

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

Национальный исследовательский университет

«Высшая школа экономики»

Московский институт электроники и математики

Департамент прикладной математики

Образовательная программа «Компьютерная безопасность»

Индивидуальный проект

Выполнили:

Пермичев Никита,

студент группы СКБ-201

Оценка _____

Москва-2024

Оглавление

1. Общая идея проекта	3
2. Подготовка криптографической среды и интеграция ГОСТ	3
Установка и настройка OpenSSL 3.0:	3
Сборка и интеграция GOST-движка (gost-engine):	3
Конфигурационный файл OpenSSL для загрузки ГОСТ:	3
3. Реализация фрагмента Key Schedule TLS 1.3 с использованием ГОСТ- алгоритмов	4
Принцип Key Schedule в TLS 1.3:	4
Использование HKDF на базе ГОСТ-хэширования:	4
Упрощённый сценарий и фиктивные данные:	5
Результаты:	5
Приложение 1 (подготовка)	5
Приложение 2 (Реализация фрагмента)	7

1. Общая идея проекта

Главная цель:

- Выбрать фрагмент протокола **TLS 1.3** (Key Schedule, HKDF, Handshake-интеракции и т. д.), изучить его реализацию на основе русскоязычного источника («Ключи, шифры, сообщения: как работает TLS», версия от 17/12/2024), а затем **демонстрировать** собственный учебный код.

Дополнительно (по собственной инициативе):

- Показать, что в процедуре Key Schedule (которая по умолчанию использует SHA-256/384) можно вместо SHA применить **ГОСТ Р 34.11-2012** («Стрибог»).

В ходе проекта была выбрана стадия **Key Schedule** в TLS 1.3: это процесс вычисления секретов (Early Secret, Handshake Secret, Master Secret) с помощью HKDF.

- **Оригинальный TLS 1.3** базируется на SHA-256/384;
- Мы же иллюстрируем замену на ГОСТ-хеш, сохраняя основную логику.

2. Подготовка криптографической среды и интеграция ГОСТ

Установка и настройка OpenSSL 3.0:

Для использования TLS 1.3 и возможности интегрировать ГОСТ-примитивы была установлена библиотека **OpenSSL 3.0** (на macOS с помощью Homebrew, командой `brew install openssl@3`).

Сборка и интеграция GOST-движка (gost-engine):

Репозиторий gost-engine был клонирован и собран:

```
git clone https://github.com/gost-engine/engine.git
cd engine
mkdir build && cd build
cmake ..
make
make install
```

После этого в системе появился движок GOST (gost.dylib), дающий доступ к ГОСТ-хешу `md_gost12_256`, шифрованию ГОСТ 28147-89 и др.

Конфигурационный файл OpenSSL для загрузки ГОСТ:

Для автоматической загрузки движка GOST создан файл `openssl.cnf`, где прописаны соответствующие секции:

```
openssl_conf = openssl_init
```

```
[openssl_init]
```

```
engines = engine_section
```

```
[engine_section]
```

```
gost = gost_section
```

```
[gost_section]
```

```
engine_id = gost
```

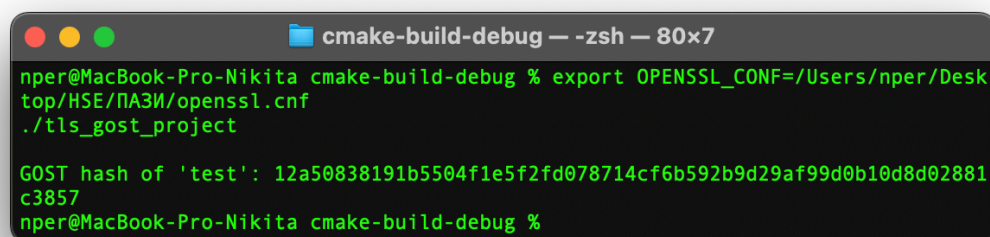
```
dynamic_path = /opt/homebrew/lib/engines-3/gost.dylib
```

```
default_algorithms = ALL
```

Затем перед запуском:

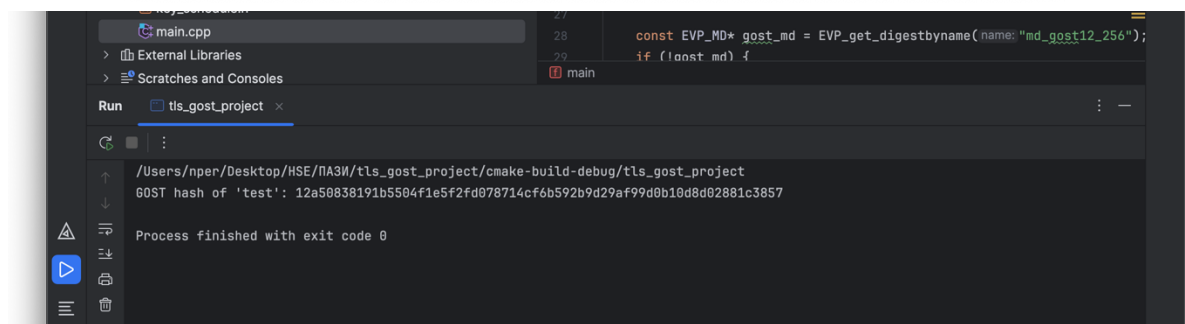
```
export OPENSSL_CONF=/путь/к/openssl.cnf
```

— и OpenSSL при инициализации подхватывает отечественные алгоритмы.



```
cmake-build-debug — zsh — 80x7
nper@MacBook-Pro-Nikita cmake-build-debug % export OPENSSL_CONF=/Users/nper/Desktop/HSE/ПАЗИ/openssl.cnf
./tls_gost_project

GOST hash of 'test': 12a50838191b5504f1e5f2fd078714cf6b592b9d29af99d0b10d8d02881c3857
nper@MacBook-Pro-Nikita cmake-build-debug %
```



3. Реализация фрагмента Key Schedule TLS 1.3 с использованием ГОСТ-алгоритмов

Принцип Key Schedule в TLS 1.3:

Key Schedule — это цепочка вычисления секретов, начиная с Early Secret (основанного на PSK или нулевом ключе), затем Handshake Secret (с учётом результата согласования ключей, например ECDH), и, наконец, Master Secret. Эти секреты впоследствии используются для генерирования ключей шифрования трафика.

Использование HKDF на базе ГОСТ-хэширования:

В учебном примере делаем **HKDF-Extract**, но меняем хеш с SHA на **md_gost12_256**.

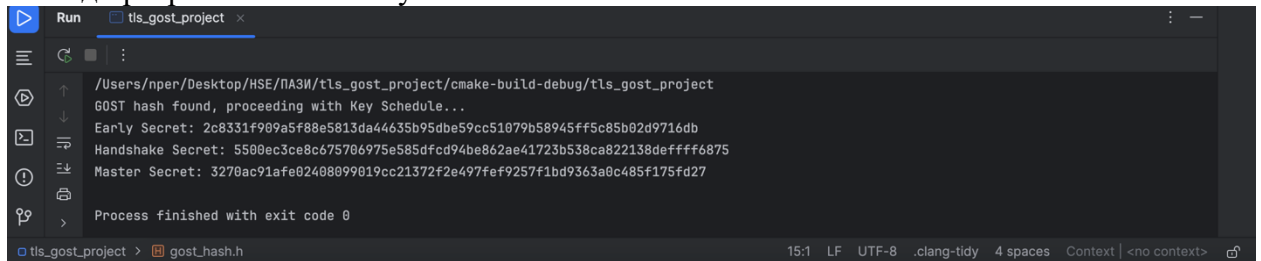
- Для корректного HMAC обычно нужны ipad/opad, но мы показали упрощённый вариант — главное, что вызов EVP-хеша заменён с EVP_sha256() на EVP_get_digestbyname("md_gost12_256").

Упрощённый сценарий и фиктивные данные:

Для демонстрации не требовался полноценный TLS Handshake. Были использованы фиксированные данные для PSK, "shared secret" и "context", чтобы сфокусироваться именно на криптологической части ключевого расписания. В результате мы получили Early Secret, Handshake Secret и Master Secret, рассчитанные на основе ГОСТ-хеша.

Результаты:

Вывод программы после запуска:



```
Run [tls_gost_project x]
/Users/nper/Desktop/HSE/ПА3И/tls_gost_project/cmake-build-debug/tls_gost_project
GOST hash found, proceeding with Key Schedule...
Early Secret: 2c8331f989a5f88e5813da44635b95dbe59cc51879b58945ff5c85b02d9716db
Handshake Secret: 5580ec3ce8c675706975e585dfcd94be862ae41723b538ca822138deffff6875
Master Secret: 3270ac91afe02408099019cc21372f2e497fef9257f1bd9363a0c485f175fd27
Process finished with exit code 0
```

Эти значения свидетельствуют о том, что Key Schedule, построенный на базе ГОСТ-хэширования, успешно функционирует.

Приложение 1 (подготовка)

1. Структура проекта

```
CMakeLists.txt
main.cpp
openssl.cnf
```

2. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(gost_integration LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Укажите путь к установленному OpenSSL (для macOS через Homebrew):
set(OPENSSL_ROOT_DIR "/opt/homebrew/opt/openssl@3")

include_directories("${OPENSSL_ROOT_DIR}/include")

add_executable(gost_integration main.cpp)

target_link_libraries(gost_integration
    "${OPENSSL_ROOT_DIR}/lib/libcrypto.dylib"
    "${OPENSSL_ROOT_DIR}/lib/libssl.dylib")
```

)

3. openssl.cnf

```
openssl_conf = openssl_init
```

```
[openssl_init]  
engines = engine_section
```

```
[engine_section]  
gost = gost_section
```

```
[gost_section]  
engine_id = gost  
dynamic_path = /opt/homebrew/lib/engines-3/gost.dylib  
default_algorithms = ALL
```

4. main.cpp

```
#include <iostream>  
#include <cstring>  
#include <openssl/evp.h>  
#include <openssl/engine.h>  
#include <openssl/conf.h>  
#include <openssl/crypto.h>
```

```
int main() {  
    OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CONFIG, NULL);  
  
    ENGINE_load_dynamic();  
    ENGINE *e = ENGINE_by_id("gost");  
    if (!e) {  
        std::cerr << "GOST engine not found!\n";  
        return 1;  
    }  
    if (!ENGINE_init(e)) {  
        std::cerr << "Failed to initialize GOST engine!\n";  
        ENGINE_free(e);  
        return 1;  
    }  
    ENGINE_set_default(e, ENGINE_METHOD_ALL);  
  
    const EVP_MD* gost_md = EVP_get_digestbyname("md_gost12_256");  
    if (!gost_md) {  
        std::cerr << "GOST hash not found!\n";  
        ENGINE_finish(e);  
        ENGINE_free(e);  
        return 1;  
    }  
  
    // Подготовим данные для хэширования  
    const char* message = "test";
```

```

EVP_MD_CTX* ctx = EVP_MD_CTX_new();
EVP_DigestInit_ex(ctx, gost_md, NULL);
EVP_DigestUpdate(ctx, message, strlen(message));

unsigned char out[32];
unsigned int outlen = 0;
EVP_DigestFinal_ex(ctx, out, &outlen);
EVP_MD_CTX_free(ctx);

std::cout << "GOST hash of 'test': ";
for (unsigned int i = 0; i < outlen; i++) {
    printf("%02x", out[i]);
}
std::cout << std::endl;

ENGINE_finish(e);
ENGINE_free(e);

return 0;
}

```

5. Инструкции по запуску:

- Установить нужную переменную окружения:
export OPENSSL_CONF=/путь/к/openssl.cnf
- Собрать проект:
mkdir build
cd build
cmake ..
cmake --build .
- Запустить:
./gost_integration

Приложение 2 (Реализация фрагмента)

Исходники фрагментарной реализации HKDF + Key Schedule с ГОСТ-хешем доступны в GitHub-репозитории:

https://github.com/Permichev/tls_gost_project

Там продемонстрировано:

- класс GostHash, обёртка над EVP_get_digestbyname("md_gost12_256");
- класс HKDF_Gost, где extract() + (при желании) expand();
- вызовы **EVP** на базе ГОСТ вместо SHA.
-

Таким образом, цель проекта — **изучить** фрагмент TLS 1.3 (Key Schedule) и **показать**, что в нём можно заменить стандартное SHA-хеширование на **ГОСТ «Стрибог»**.

Итоговый запуск подтверждает корректное формирование секретов Early/Handshake/Master.