ПРОИЗВОДСТВЕННАЯ ПРАКТИКА.

О КОМПАНИИ.

Mercado App Los-Lugares Limited — это английская компания, специализирующаяся на разработке инновационного маркетплейса для кодеров/программистов и создании высококачественного веб-приложения - цель которого заключается в предоставлении доступа к передовым технологиям и образованию будущего, основанному на искусственном интеллекте, чтобы подготовить будущее поколение к вызовам современного мира. Компания активно развивает и внедряет передовые технологии в производственные процессы, предлагая ученикам/студентам и молодым специалистам возможность получить ценный опыт и знания на практике через программы производственной практики. Mercado App Los-Lugares Limited стремится стать главным партнером в цифровой трансформации бизнеса и образования, предоставляя инновационные решения и поддержку на каждом этапе развития наших клиентов.

ПУТЬ ПОЛУЧЕНИЯ ПРАКТИЧЕСКОГО ОПЫТА.

Согласование и подготовка:

- **Определение целей:** Обсуждение и согласование целей и ожиданий от практики с преподавателем.
- **Подготовка рабочего места:** Подготовка необходимого программного обеспечения и доступа к ресурсам компании.

Разработка на C/C++:

- Участие в разработке backend-части приложений, включая оптимизацию и написание высокопроизводительного кода.
- Работа с базами данных и алгоритмами, реализация сложных вычислений и обработки данных.

Разработка на Python:

- Создание скриптов и автоматизация процессов.
- Работа с веб-фреймворками для backend-разработки (например, Django или Flask).
- Работа с анализом данных и машинным обучением (если применимо).

Разработка на JavaScript (frontend):

- Разработка и поддержка пользовательского интерфейса веб-приложений.
- Использование современных фреймворков и библиотек (например, React.js).
- Взаимодействие с RESTful API и интеграция различных компонентов frontend-стека.

СПЕЦИАЛИЗАЦИЯ ЯЗЫК C++.

Основная задача:

разработка криптографической библиотеки на языке C++. Библиотека должна поддерживать основные алгоритмы симметричного и асимметричного шифрования, хэширования данных, аутентификации сообщений и генерации криптографически безопасных случайных чисел. Вам необходимо обеспечить высокую производительность и безопасность реализации, используя современные подходы и стандарты криптографии.

ШАГИ/ТРЕБОВАНИЯ:

- **Согласование задачи/Ознакомление.**
- **Симметричное шифрование:** Реализация алгоритмов, таких как AES (Advanced Encryption Standard), для защиты данных.
- **Асимметричное шифрование:** Реализация алгоритма RSA для обеспечения безопасного обмена ключами и шифрования данных.
- **Хэширование данных:** Реализация алгоритма хэширования, SHA-256 для обеспечения целостности данных.
- **Аутентификация сообщений:** Использование алгоритма HMAC (Hash-based Message Authentication Code) для проверки подлинности и целостности сообщений.
- **Генерация случайных чисел:** Реализация криптографически безопасных генераторов случайных чисел, таких как HMAC-DRBG (Deterministic Random Bit Generator).
- **Тестирование/Оптимизация.**
- **Теоретический отчет.**

Реализация + Теория/Определения.

Симметричное шифрование — это метод шифрования, при котором один и тот же ключ используется как для шифрования, так и для расшифровки данных. Одним из наиболее распространенных алгоритмов симметричного шифрования является AES (Advanced Encryption Standard). AES использует блочное шифрование, где данные разбиваются на блоки фиксированного размера и каждый блок шифруется отдельно с использованием ключа.

AES Алгоритм/Определения:

1. **Блочное шифрование**: Данные шифруются блоками фиксированного размера (128 бит). Каждый блок проходит через несколько итераций (раундов), каждый из которых применяет различные преобразования к блоку.
2. **Размер ключа**: AES поддерживает ключи различной длины: 128 бит (16 байт), 192 бит (24 байта) и 256 бит (32 байта). Более длинные ключи обеспечивают более высокий уровень безопасности.
3. **Итеративность и раунды**: Алгоритм AES состоит из нескольких итераций (раундов), в каждом из которых выполняются следующие шаги:

   a. **SubBytes**: Замена каждого байта блока на соответствующий байт из S-блока (таблицы замен).
   b. **ShiftRows**: Циклический сдвиг строк блока: первая строка остается без изменений, вторая сдвигается на один байт влево, третья на два байта влево, четвертая на три байта влево.
   c. **MixColumns**: Каждый столбец блока трансформируется с использованием фиксированной матрицы умножения в поле Галуа.
   d. **AddRoundKey**: К блоку применяется операция XOR с раундовым ключом, полученным из основного ключа.
4. **Ключевое расширение**: Из основного ключа генерируются раундовые ключи для каждого раунда шифрования с помощью алгоритма расширения ключа (Key Expansion), который использует S-блок и операции циклического сдвига, XOR и другие трансформации.
5. **Финальный раунд**: После выполнения основных раундов выполняется финальный раунд без операции MixColumns, что завершает процесс шифрования.

Шифрование AES: При шифровании исходные данные разбиваются на блоки по 128 бит, каждый блок последовательно проходит через все раунды AES, начиная с начального добавления раундового ключа и заканчивая финальным добавлением раундового ключа.

Дешифрование AES: Для расшифровки данные проходят через аналогичный процесс, но в обратном порядке: сначала применяется финальный раунд, затем выполняются обратные операции ShiftRows, SubBytes и AddRoundKey. Операция MixColumns заменяется на обратное преобразование.

Для реализации AES на C++ можно использовать библиотеку OpenSSL, которая предоставляет готовые реализации криптографических алгоритмов. #include <openssl/aes.h> НО мы будем реализовывать самостоятельно!

Размер блока AES 16 байт и количество раундов для AES 256-bit имеет 14.

```
const int AES_BLOCK_SIZE = 16;
const int AES_ROUNDS = 14;
```

AES SBOX

```
const unsigned char AES_SBOX[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
```

```
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16};
```

AES ROUND KEYS.

```
std::vector<unsigned char> aesRoundKeys;
```

```
// AES Key Expansion (using Rijndael's key schedule)
void keyExpansion(const std::vector<unsigned char>& key) {
    aesRoundKeys = key; // Простое копирование ключа (это демонстрационный пример)
    // В реальности нужно реализовать алгоритм расширения ключа AES
}
```

```
void subBytes(unsigned char state[AES_BLOCK_SIZE]) {
    for (int i = 0; i < AES_BLOCK_SIZE; ++i) {
        state[i] = AES_SBOX[state[i]];
    }
}
void shiftRows(unsigned char state[AES_BLOCK_SIZE]);
void mixColumns(unsigned char state[AES_BLOCK_SIZE]);
```

```
void addRoundKey(unsigned char state[AES_BLOCK_SIZE], int round) {
    // XOR state с ключом раунда
    for (int i = 0; i < AES_BLOCK_SIZE; ++i) {
        state[i] ^= aesRoundKeys[round * AES_BLOCK_SIZE + i];
    }
}
// AES Encryption function
void encryptAES(const unsigned char plaintext[AES_BLOCK_SIZE],
```

```
unsigned char ciphertext[AES_BLOCK_SIZE]) {
    unsigned char state[AES_BLOCK_SIZE];
    for (int i = 0; i < AES_BLOCK_SIZE; ++i) state[i] = plaintext[i];
    addRoundKey(state, 0);
    for (int round = 1; round < AES_ROUNDS; ++round) {
        subBytes(state);
        shiftRows(state);
        mixColumns(state);
        addRoundKey(state, round);
    }
    subBytes(state);
    shiftRows(state);
    addRoundKey(state, AES_ROUNDS);
    for (int i = 0; i < AES_BLOCK_SIZE; ++i) ciphertext[i] = state[i];
}
```

```
int main() {
    unsigned char plaintext[AES_BLOCK_SIZE] = {'H', 'e', 'l', 'l', 'o', ',', ' ', 'w',
'o', 'r', 'l', 'd', '!', '!', '!', '!'};
    unsigned char ciphertext[AES_BLOCK_SIZE];
    // Вставить AES ключ (256 бит)
    std::vector<unsigned char> aesKey = {'k', 'e', 'y', '1', '2', '3', '4', '5', '6',
```

```
'7', '8', '9', '0', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r'};
   keyExpansion(aesKey);
   // Зашифровать
   encryptAES(plaintext, ciphertext);
   // Вывести зашифрованный текст
   for (int i = 0; i < AES_BLOCK_SIZE; ++i) std::cout << ciphertext[i];
   return 0;}
```

Асимметричное шифрование (или открытое шифрование) использует пару ключей: открытый и закрытый. Открытый ключ используется для шифрования данных, а закрытый ключ — для их расшифровки. Один из наиболее известных алгоритмов асимметричного шифрования — RSA (Rivest-Shamir-Adleman), который основан на сложности разложения больших простых чисел на множители.

Основные шаги алгоритма RSA:

1. **Генерация ключей**:
   - Владелец генерирует пару ключей: открытый ключ (N, e) и закрытый ключ (N, d), где N - произведение двух больших простых чисел, e - открытая экспонента (часть открытого ключа), d - закрытая экспонента (часть закрытого ключа). Операция **mod** в C++ эквивалентна операции %.
2. **Шифрование данных**:
   - Для зашифрования данных используется открытый ключ (N, e); e - public key. Данные преобразуются в числовое представление m, которое затем возводится в степень e по модулю N, что дает зашифрованное сообщение c: $c = m^e \ mod \ N$
3. **Расшифровка данных**:
   - Зашифрованное сообщение c расшифровывается с использованием закрытого ключа (N, d); d - private key. Зашифрованное сообщение c возводится в степень d по модулю N, в результате чего восстанавливается исходное сообщение m: $m = c^d \ mod \ N$

RSA план действий:

- воспользоваться готовыми математическими функциями.

```
// Функция для проверки простоты числа
bool isPrime(int n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) return false;
    }
    return true;
}
// Функция для генерации случайного простого числа
```

```
int generatePrime(int min, int max) {
    int num;
    do {
        num = rand() % (max - min + 1) + min;
    } while (!isPrime(num));
    return num;
}
// Функция для нахождения наибольшего общего делителя
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
// Функция для вычисления обратного элемента по модулю
int modInverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) return x;
    }
    return 1;
}
```

- написать функцию генерацию ключей;
  - ➢ генерируем два случайных простых числа p и q;
  - ➢ находим их произведение - это будет n;
  - ➢ вычисляем ф-ю эйлера через phi; $phi = (p - 1) * (q - 1)$
  - ➢ выбираем d, e;

```
publicKey = generatePrime(2, phi - 1);
privateKey = modInverse(publicKey, phi);
```

```
void generateRSAKeys(int& publicKey, int& privateKey, int& n) {
    srand(time(NULL));
...
}
```

- написать функции шифрования/дешифрования;

```
int encryptRSA(int message, int publicKey, int n);
int decryptRSA(int encryptedMessage, int privateKey, int n);
```

Хэширование данных — это процесс преобразования входных данных произвольной длины в фиксированный хэш-значение фиксированной длины. Одним из популярных алгоритмов хэширования является SHA-256 (Secure Hash Algorithm 256-bit), который создает хэш-значение длиной 256 бит (32 байта). Хэширование используется для проверки целостности данных и создания уникального идентификатора для набора данных.

Основные характеристики SHA-256:

1. **Длина хэш-значения**: SHA-256 создает хэш-значение длиной 256 бит (32 байта).
2. **Безопасность**: SHA-256 обеспечивает высокий уровень безопасности благодаря своей структуре и сложности алгоритма. Он устойчив к коллизиям (ситуации, когда два разных набора данных дают одинаковое хэш-значение), что делает его подходящим для проверки целостности данных и создания цифровых подписей.
3. **Использование**: Хэширование данных с помощью SHA-256 применяется для:
    - **Проверки целостности данных**: Хэш-значение может служить проверкой, что данные не были изменены или повреждены.
    - **Цифровых подписей**: Хэш-значение используется в процессе создания и проверки цифровых подписей для подтверждения авторства и целостности данных.
    - **Хранения паролей**: Хэширование SHA-256 часто используется для хранения паролей в хеш-таблицах, чтобы предотвратить возможность восстановления исходного пароля из хранимого значения.

ПРОЦЕСС ХЭШИРОВАНИЯ:

1. **Инициализация**: Начальное значение (initial hash value) задается заранее определенным образом для каждого из 8 раундов.
2. **Предварительная обработка данных**: Входные данные дополняются до нужного размера и делятся на блоки данных фиксированной длины (512 бит).
3. **Циклическая обработка блоков**: Каждый блок данных проходит через серию итераций и операций, включая замены битовых значений, циклические сдвиги и комбинирование данных.
4. **Формирование хэш-значения**: После обработки всех блоков данных получается итоговое 256-битное хэш-значение.

Аутентификация сообщений обеспечивает подтверждение подлинности и целостности переданных данных. Один из методов аутентификации сообщений — HMAC (Hash-based Message Authentication Code), который использует хэширование с ключом для генерации аутентификационного кода. HMAC обеспечивает защиту от подделки данных и изменений в передаваемых сообщениях.

Криптографически безопасные случайные числа (Cryptographically Secure Random Numbers) необходимы для генерации ключей шифрования, сеансовых ключей и других криптографических параметров. Генераторы случайных чисел должны обеспечивать предсказуемость и невозможность восстановления последовательности случайных чисел из полученных значений. Один из стандартов для генерации криптографически безопасных случайных чисел — HMAC-DRBG (Deterministic Random Bit Generator).

СПЕЦИАЛИЗАЦИЯ ЯЗЫК PYTHON.

Основная задача:

разработка системы управления задачами с использованием языка Python. Система должна позволять пользователям создавать, управлять и отслеживать задачи, а также обеспечивать возможность совместной работы над задачами в рамках команды или проекта.

ШАГИ/ТРЕБОВАНИЯ:

- **Согласование задачи/Ознакомление.**
- **Интерфейс командной строки (CLI):** Разработка интерфейса командной строки для добавления, удаления, обновления и просмотра задач.
- **Веб-интерфейс:** Реализация простого веб-интерфейса с использованием фреймворка Flask или Django для удобного взаимодействия с системой через браузер.
- **Управление задачами:** Возможность создания задач с указанием заголовка, описания, статуса (например, "в работе", "завершено"), приоритета и сроков выполнения.
- **Аутентификация и авторизация:** Реализация системы аутентификации пользователей и управление правами доступа для различных ролей (администратор, пользователь).
- **База данных (SQL):** Использование простой локальной базы данных (например, SQLite) для хранения информации о задачах и пользовательских данных.
- **Тестирование/Оптимизация.**
- **Теоретический отчет.**

СПЕЦИАЛИЗАЦИЯ FRONTEND.