CS 2530 - Computer and Network Security
Project presentation

Leveraging Intel SGX to Create a Nondisclosure Cryptographic library

Mohammad H. Mofrad & Spencer L. Gray
University of Pittsburgh
Thursday, December 15, 2016

hasanzadeh@cs.pitt.edu & slg94@cs.pitt.edu

Preface

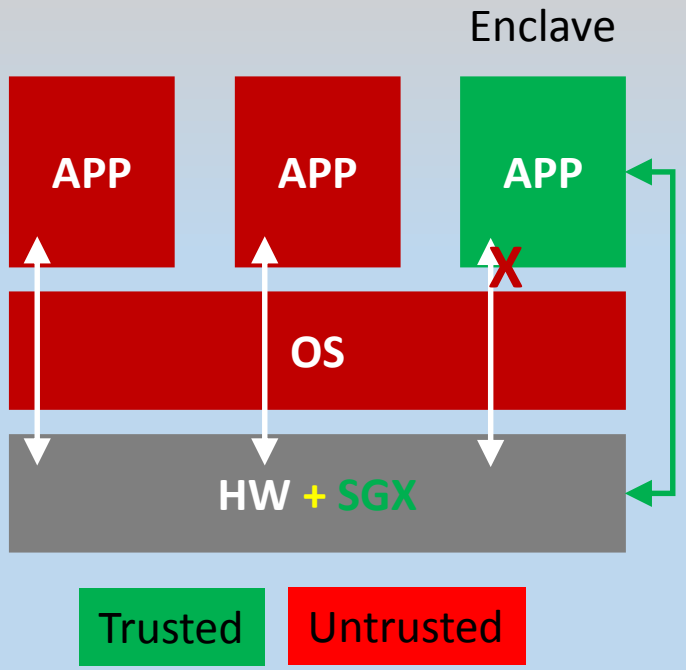
- Motivation
- Intel Software Guard Extension (SGX)
- SGX simulator
  - Linux SGX SDK
  - Project details
- Cryptographic Algorithms
  - Key generation
  - Hashes
    - SHA 256
    - HMAC SHA 256
  - Encryption
    - AES ECB
    - AES CBC
- Discussion
- Future work

Motivation

- Threat Model
  - An adversary has complete access to the OS
  - No access to the hardware of the machine
- Need a way to keep important data confidential under this model
- Solution: a cryptographic library inside trusted hardware
  - Cryptographic keys never leave the enclave

Introducing the Intel® SGX

- Intuition: *Computing devices are not trustworthy*
- Inspired by Trusted Platform Module (TPM)
  - TPM implemented the context of software attestation using hardware modules
- Enclaves are isolated regions of memory for code and data
  - Encrypted Enclave Page Caches (EPCs) in RAM
- In SGX only CPU is trusted
  - Added 18 new ISA Instructions
  - Supervisor instructions: ECREATE, EINIT, …
  - User instructions: EENTER, EEXIT, …
- Features:
  - Reduced attack surface
  - Attestation (local/remote) & Sealing



Related Work

- SGX1 (Initial release)
  1. Innovative instructions and software model for isolated execution
     - **Introducing the new instruction set**
  2. Using innovative instructions to create trustworthy software solutions
     - **One Time Password (OTP), Secure Video Conferencing (SVC), and etc. case studies**
  3. Innovative technology for CPU based attestation and sealing
     - **How to add SGX into Software lifecycle**
- SGX2 (Latest release)
  1. Intel SGX support for dynamic memory management inside an enclave
     - **Added support for dynamic memory management**
     - **Added support for changing permissions**
     - **Secure exception handling**
- Applications:
  - Cloud, Network Function Virtualization (NFV), Containeraization

SGX Emulator

- Linux SGX SDK from **01.org** INTEL OPEN SOURCE .org
  - Code: C and C++
  - 
    - Github repository at https://github.com/01org/linux-sgx
  - Supported OS
  - 
    - Ubuntu 14.04-LTS 64-bit
  - References
    - Intel SGX SDK for Linux OS programming reference
    - 
    - Intel SGX SDK for Linux OS developer reference
    - Intel Developer Zone
- Our implementations servers as an example in SampleCode directory of linux-sgx
  - **CryptoEnclave** project
  - 
    - Github repository: https://github.com/hmofrad/linux-sgx/
  - Algorithms from libtomcrypt https://github.com/libtom/libtomcrypt
    - 
    - **Hashes**: SHA 256 and HMAC SHA 256
    - **Symmetric key**: AES ECB and CBC with 128|192|256 key lengths

Cryptographic Library

- We chose LibTomCrypt
  - Designed to be extremely modular
  - Easy to take only the algorithms we needed
  - Minimize our trusted computing base

- Other options explored: OpenSSL and Crypto++
  - OpenSSL was designed to be fully downloaded
  - Crypto++ was very unorganized (all files were in 1 directory)

Implementation Details

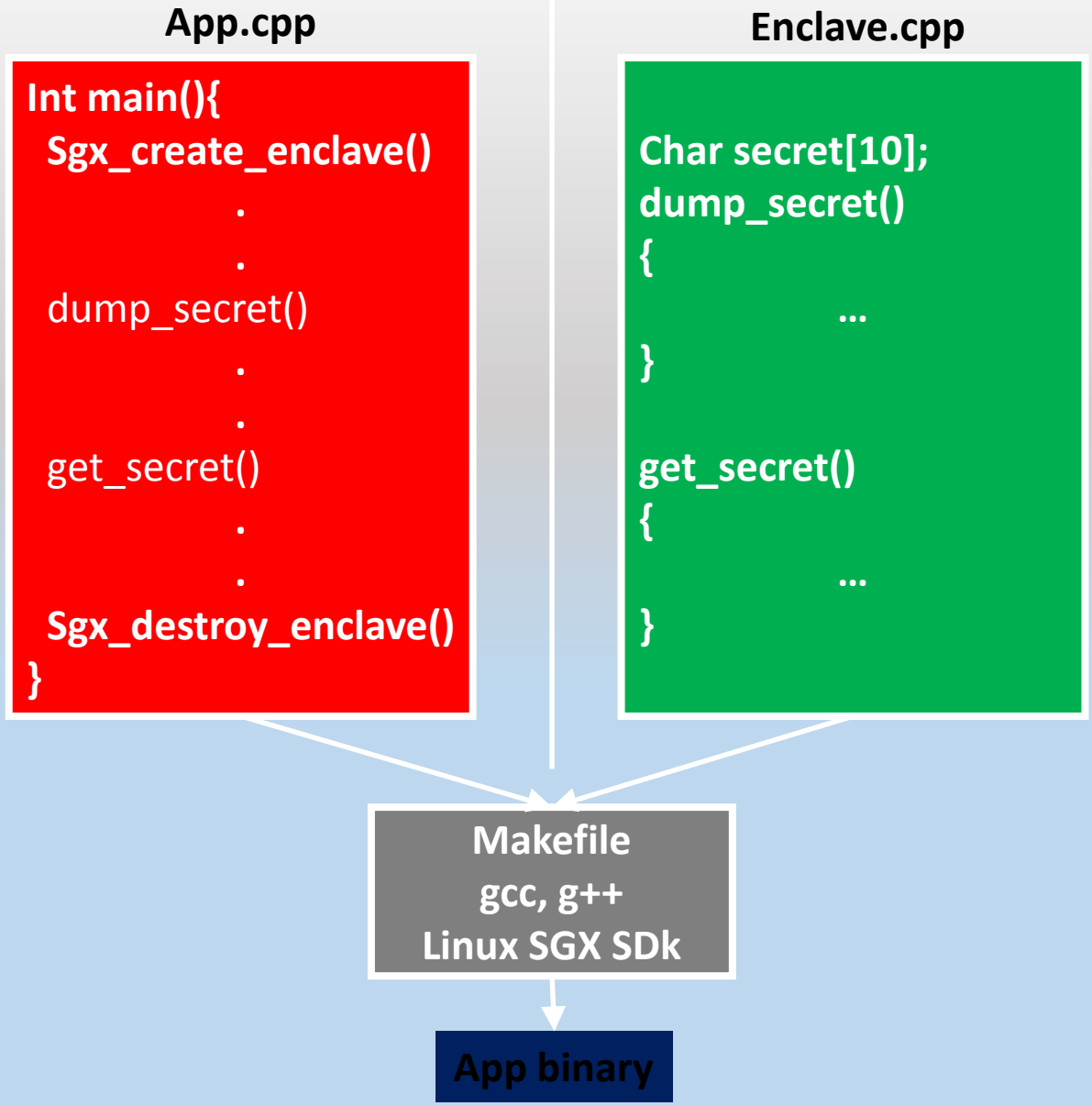
- Ubuntu 14.04 Virtual Machine
  - Architecture: 64-bit virtual Machine
  - RAM: 2GB
  - Disk: 8GB

- Crypto Enclave project
  - (**~500**) Application code
    - Command line
    - I/O interface
    - Utility functions
  - (~2300) Enclave implementation
    - (~1600) Modified libtomcrypt borrowed codes
    - (**~700**) Enclave crypto wrappers for crypto operations
  - (**~50**) Enclave.edl
  - Enclave.config.xml
  - Makefile

Sample Scenario

Untrusted code & data

Trusted code



Key generation

- Symmetric keys- just need random bytes
  - Depends on a good pseudo random number generator
  - Poor key generation leads to poor security
- Asymmetric keys- based upon “fancy” math
  - Still depend on random numbers!
  - If the base random numbers can be predicted, private keys can be easily computed
- The SGX already has key generation functions built in
- **sgx_read_rand(key, keylen)**

Hash Function

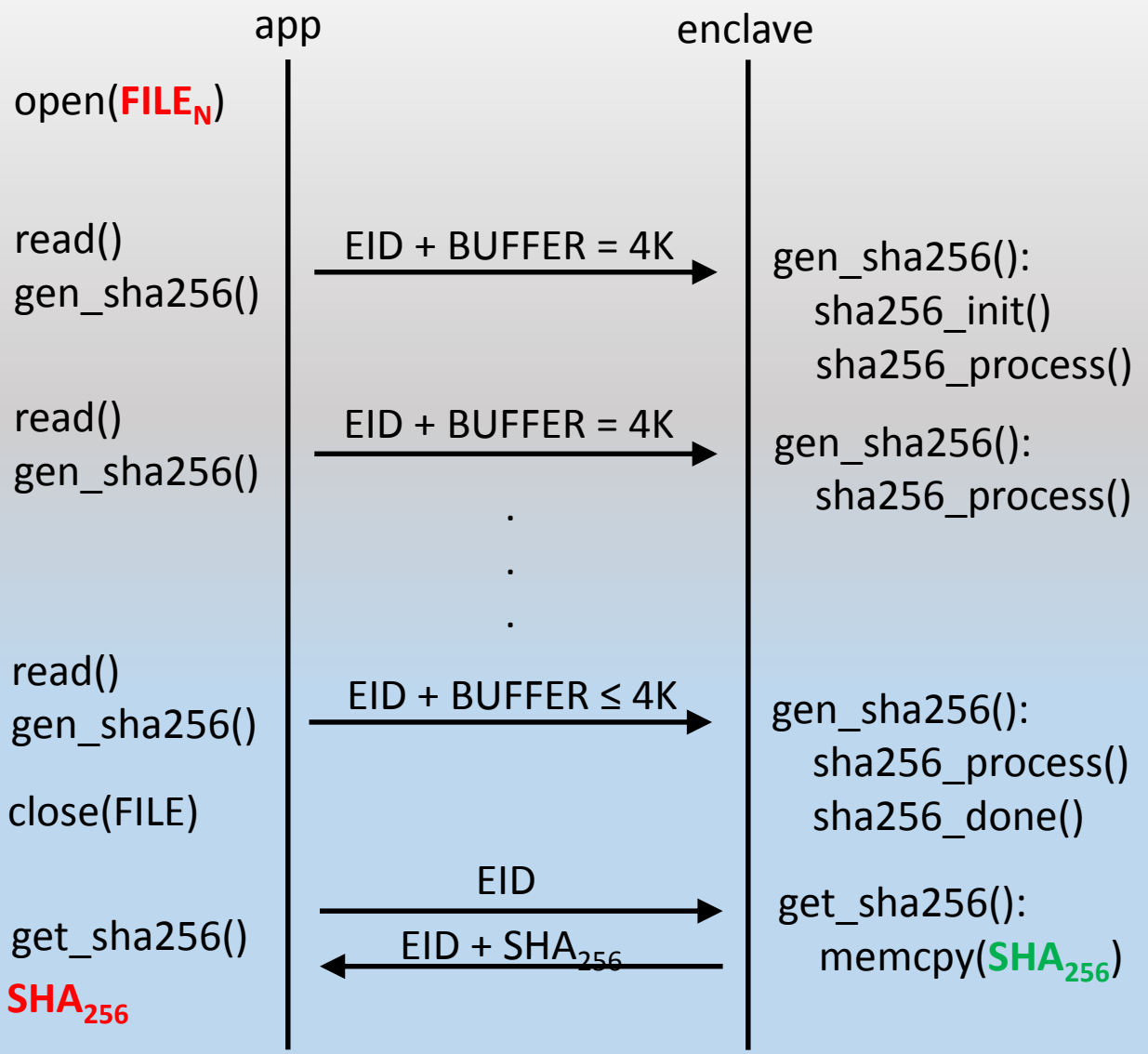
- Cryptographic primitive to provide integrity to data
- The base algorithm can be computed safely in user space
  - One-way function
  - No secrets are involved
- HMAC depends on hash function to operate
  - Keyed hash function (cannot be computed safely in user space)
  - Provides higher level of data integrity

Hash Implementation

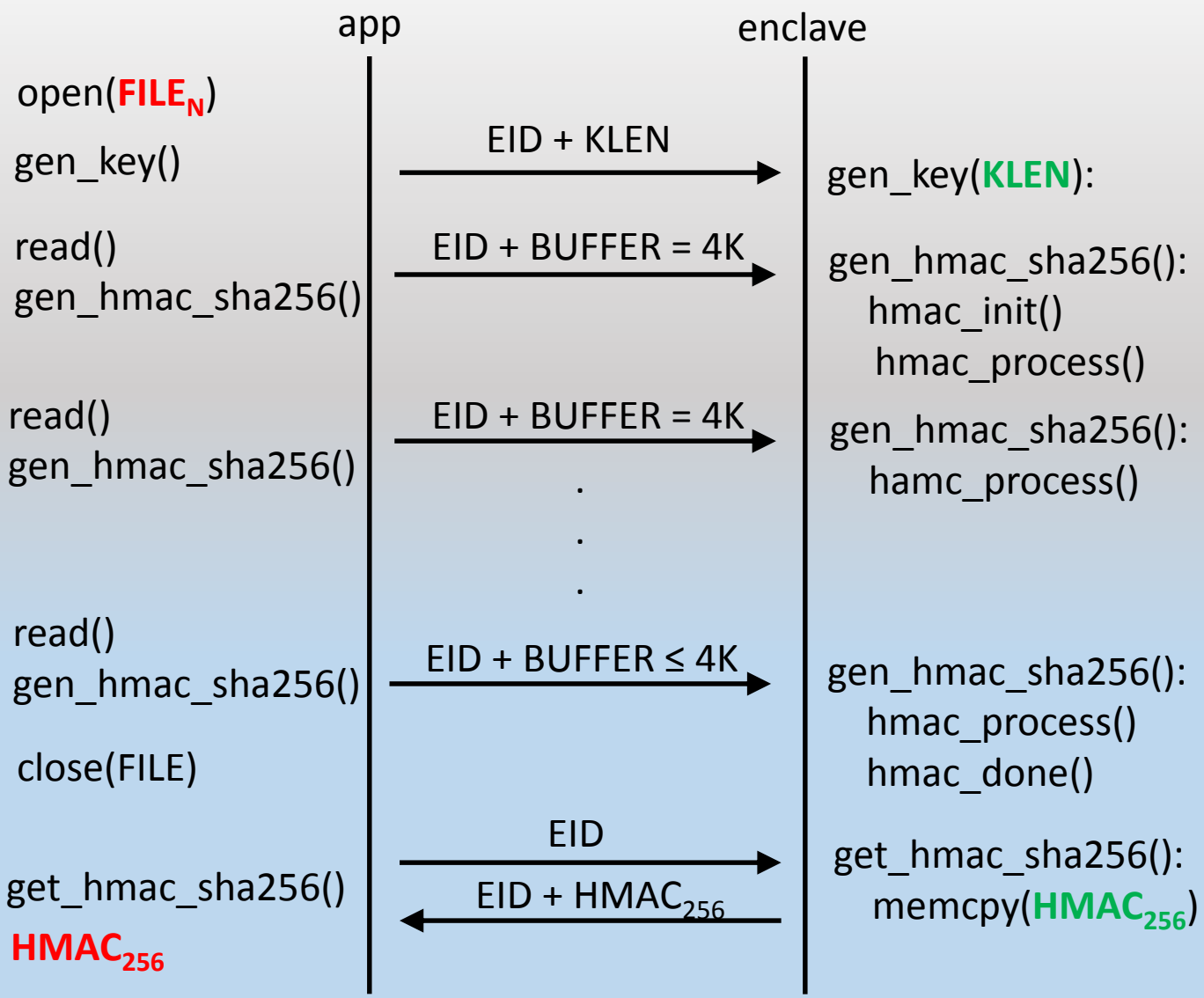
- LibTomCrypt's functions:
  - sha256_init() – Initializes the hash
  - sha256_process() - Takes input and uses it to build hash
  - sha256_done() – Writes out the final hash to an output buffer

- Our function gen_sha256() is essentially a wrapper for LibTomCrypt's implementation

SGX-enabled SHA 256

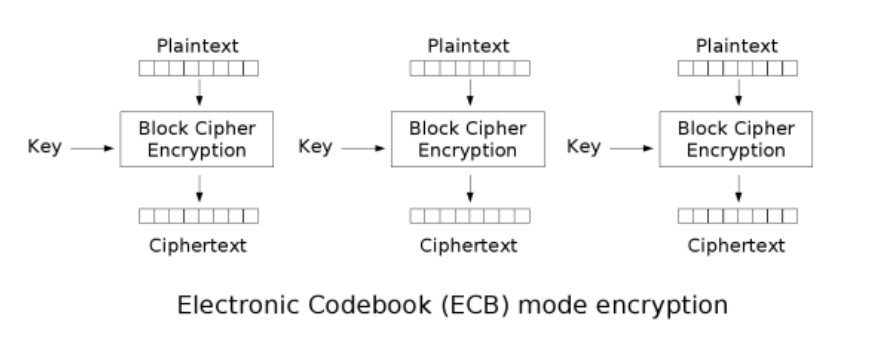


SGX-enabled HMAC SHA 256

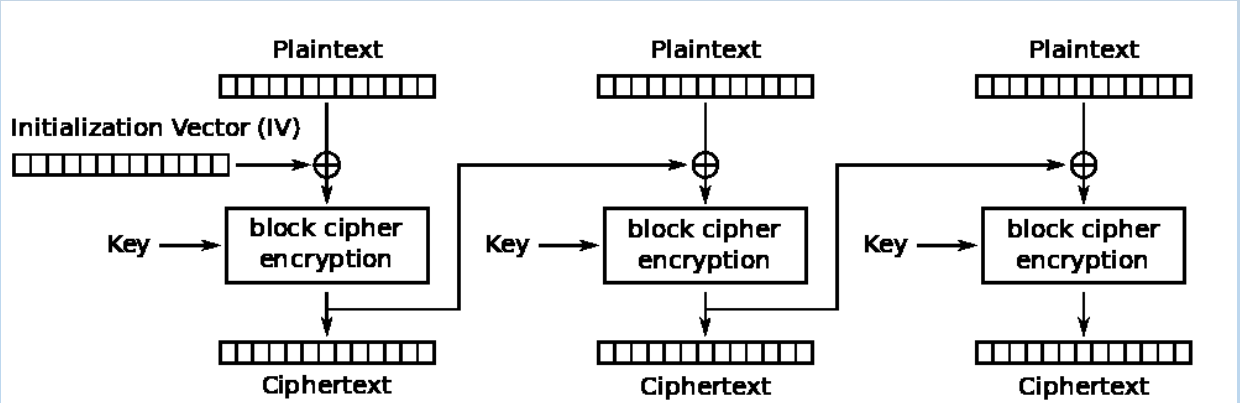


Symmetric Key Cryptography

- Uses same key for encryption and decryption
- Encrypts/decrypts data in fixed sized blocks (usually)
- Modes of block chaining
  - ECB-same block will always encrypt to the same cipher text
  - CBC-Cipher text is partially determined by previous block

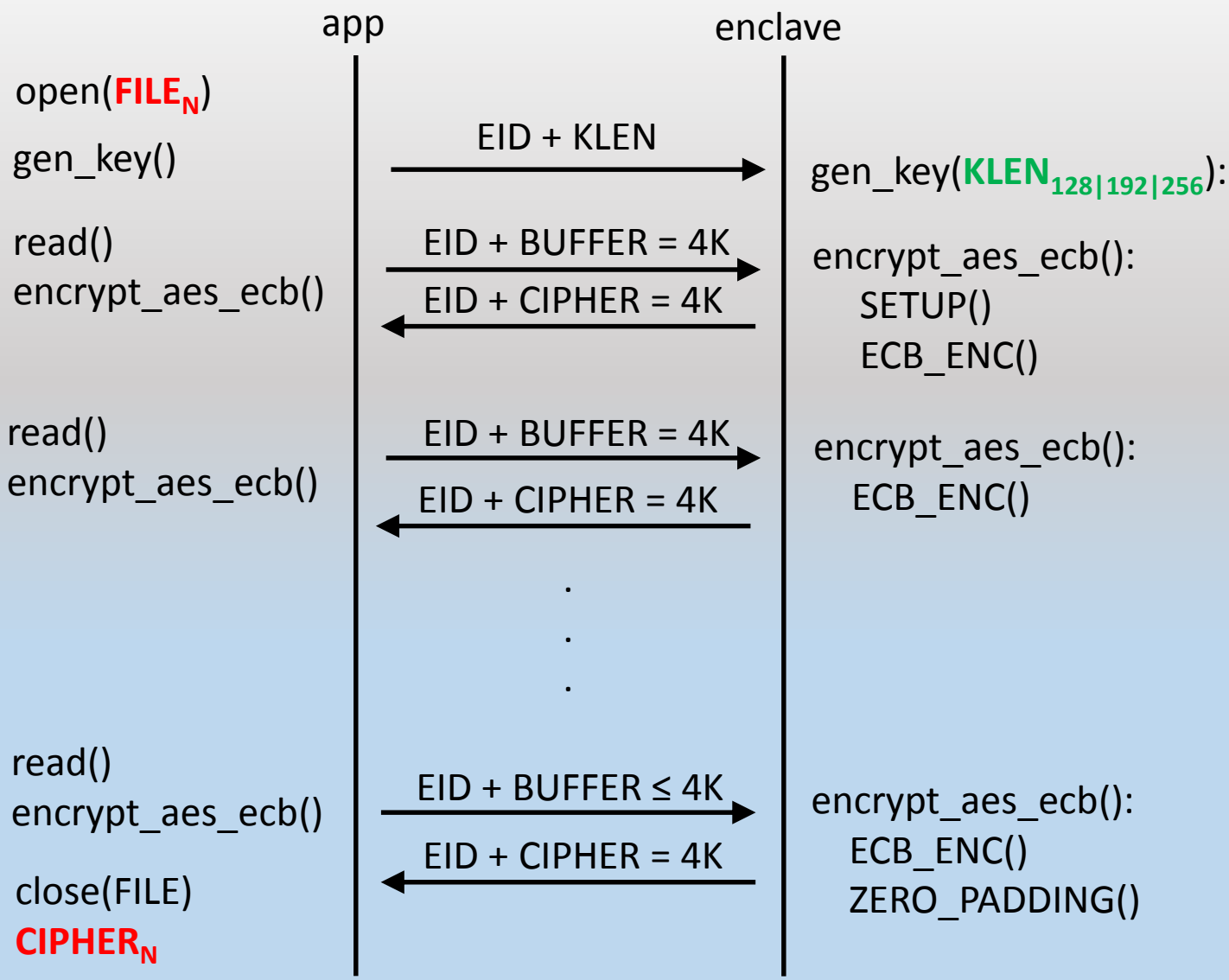


Electronic Codebook (ECB) mode encryption

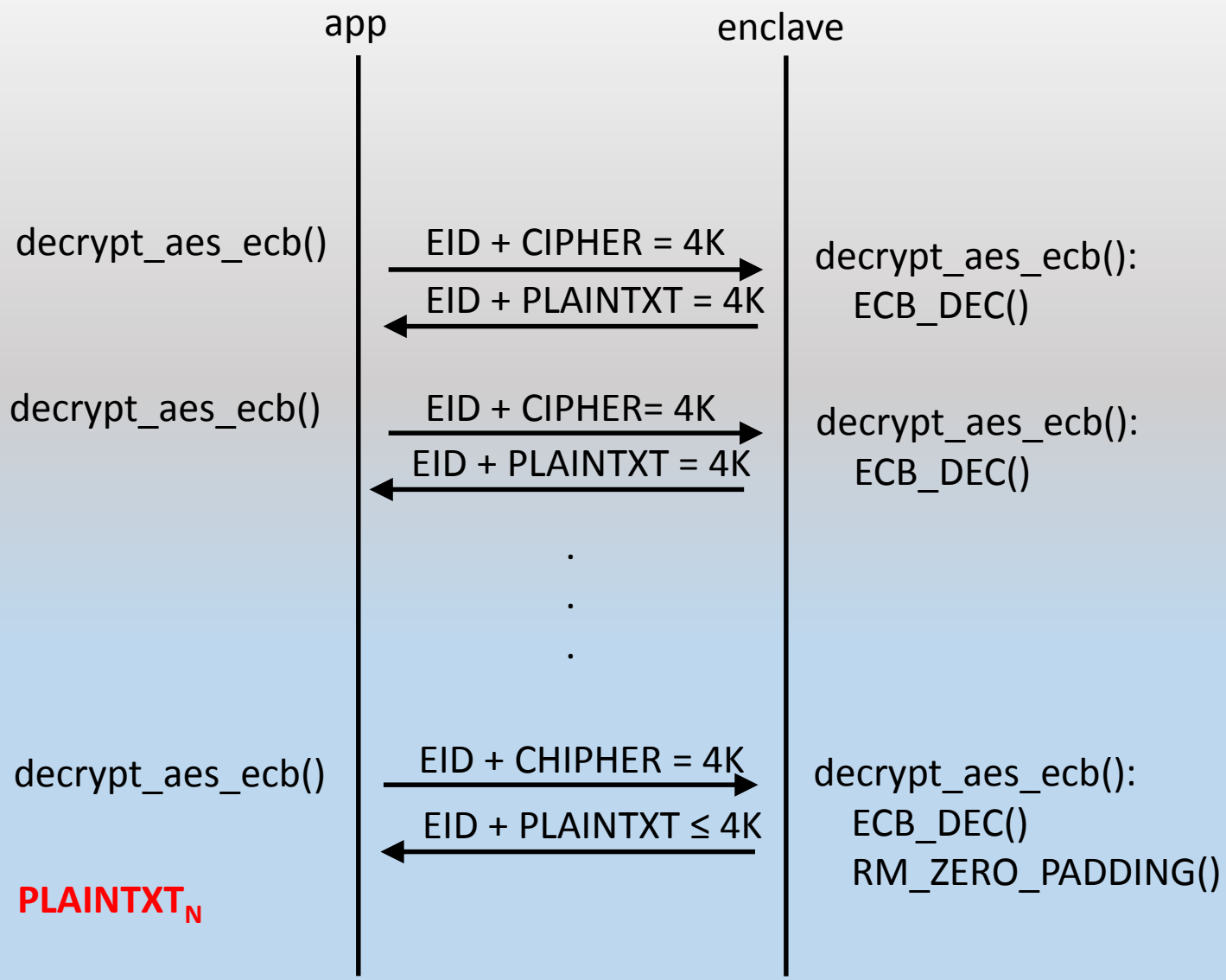


Cipher Block Chaining (CBC) mode encryption

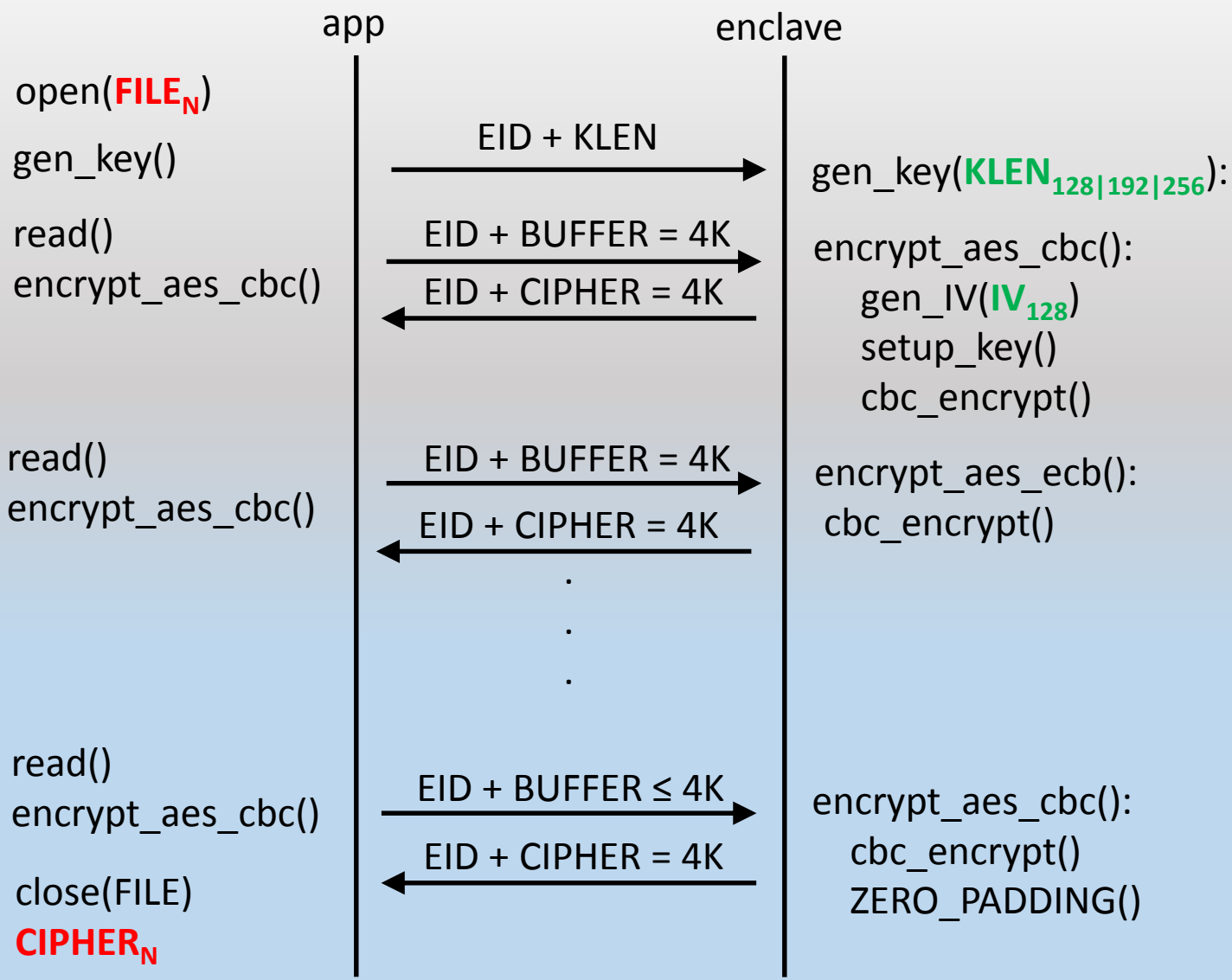
SGX-enabled AES ECB (Encryption)



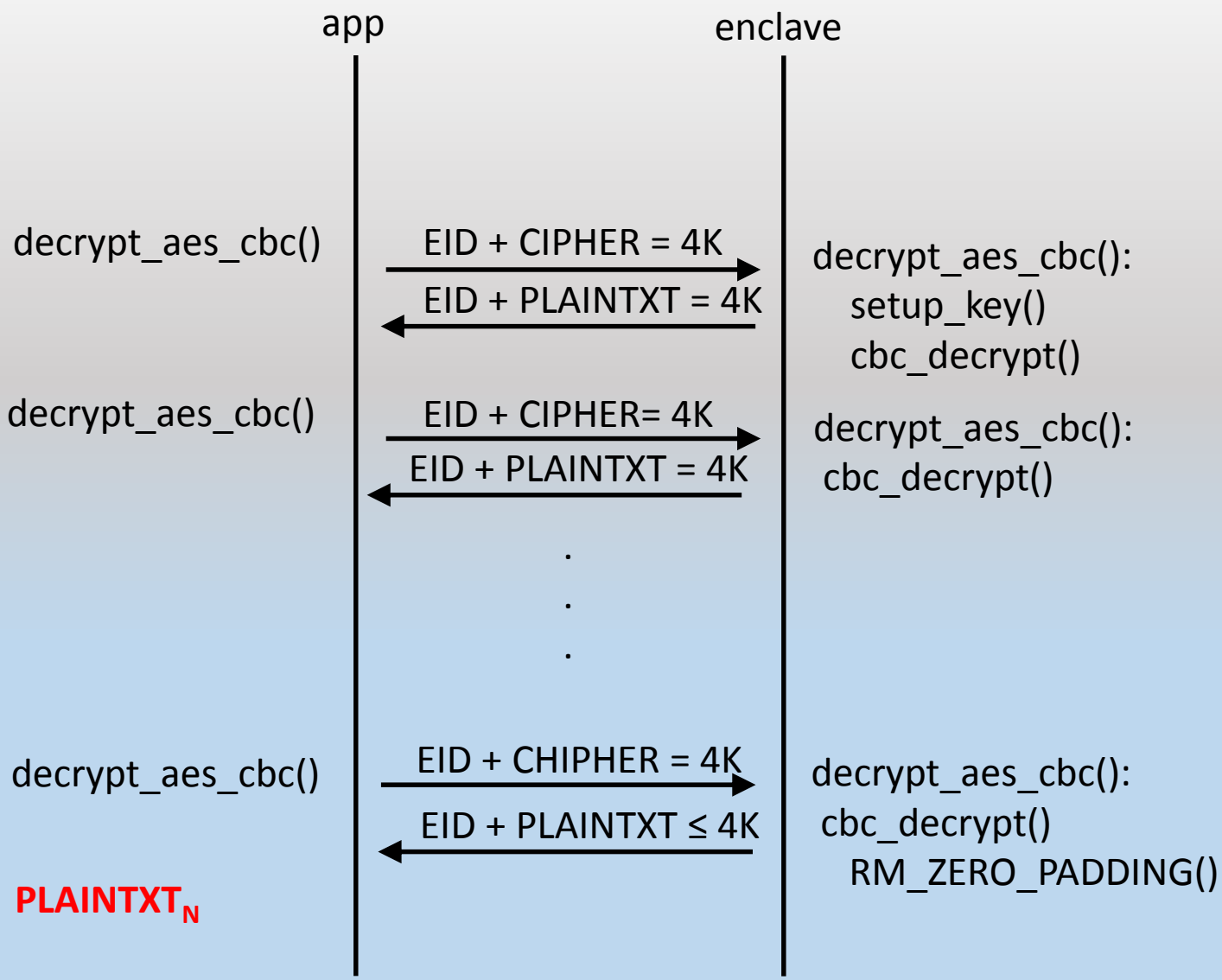
SGX-enabled AES ECB (Decryption)



SGX-enabled AES CBC (Encryption)



SGX-enabled AES CBC (Decryption)



Design Decisions

- Why we choose 4KB buffer size?
  - OS page size 4KB
  - Multiple of 16
  - Enclave Heap size is a multiple of 4
- Why we pass buffers of plaintext and ciphertext back and forth?
  - We don’t want to exhaust the enclave memory
  - In hardware mode, the Enclave heap size at max ~90MB
- Why we have a huge amount of I/O in this project?
  - The basic idea here is that if we encrypt the keys and data inside the enclave, no one can extract the keys and original data even if an attacker can access them!
  - The silver bullet of our project is that instead of bringing an insecure computation to the data, we bring the data to the enclave to securely perform the computations

Discussion

- Our implementation meets our goals under our threat model
  - Cryptographic keys never leave the enclave
  - A compromised OS cannot recover secrets except through brute force attacks

- The security of the keys relies on the security of the SGX
  - Discovery of attacks against SGX threaten our system
  - Still weak to hardware and side channel attacks

Discussion

- What drives Intel's incentive to launch SGX
  - For the good of the users i.e. adding another level of security on top of OS
  - For the good of its market i.e. software that are SGX-enabled
    - Need to accept the Intel license agreement
    - Need to rely on Intel servers for remote attestation and sealing
    - Literally, Intel is putting itself in the middle of software lifecycle
- Small Trusted Code Base (TCB)
  - We only store keys, and hashes and nothing else
  - We barely need less than 100KB of memory to handle a user session no matter what is the file size (~200)
  - The computation are done inside the enclave
  - Keys may never leave the enclave

Future Work

- Current uses are limited!
- Expand the existing library
  - Public key cryptography (RSA, ECC, Diffie Hellman)
  - Pseudo random number generating algorithms
- Protocol to allow two enclaves to share keys
  - Allow separate users to safely communicate using keys sealed in enclaves