Hardening Application Security using Intel SGX

Max Plauth, Frederik Teschke, Daniel Richter, and Andreas Polze

Operating Systems & Middleware Group
Hasso Plattner Institute at University of Potsdam, Germany
Motivation

- data security: encryption
  - securely **transmitting** data
  - secure data **processing**
Motivation

- trusted computing approaches
  - Trusted Platform Modules
  - ARM’s TrustZone
  - Intel’s Software Guard Extensions (SGX)
Motivation

Intel SGX

- not widely utilized
- high complexity
  - needs profound knowledge in fields of cryptography, operating systems, and hardware design

- our goal: practical perspective, approaching the challenges of trusted computing from a software engineer’s point of view
  - helper library overcoming hurdles of integrating SGX API with code base
  - case study: porting existing applications to run inside SGX enclaves
Background
Software Guard Extensions

- implemented entirely CPU hardware
- exposed by instruction set extensions

**Enclave**
- encrypted, process-like memory regions
- code, stack + heap memory
- decrypt memory when loading into CPU cache
- protected from being accessed by privileged code
  - even from code in *System Management Mode* (SMM) and *Direct Memory Access* (DMA)
Background

Intel SGX Enclaves
Background

Intel SGX Enclaves

![Diagram showing the structure of Intel SGX Enclaves and the access levels of various systems.]
Background

Intel SGX Enclaves

- untrusted operating system
  - scheduling & memory allocation
  - setting up an enclave

- Enclave attestation
  - each SGX-capable CPU has embedded cryptographic private key
  - use this key + special group signature schema to attest state of Enclave
  - remote attestation with “architectural enclaves”
Background

Intel SGX Enclaves

- code in Enclaves may not execute certain calls
  - calls that may case a VMEXIT, input/output instructions, calls requiring change of privilege levels
- multiple threads
  - number must be statically defined
- maximum enclave size (memory) must be defined before initialization of Enclave
Enclave Development
Enclave Development

Software Development Kit

▪ SDK provided by Intel
  ▪ Windows + Linux
  ▪ language support: C and C++
  ▪ interface definition: Enclave Definition Language (EDL)
  ▪ trusted library: helper functions
    ▪ subset of standard C library (e.g. without file input/output)
    ▪ random number generation, cryptographic primitives, key exchange and data sealing
  ▪ debug mode: all protection mechanisms disabled
  ▪ simulation mode: if SGX hardware is absent
  ▪ complete authoring chain
Enclave Development

Enclave Definition Language

- trusted section (**E-call**, enclave calls)
  - proxies are generated for the untrusted wrapper
- untrusted section (**O-call**, outside of enclave calls)
  - proxies are generated for the enclave
- parameter marshalling
  - direction of data flow
  - pass-by-value (recommended) & pass-by-reference
  - annotations (size, sizefunc, count) for pointer arguments
enclave {
  trusted {
    public void add_secret(int secret);
    public void print_secrets();
    public void test_encryption();
    public void set_key([in, size=128] uint8_t *key);
  }
  from "sgx_lib.edl" import *;
}

untrusted {
}
}
SGX Helper Library
SGX Helper Library

- enable easier and faster prototyping
- contains scripts and wrapper functions to make working with the SDK easier:
  - Generation of O-call Proxies
  - Error Code Handling
  - Easy-to-Use Encryption
  - Transparent Encryption of Input/Output
- available for public use: https://github.com/ftes/sgx-lib
SGX Helper Library

Generation of O-Call Proxies

- O-call proxies – shim inside Enclave to proxy calls to outside world
- provide trusted functions with original signature
  - e.g. for directly linking against the C library implementation
  - different signatures in EDL for calls with return values
- automate process of defining these proxies
SGX Helper Library

Generation of O-Call Proxies

- SDK proxies deal with parameter handling
- untrusted library proxy delegates to the C library
SGX Helper Library

Error Code Handling

- utility function to check return values
- looking up error codes scraped from Intel SDK’s `sgx_error.h`

```c
SGX_ERROR_UNEXPECTED = SGX_MK_ERROR(0x0001), /* Unexpected error */
SGX_ERROR_INVALID_PARAMETER = SGX_MK_ERROR(0x0002), /* The parameter is incorrect */
SGX_ERROR_OUT_OF_MEMORY = SGX_MK_ERROR(0x0003), /* Not enough memory is available to complete this operation */
SGX_ERROR_ENCLAVE_LOST = SGX_MK_ERROR(0x0004), /* Enclave lost after power transition or used in child process created */
SGX_ERROR_INVALID_STATE = SGX_MK_ERROR(0x0005), /* SGX API is invoked in incorrect order or state */

SGX_ERROR_INVALID_FUNCTION = SGX_MK_ERROR(0x1001), /* The ecall/ocall index is invalid */
SGX_ERROR_OUT_OF_TCS = SGX_MK_ERROR(0x1002), /* The enclave is out of TCS */
SGX_ERROR_ENCLAVE_CRASHED = SGX_MK_ERROR(0x1003), /* The enclave is crashed */
SGX_ERROR_ECALL_NOT_ALLOWED = SGX_MK_ERROR(0x1004), /* The ECALL is not allowed at this time, e.g. ecall is blocked by the OS */
SGX_ERROR_OCALL_NOT_ALLOWED = SGX_MK_ERROR(0x1005), /* The OCALL is not allowed at this time, e.g. ocall is not allowed during TCS */
SGX_ERROR_STACK_OVERRUN = SGX_MK_ERROR(0x1006), /* The enclave is running out of stack */

SGX_ERROR_UNDEFINED_SYMBOL = SGX_MK_ERROR(0x2000), /* The enclave image has undefined symbol. */
SGX_ERROR_INVALID_IMAGE = SGX_MK_ERROR(0x2001), /* The enclave image is not correct. */
SGX_ERROR_INVALID_ENCLAVE_ID = SGX_MK_ERROR(0x2002), /* The enclave id is invalid */
SGX_ERROR_INVALID_SIGNATURE = SGX_MK_ERROR(0x2003), /* The signature is invalid */
SGX_ERROR_NDEBUG_ENCLAVE = SGX_MK_ERROR(0x2004), /* The enclave is signed as product enclave, and can not be created as an */
SGX_ERROR_OUT_OF_EPCR = SGX_MK_ERROR(0x2005), /* Not enough EPCR is available to load the enclave */
```
Easy-to-Use Encryption

- some of SDK’s cryptography functions are cumbersome to use
  - size of encrypted/sealed data not trivial to determine

- extensive wrapper for encryption

```c
uint32_t get_sealed_data_size(uint32_t
  plaintext_data_size);
int seal(void* plaintext_buffer, uint32_t
  plaintext_data_size, sgx_sealed_data_t*
  sealed_buffer, size_t sealed_data_size);
int unseal(void* plaintext_buffer, uint32_t
  plaintext_data_size, sgx_sealed_data_t*
  sealed_buffer);

uint32_t get_encrypted_data_size(uint32_t
  plaintext_data_size);
int encrypt(void* plaintext_buffer, uint32_t
  plaintext_data_size, sgx_lib_encrypted_data_t
  * encrypted_buffer, sgx_aes_ctr_128bit_key_t*
  key);
int decrypt(void* plaintext_buffer, uint32_t
  plaintext_data_size, sgx_lib_encrypted_data_t
  * encrypted_buffer, sgx_aes_ctr_128bit_key_t*
  key);
```
SGX Helper Library

Transparent I/O Encryption

- transparent de- & encryption of input/output data – protects data operated on by legacy code without requiring any code modifications
- intercepting calls to C library for file input/output
- choose desired security level (at compile time)
  - no security: plain file input/output
  - encryption with custom key: use of symmetric encryption key
  - data sealing (default): seals all input/output to the Enclaves identity
Case Study: KISSDB
Case Study: KISSDB

- hardening security of existing database management system using SGX
- interesting target for trusted computing
  - stored data may be sensitive, requiring protection from the infrastructure, provider or other tenants
- avoid excessively complex code: KISSDB
  - simple key/value store
  - implemented in plain C using only file I/O functions
  - https://github.com/adamierymenko/kissdb
- goal: protect data KISSDB operates on
Case Study: KISSDB

Design Decisions

- Enclave Design
  - only move application code into enclave
  - shim C library to utilize external host C library

- Scope of Enclaves
  - no concurrency in KISSDB – one enclave per database

- Decomposition
  - a single enclave used for all trusted functionality
  - KISSB not sub-divided into trusted & non-trusted function and does not support data processing

- Unencrypted Metadata
  - metadata (header, hash tables) is not protected
Case Study: KISSDB

Design Decisions

- **Iterators**
  - allows to iterating through all values, several iterators per database in parallel
  - iterator data (page number and offset) stored outside of the Enclave

- **Encryption vs. Sealing**
  - sealing: encryption with key derived from Enclaves identity
  - (user) encryption: empower user to specify key
Case Study: KISSDB

Design Decisions

- hardened KISSSB architecture:

```
struct KISSDB {
    file
    hash_tables [...]
}
```
Case Study: KISSDB

Unresolved Issues

- **attestation & key-provisioning**
  - attest enclave’s identity and perform key exchange in a production setting

- **file integrity & freshness**
  - use of cryptographic mechanisms to ensure file integrity (e.g. monotonic counters provided by SDK)

- **cryptographic hash function**
  - KISSDB does not use cryptographic hash function

- **file layout**
  - deterministic file layout (known plain text attacks)
Related Work
Enclave Design Alternatives

- library operating system inside enclave
- minimal enclave size with external C library
- untrusted system calls with internal C library
Related Work

▪ application specific approaches
  ▪ Verifiable Confidential Cloud Computing (VC3)
  ▪ Secure Keeper

▪ general approaches
  ▪ Haven
  ▪ SCONE
  ▪ Software Partitioning Case Study
Summary

- Practical perspective: approaching the challenges of trusted computing in distributed scenarios from a software engineer’s point of view.
  - We provide a brief overview of the core aspects of SGX.
  - We present a helper library assisting developers in overcoming the hurdles of integrating the official SGX Software Development Kit (SDK) with their code base. [https://github.com/ftes/sgx-lib](https://github.com/ftes/sgx-lib)
  - In a case study, we demonstrate the steps necessary for porting existing applications to run inside SGX enclaves, using the KISSDB database as an exemplary application. [https://github.com/ftes/kissdb-sgx](https://github.com/ftes/kissdb-sgx)