Разработка системы защиты исполняемого кода с использованием технологии Intel SGX

А.М.Гладков, Р.К. Заводских, И.А.Левицкий

Введение

Наша задача – защита программного обеспечения

Актуальность: защита большинства ПО, не предназначенных для свободного распространения, может быть взломана, и ПО – использовано без надлежащей лицензии.

Постановка задачи

Существует множество способов обхода защиты:

Как всему этому противостоять?

Технология Intel SGX

Технология Intel Software Guard Extensions – набор инструкций ЦП, предоставляющих возможность приложению создавать **анклавы.**

Анклав – область в виртуальном адресном пространстве, защищенная от чтения и записи извне этой области, в том числе любыми другими процессами и ядру ОС.

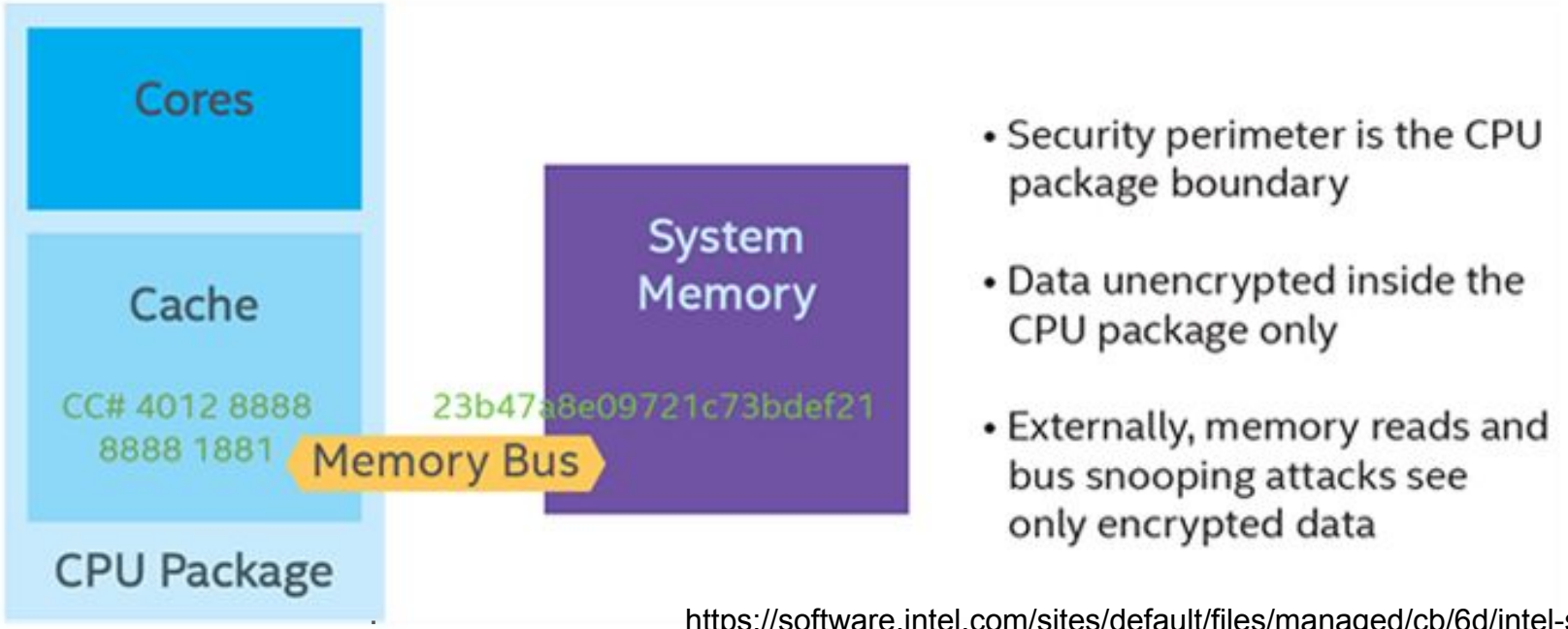
Плюсы Intel SGX

- Непрерывная защита даже в случае компрометации целостности BIOS, VMM, ОС и драйверов, когда нарушитель получил полное управление платформой
- Защита от попыток управления шиной, фальсификации данных памяти и атак "холодной загрузки"
- Аппаратные механизмы в ответ на проблемы удаленной аттестации для проверки целостности
- Создания с использованием стандартных средств разработки

https://software.intel.com/ru-ru/sgx

Intel SGX в картинках:

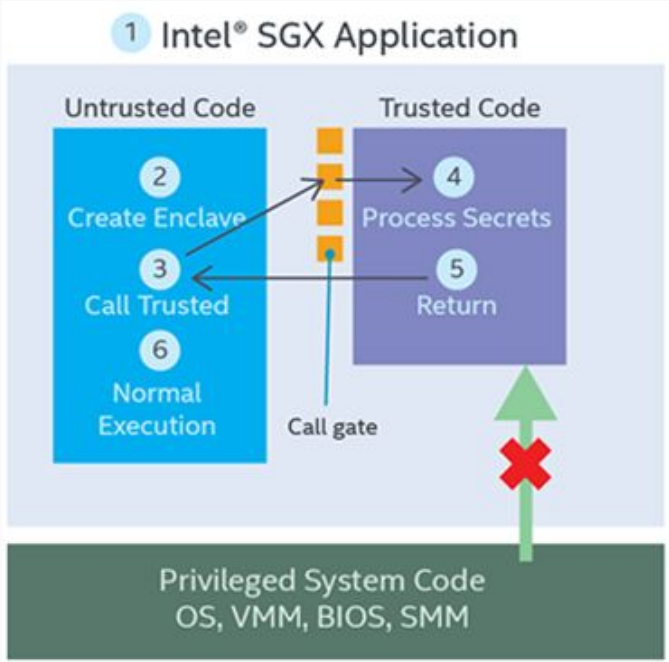
How Intel® Software Guard Extensions helps secure enclave data in protected applications



https://software.intel.com/sites/default/files/managed/cb/6d/intel-software-guard-extensions-tutorial-intel-sgx-foundation-fig02.png

Intel SGX в картинках:

Intel® Software Guard Extensions application execution flow.



1. App is built with trusted and untrusted parts
2. App runs and creates the enclave, which is placed in trusted memory
3. Trusted function is called, and execution is transitioned to the enclave
4. Enclave sees all process data in the clear; external access to the enclave data is denied
5. Function returns; enclave data remains in trusted memory
6. Normal execution resumes

https://software.intel.com/sites/default/files/managed/61/f1/intel-software-guard-extensions-tutorial-intel-sgx-foundation-fig03.png

Этапы создания анклава

Этапы создания анклава следующие:

1. **ECREATE**: Объявление будущего анклава. выделяется первая страница,
2. **EADD**: Добавление страниц с кодом программы и данными,
3. **EEXTEND**: Вычисление хэш-функции от загруженных данных, по которой можно идентицифицировать код в анклаве,
4. **EINIT:** После этого действия возможность получить извне доступ к памяти анклава напрямую пропадает. Теперь в анклав можно загружать данные, которым необходимо обеспечить защиту.

Этапы создания анклава

Этапы создания анклава следующие:

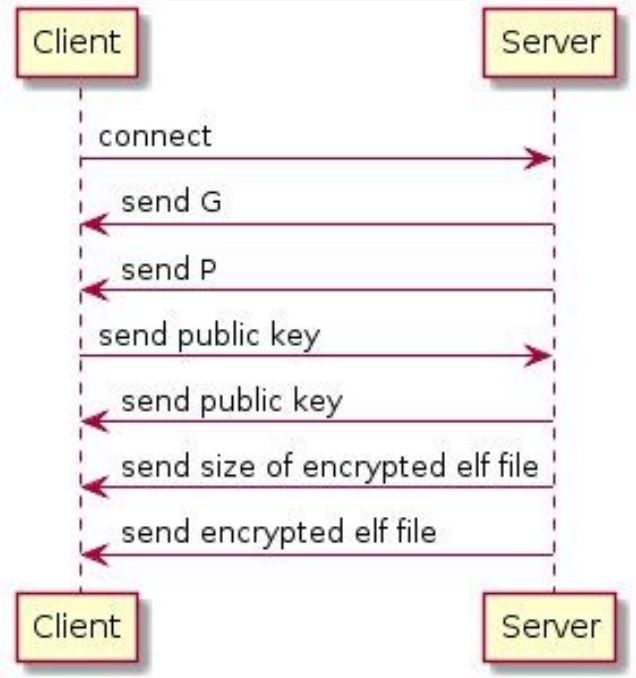
1. **ECREATE**: Объявление будущего анклава. выделяется первая страница,
2. **EADD**: Добавление страниц с кодом программы и данными,
3. **EEXTEND**: Вычисление хэш-функции от загруженных данных, по которой можно идентицифицировать код в анклаве,
4. **EINIT:** После этого действия возможность получить извне доступ к памяти анклава напрямую пропадает. Теперь в анклав можно загружать данные, которым необходимо обеспечить защиту.

Здесь код еще открыт!

Решение задачи

Предлагается клиент-серверная архитектура

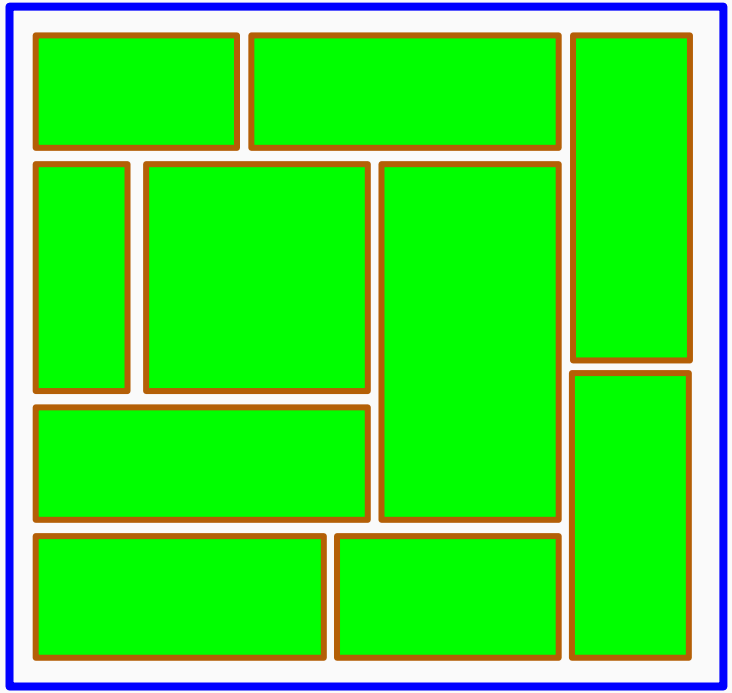
- Клиент проходит авторизацию, создает анклав,
- Сервер разработчика ПО формирует с Клиентом шифрованный канал по схеме открытого распределения ключей,
- Сервер зашифровывает секретный код сессионным ключом и передаёт его Клиенту
- Клиент расшифровывает код в анклаве. Программе передается управление.



Принцип работы

– Весь код программы

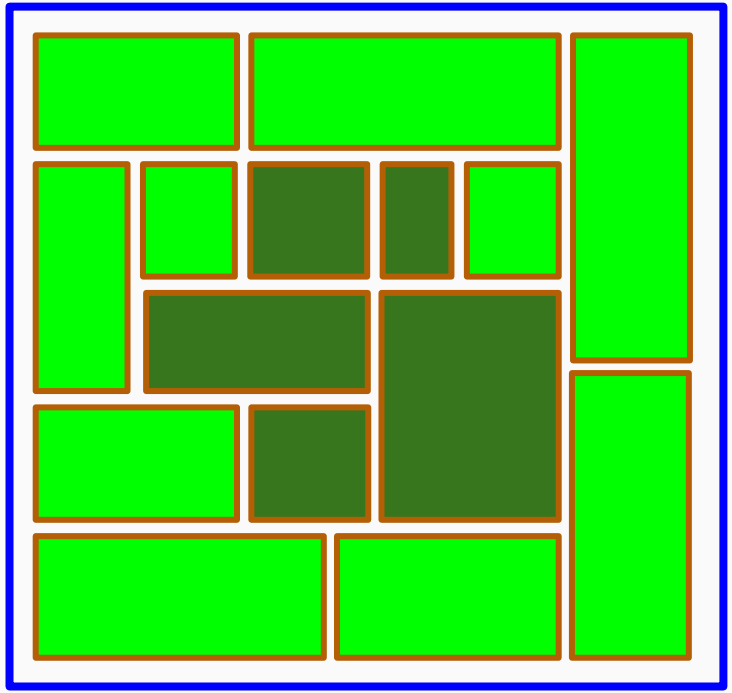
– часть программы (группа логически связанных функций)



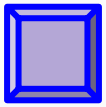
Принцип работы

Разработчик, для использования разработки, разделяет программу:

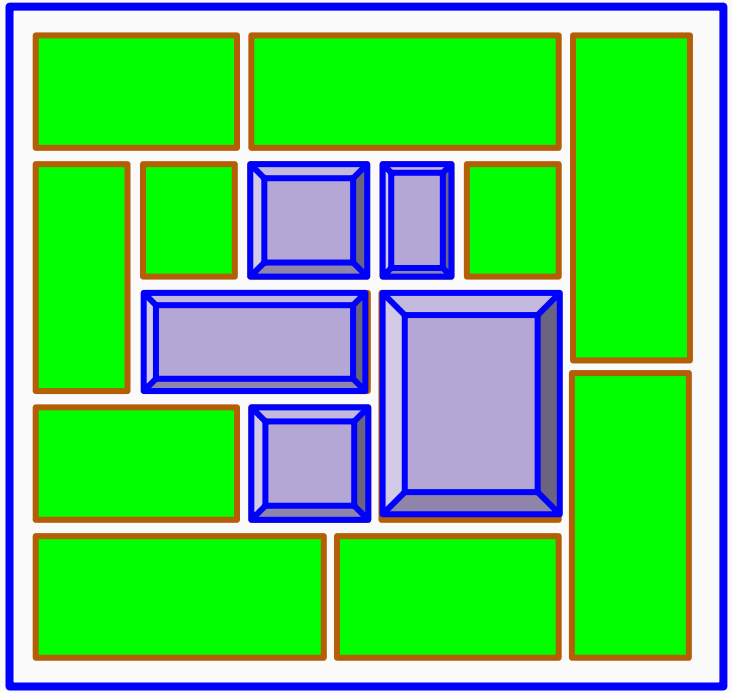
- Недоверенную часть
  - неосновные функции, работающие с системными вызовами
  - исполняются вне анклава
- Доверенную часть
  - основная смысловая нагрузка
  - будет секретной, исполняется в анклаве
  - нет системных вызовов



Принцип работы

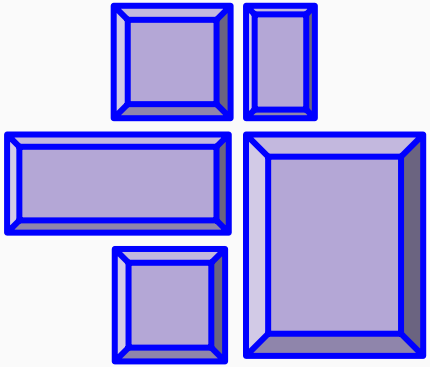


– доверенная часть кода будет помещена в анклав. Для “общения” с недоверенной частью используются функции-мосты.



Принцип работы

Клиент скачивает недовереную часть программы стандартным способом.



Зашифрованный код доверенной части отправляется Клиенту только после успешной авторизации.

Реализация: пересылка кода

После успешной авторизации, Клиент запускает анклав.

Анклав

- хранит ключи при реализации протокола Диффи-Хеллмана,
  - Приватный ключ генерируется функцией sgx_read_rand
  - Для длинной арифметики используется библиотека MPIR
- получает через недоверенную часть зашифрованный elf-файл,
- после загрузки расшифровывает код.
  - используется шифр AES в режиме электронной кодовой книги (ECB)

Реализация: исполнение кода

До запуска скачанной программы соответствующие страницы должны быть помечены на исполнение с помощью системного вызова mprotect.

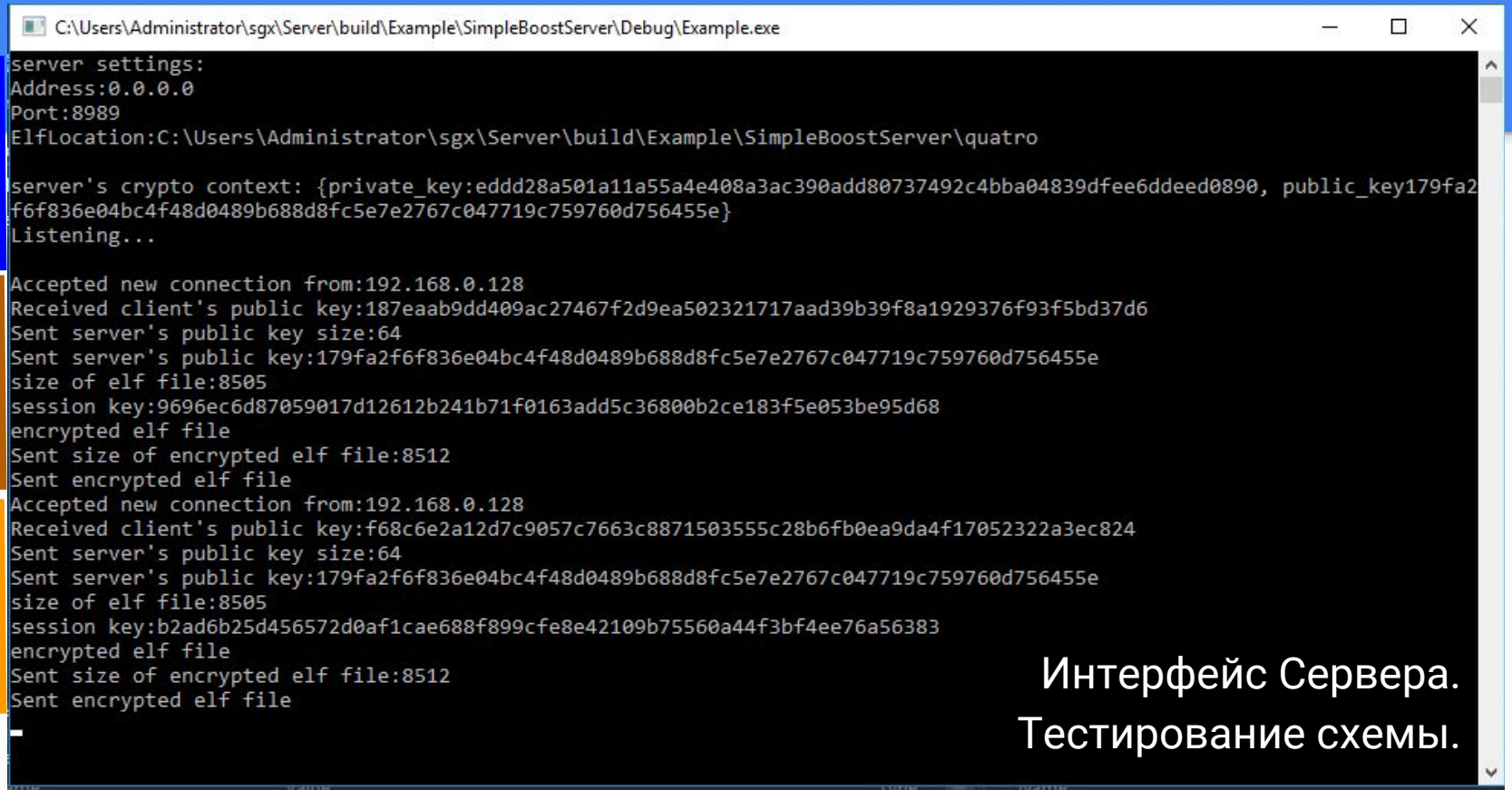
Все скачанные коды суть elf-файлы. Поиск по имени необходимых функций в elf-файле производится написанной вспомогательной функцией elf_parser.

Результаты

Запуск

Отсылка

Отсылка



Интерфейс Сервера.

Тестирование схемы.

Результаты

Интерфейс Клиента. Тестирование схемы, пробный запуск ПО.

```
The project has been built in debug simulation mode.
roza@roza-S301LA ~/Downloads/SGX/sgxsdk/SampleCode/RoZaEnclave/Client $ ./app 192.168.0.100 Quadripler 4
(App/App.cpp:343): A(Client public key): 187eaab9dd409ac27467f2d9ea502321717aad39b39f8a1929376f93f5bd37d6
(App/App.cpp:347): B(Server public key): 179fa2f6f836e04bc4f48d0489b688d8fc5e7e2767c047719c759760d756455e
Client private key:
e26e6d158c7f25f85dce7e8103843968335e60756ab2080b0aac0781dd33db56
Session key (used in AES):
9696ec6d87059017d12612b241b71f0163add5c36800b2ce183f5e053be95d68
ret_val=16
roza@roza-S301LA ~/Downloads/SGX/sgxsdk/SampleCode/RoZaEnclave/Client $ ./app 192.168.0.100 Quadripler 100
(App/App.cpp:343): A(Client public key):  f68c6e2a12d7c9057c7663c8871503555c28b6fb0ea9da4f17052322a3ec824
(App/App.cpp:347): B(Server public key): 179fa2f6f836e04bc4f48d0489b688d8fc5e7e2767c047719c759760d756455e
Client private key:
6bb289084a3461637eab3e3880a02de9e320f5aeeaea5c62e24d5ff9685f5dc7
Session key (used in AES):
b2ad6b25d456572d0af1cae688f899cfe8e42109b75560a44f3bf4ee76a56383
ret_val=400
```

Альтернативные решения

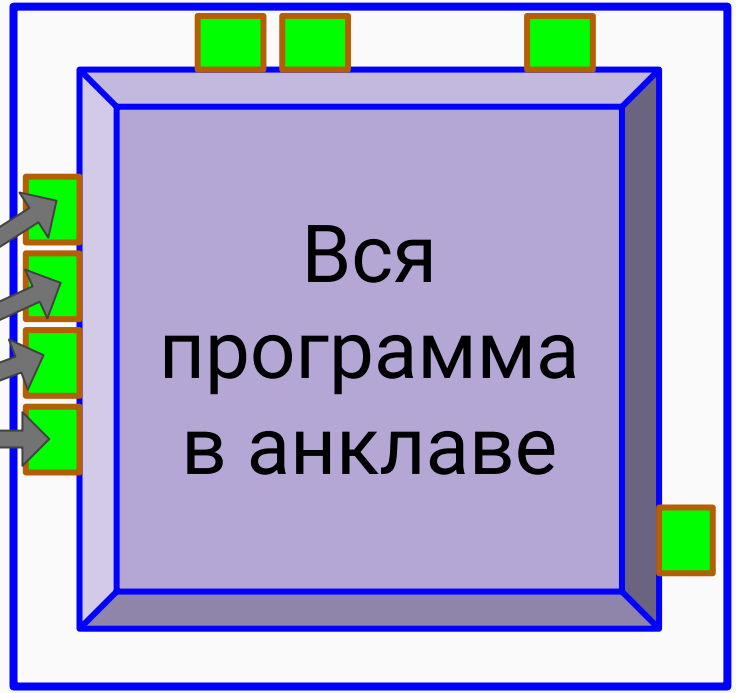
Режим шифрования AES	
Electronic code book (используется)	Cipher block chaining
Шифрование блоков кода независимо друг от друга	Сцепление блоков (отсутствие статистических особенностей)
Возможно распараллеливание вычислений	Невозможность распараллеливания шифрования

Поиск функции в анклаве при обращении	
elf_parser (используется)	Таблица соответствия имен и смещений
Использование проманглированных имен анклавных функций	Использование простых имен для анклавных функций
Уже реализованное решение	Для создания таблицы необходима доп. утилита

Альтернативные решения

Почему не сделать так?

Разработчику придется самому сделать обертку для каждого возможного системного вызова



Выводы

В предположении

1. Корректной работы технологии Intel SGX,
2. Криптографической стойкости используемых шифров,
3. Криптографической стойкости системы с открытым ключом,

представленная схема работает корректно.

Версия разработки может быть использована разработчиками ПО.