZ_sign_rsa_sha1(signed_gpsData, private_key, &
      ↪ gpsData, ...);
  END_INSERT
}
```

D-Lang code



dCFG-based Code partitioning

- **Partitioning algorithm**
  - **Input:** a directed acyclic graph $G\ (V\ , E)$

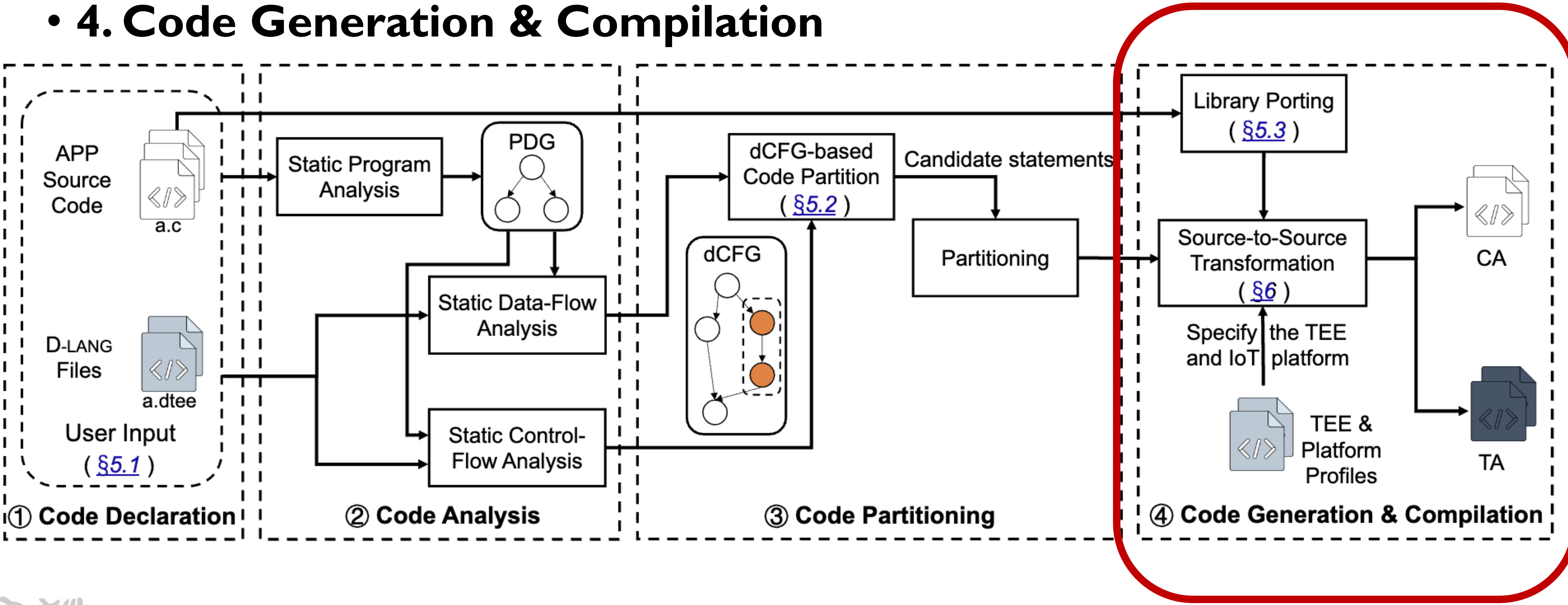
$$X_{b_i w_i} = \begin{cases} 1 & \text{logic block } b_i \text{ is assigned to world } w_i \\ 0 & \text{logic block } b_i \text{ is not assigned to world } w_i \end{cases}$$

$$Y_{b_i s_j} = \begin{cases} 1 & \text{logic block } b_i \text{ uses a variable with sensitivity } s_j \\ 0 & \text{logic block } b_i \text{ does not use variables with sensitivity } s_j \end{cases}$$

  - **Objective**
    - minimize computation and switching time
  - **Constraints**
    - 1. TA memory should be less than TEE secure memory
    - 2. All sensitive operations and data need to be protected in TEE

dTEE Workflow

- 4. Code Generation & Compilation



issue#2

- **Libraries and drivers are different in the REE and TEE**

- **We first analyze IoT libraries**
  - Portable libraries
  - **Auto-transformable libraries**
  - Manually-transformable libraries

Peripheral-oriented Library Porting Mechanism

- **Non-MMU devices (easy)**
  - directly configure physical addresses
- **MMU-equipped devices (hard)**
  - map physical and virtual addresses (e.g., via mmap interface)
  - TA lacks permissions for physical address mapping

- **dTEE extends the basic mapping mechanism for TEE OS**
  - Utilizes the phys_to_virt() function
  - Provides pre-configured kernel-level functions for GPIO access

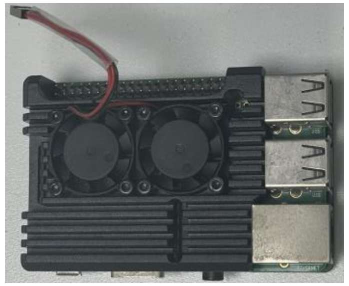
Implementation

- **Analysis framework**
  - Frama-C

- **Source-to-source transformation**
  - Auxiliary code is generated by JAVA
  - Peripheral driver porting is based on pseudo-TAs

Evaluation

- **Three questions**
  - (i) Does dTEE achieve better **expressiveness** and **rapid development** than existing approaches?
  - (ii) What is the **dCFG-based optimization improvement** performance?
  - (iii) What is the **overhead** of dTEE?



Raspberry Pi 3B+

Raspbian

Raspbian OS
(REE OS)

OP-TEE OS
(TEE OS)

Evaluation

- **Expressiveness**
  - Support four real-world apps and six microbenchmarks

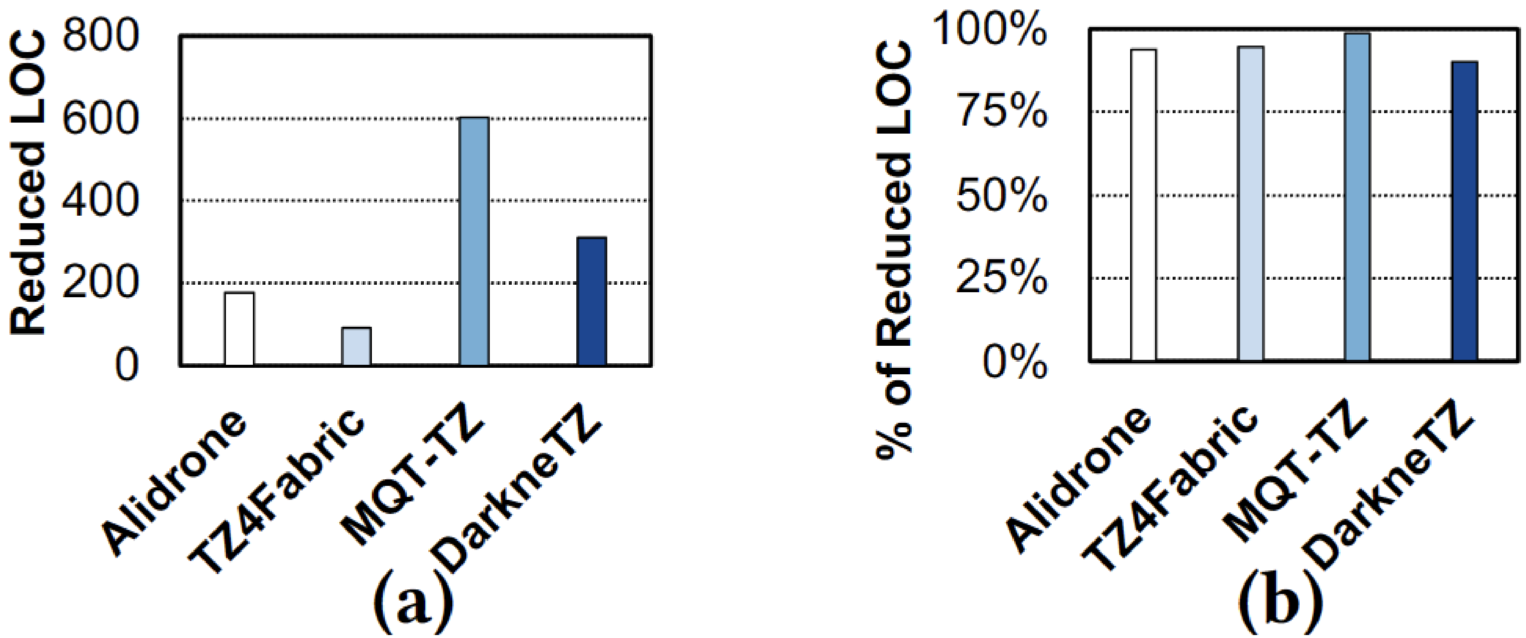
Category	Benchmark	Konstantin et al. [1]	Glamdering [2]	WATZ [3]	dTEE
Basic programming	Print	✓	✓	✓	✓
	cJSON	✗	✓	✓	✓
	concat	✓	✓	✓	✓
Sensing	Blink	✗	✗	✗	✓
	Temperature	✗	✗	✗	✓
	Humidity	✗	✗	✗	✓
Re	[ICDCS'18] Alidrone	✗	✗	✗	✓

Why dTEE is better?

- dTEE's peripheral-oriented library porting mechanism

Evaluation

- **Reduced Lines of Code (LOC)**
  - Reduce more than 90% LoC in four real-world apps



- dTEE's D-Lang language provides abstractions for protecting sensitive operations and data

Evaluation

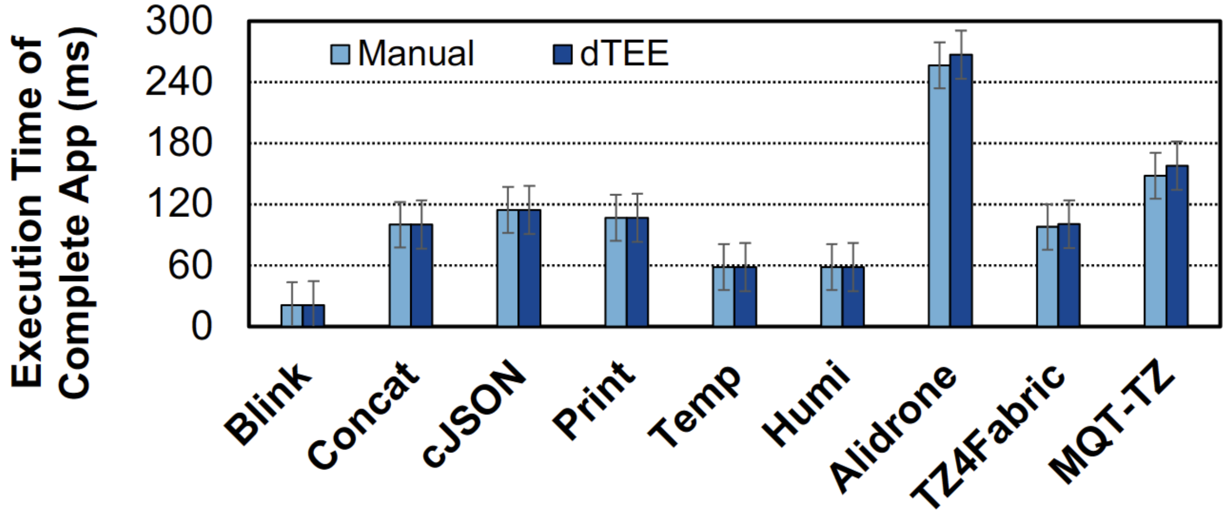
- **Performance improvement (speedtest1 benchmark)**
  - Reduce ~50% execution time
  - Reduce ~75% secure memory usage

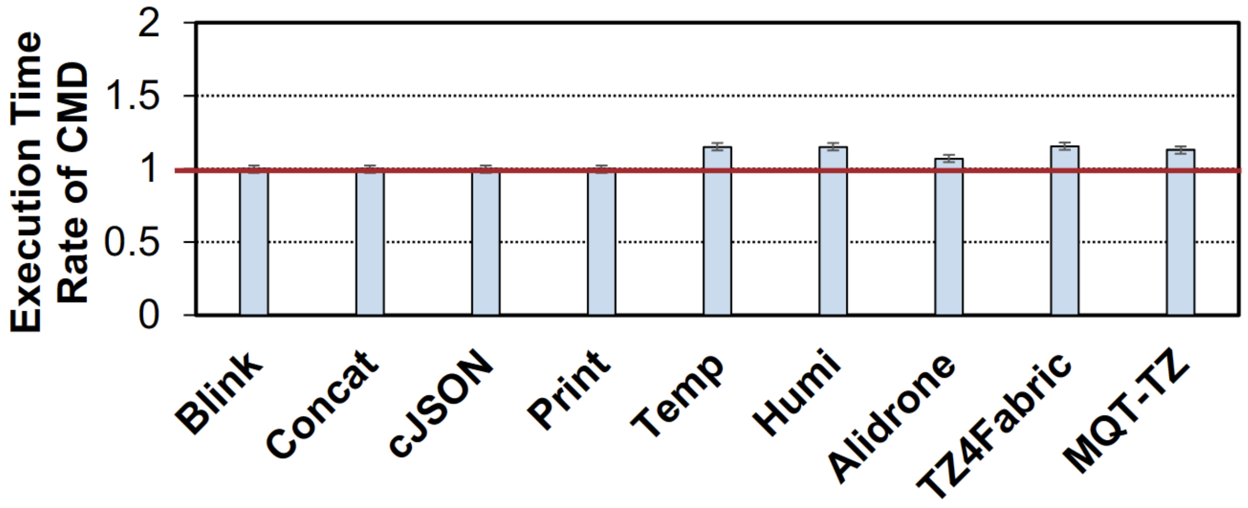
	Execution time		Memory usage	
	Overall	Switching	Overall	Wasm runtime
WATZ [32]	2,160 ms	/	12.10 MB	9.15 MB
dTEE (w/o dCFG opt)	1,217 ms	460.27 ms	2.14 MB	/
dTEE (w/ dCFG opt)	1,050 ms	293.03 ms	2.89 MB	/

- dTEE's dCFG optimization works well (~14%)
- dTEE utilizes a partitioning-based method which reduces memory usage

Evaluation

- **dTEE overhead**
  - Compared to using OP-TEE directly, dTEE incurs less than 6% overhead in terms of execution time





- dTEE results in a negligible overhead

dTEE

*A **d**eclarative approach to secure IoT applications using **TEE***

- ***D-Lang language***
- ***dCFG optimization***
- ***Peripheral-oriented lib porting***

Thank you for your attention!

Tong Sun, Borui Li, Yixiao Teng, Yi Gao, and Wei Dong

If you have any questions, please contact tongsun@zju.edu.cn

浙江大學 区块链与数据安全全国重点实验室
STATE KEY LABORATORY OF BLOCKCHAIN AND DATA SECURITY
ZHEJIANG UNIVERSITY

東南大學
SOUTHEAST UNIVERSITY

Declarative development language

- **D-Lang**

Statement types	Keywords	Comments
General	`FROM, FUNC`	The general format for D-LANG files. FROM{...} FUNC{...}
	`@DRIVER, INSERT_AFTER [], END_INSERT`	Configure the TEE environment.
Declarative Development of Trusted Logic	`INT_TZ, CHAR_TZ, FLOAT_TZ, STRING_TZ, STRUCT_TZ`	Declare secure variables which have not existed in the original apps.
	`TZ_genKey(), TZ_sign_rsa_sha1(), ...`	Built-in crypto functions, including enc/dec, sign/verify, hash, and random.
	`TZ_digitalWrite(), TZ_digitalRead(), ...`	Built-in peripheral APIs.
Tiered Protection	`TZ_STORE`	Permanently protect data based on the secure storage mechanism.
	`TZ_DATA_ONLY, TZ_DATA_CONF, TZ_DATA_INTG, TZ_DATA_ALL`	Tiered degrees for temporary protection.
	`TZ_FUNC`	Protect functions.
	`TZ_FUNC_SUB`	Substitute an original function with a new function.

```
FROM DroneAPP/main.c FUNC main {
// To permanently store the data in secure storage
  TZ_STORE gpsData;
}
```

Table 2: Three types for protecting libraries of IoT apps

Types	Portable libraries	Auto-transformable libraries	Manually-transformable libraries
Program logic	No changed	No changed	Changed
Transformation	No need	Need	Need
Conditions	All the functions supported by TEE, and no operations of Linux system calls.	The peripheral libs of IoT apps use mmap() operation to access GPIO.	The developers add new logic, or the libs increase the TCB significantly.
Examples	`Libquirc` [11], `Libnmea` [24]	`LibwiringPi` [19]	`Libopenssl` [35], `Libcrypto` [35]
Konstantin et al. [42]	✓	✘	✘
Glamdring [29]	✓	✘	✘
dTEE (Our paper)	✓	✓	✓