НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота 1

Вибір та реалізація базових фреймворків та бібліотек

Підгрупа 2А

Виконали:

Галіца О.О.

Паршин О.Ю.

Литвиненко Ю.С.

ФІ-22мн

Перевірила:

Байденко П.В.

Київ 2022

1 Теоретичні відомості про бібліотеки

1.1 PyCrypto

PyCryptodome - це безкоштовна бібліотека для мови Python із відкритим кодом. Вона є форком застарілої бібліотеки PyCrypto, і включає в себе низькорівневі криптографічні примітиви. Також автори стверджують, що PyCryptodome не є обгорткою для бібліотек мовою C як, наприклад, OpenSSL, про яку також буде згадано згодом. Наскільки це можливо, усі алгоритми в PyCryptodome розроблені саме чистою мовою Python. Виключення становлять випадки, коли швидкодія є критичною для криптографічного примітиву (наприклад, для блокових шифрів) - тоді вже код являє собою розширення мовою C.

Примітив або операція	Алгоритми або реалізації
Симетричні шифри	AES, Single and Triple DES, CAST-128, RC2
Режими блочного шифрування	ECB, CBC, CFB, OFB, CTR, OpenPGP (варіант CFB)
Аутентифіковані схеми шифрування	CCM (тільки AES), EAX, GCM (тільки AES), SIV (тільки AES), OCB (тільки AES), ChaCha20-Poly1305
Потокові шифри	Salsa20, ChaCha20, RC4
Геш-функції	SHA-1, SHA-2 (224, 256, 384, 512, 512/224, 512/256), SHA-3 (224, 256, 384, 512), SHAKE (128/256), TupleHash (128/256), KangarooTwelve (XOF), Keccak, BLAKE2b, BLAKE2s, RIPE-MD160, MD5
Коди автентифікації повідомлень	HMAC, CMAC, KMAC128, KMAC256, Poly1305
Криптографія з відкритим ключем	RSA, ECC (NIST P-curves; Ed25519, Ed448), ElGamal
Асиметричні шифри	PKCS (RSA), RSAES-PKCS1-v1, RSAES-OAEP
Асиметричний цифровий підпис	RSASSA-PKCS1-v1, RSASSA-PSS; (EC)DSA - Nonce-based (FIPS 186-3), Deterministic (RFC6979), EdDSA
Схеми падінгу	PKCS#7, ISO-7816, X.923

Серед очевидних переваг даної бібліотеки є простота її встановлення - але це характерно для більшості бібліотек мовою Python, на відміну від деяких бібліотек для C або C++. Операції та примітиви рознесені по відповідних блоках Cipher, Signature, Hash, PublicKey, Protocol, також наявні додаткові допоміжні блоки IO, Random, Util, в яких містяться, наприклад, операції по перетворенню ключів та методи для генерування випадкових бітів, що також є приємним доповненням до бібліотеки.

Порівняно з іншими бібліотеками, PyCryptodome не є обширною, автори зробили акцент на геш-фукнціях, але основні криптографічні примітиви реалізовані і, вкупі з простотою використання, ця бібліотека є основною для мови Python, коли йдеться про криптографічні алгоритми.

1.2 OpenSSL

OpenSSL – криптографічна бібліотека з відкритим вихідним кодом, написана мовою C++ (хоча для компіляції використовується Perl), перший реліз відбувся ще в далекому 1998 році. Найбільшого розсповсюдження зазнала через вміст open-source реалізації SSL та TLS протоколів, через що включена по замовчуванню у більшість дистрибутивів Linux (Трохи статистики – 66% усіх веб-серверів використовують OpenSSL). Хоча ядро написано мовою C/C++, певні високорівневі обгортки дозволяють використовувати OpenSSL в широкому спектрі мов програмування.

Примітив або операція	Алгоритми або реалізації
Блокові шифри	AES, Blowfish, Camellia, SEED, CAST-128, DES, IDEA, RC2, RC5, 3DES, GOST 28147-89, SM4
Потокові шифри	ChaCha20, RC4
Геш-функції	MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, MDC-2, GOST R 34.11-94, BLAKE2, Whirlpool, SM3
Криптографія з відкритим ключем	RSA, DSA, GOST R 34.10-2001, SM2
Генератори псевдовипадкових чисел	ANSI X9.31, SP800-90A (AES-256 CTR)
Криптографія на еліптичних кривих	X25519, ED25519, X448, ED448
Протоколи обміну ключами	Diffie-Hellman, elliptic curve Diffie-Hellman

Однією з переваг даної бібліотеки є EVP-інтерфейс: він дозволяє використовувати одні і ті ж функції криптографічних перетворень (шифрування, розшифрування, підпис тощо) для різних криптографічних систем. Також при використанні цього інтерфейсу буде автоматично увімкнено апаратне прискорення (наприклад, AES-NI – розширення команд x86, тобто, грубо кажучи, окремий блок в процесорі для прискорень обчислення AES).

Окрім цього в бібліотеці реалізовано тип BIGNUM – тобто є можливість працювати з усіма необхідними операціями довгої (додавання, множення, ділення, взяття в степінь, знаходження НСД) та модулярної (ті ж операції, але по модулю) арифметик. Це було написано для того, щоб реалізувати алгоритми асиметричної криптографії, проте користувач їх теж можеш використовувати для написання власних програмних рішень.

Останніми роками автори в основному працювали, створюючи підмодуль – велика частина цього модулю перетинається з уже написаним кодом, проте добавлено ряд оптимізацій та заходів безпеки. Все це для того, щоб відповідати вимогам FIPS 140-2 (тобто реалізовує усі затверджені NIST-ом стандарти з необхідним рівнем безпеки), що дозволить використовувати цю бібліотеку в урядових установах Канади та США. З другої половини 2021 року ведеться робота, спрямована вже на задоволення вимог FIPS 140-3.

1.3 Crypto++

Crypto++ (інші назви: CryptoPP, libcrypto++ і libcryptopp) — це безкоштовна бібліотека C++ із відкритим кодом. Ця бібліотека криптографічних алгоритмів і схем, написана Вей Даєм і вперше випущена у 1995 році. Crypto++ широко використовується в наукових колах, студентських проектах, проектах з відкритим кодом і некомерційних проектах, а також у бізнесі. Бібліотека повністю підтримує 32- і 64-розрядні архітектури для багатьох основних операційних систем і платформ, наприклад, Android, Apple, Linux, Windows і багато інших, менш відомих. Проект також підтримує різноманітні компілятори та IDE. Повний перелік можна переглянути на сайті. Вихідний код та заплановані зміни можна переглянути у репозиторії GitHub.

Crypto++ зазвичай забезпечує повну криптографічну реалізацію та часто включає менш популярні та рідше використовувані схеми. Наприклад, Camellia — це блоковий шифр, приблизно еквівалентний AES, а Whirlpool — хеш-функція, приблизно еквівалентна SHA. Обидва алгоритми включені в бібліотеку. Наведемо список алгоритмів, представлених у бібліотеці.

Примітив або операція	Алгоритми або реалізації
Аутентифіковані схеми шифрування	GCM, CCM, EAX, ChaCha20Poly1305, XChaCha20Poly1305
Швидкі потокові шифри	ChaCha (8/12/20), ChaCha (IETF) HC (128/256), Panama, Rabbit (128/256), Sosemanuk, Salsa20 (8/12/20), XChaCha (8/12/20), XSalsa20
AES та AES кандидати	AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256
Інші блокові шифри	ARIA, Blowfish, Camellia, CHAM, HIGHT, IDEA, Kalyna (128/256/512), LEA, SEED, RC5, SHACAL-2, SIMECK, SIMON (64/128), Skipjack, SPECK (64/128), Simeck, SM4,Threefish (256/512/1024), Triple-DES (DES-EDE2 and DES-EDE3), TEA, XTEA
Режими блочного шифрування	ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR), XTS
Коди автентифікації повідомлень	BLAKE2b, BLAKE2s, CMAC, CBC-MAC, DMAC, GMAC (GCM), HMAC, Poly1305, SipHash, Two-Track-MAC, VMAC
Геш-функції	BLAKE2b, BLAKE2s, Keccack (F1600), SHA-1, SHA-2, SHA-3, SHAKE (128/256), SipHash, LSH (128/256), Tiger, RIPEMD (128/160/256/320), SM3, WHIRLPOOL
Генератори псевдовипадкових чисел	LCG, KDF2, Blum Blum Shub, ANSI X9.17, Mersenne Twister, RandomPool, VIA Padlock, DARN, RDRAND and RDSEED
Криптографія на еліптичних кривих	ECDSA, Determinsitic ECDSA (RFC 6979), ed25519, ECGDSA, ECNR, ECIES, x25519, ECDH, ECMQV

Примітив або операція	Алгоритми або реалізації
Криптографія з відкритим ключем	RSA, DSA, Determinsitic DSA (RFC 6979), ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), EC-based German Digital Signature (ECGDSA), LUC, LUCELG, DLIES (variants of DHAES), ESIGN
Схеми падінгу	PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5
Історичні алгоритми (небезпечні або застарілі)	MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square
Протоколи обміну ключами	Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), Hashed MQV (HMQV), Fully Hashed MQV (FHMQV), LUCDIF, XTR-DH

Також бібліотека дозволяє працювати у скінченних полях $GF(p)$ та $GF(2^n)$, виконувати операції з поліномами, генерувати прості числа та перевіряти простоту числа. Більш того передбачено деякі корисні некриптографічні алгоритми, наприклад, DEFLATE (алгоритм стиснення даних), кодування у шістнадцятковій та інших системах числення та алгоритми підрахунку контрольних сум. Алгоритми AES, CRC, GCM і SHA використовують апаратне прискорення, якщо це можливо.

Crypto++ намагається протистояти атакам за побічними каналами, використовуючи різні засоби протидії. Так, наприклад, за можливості, першою лінією оборони є апаратні інструкції. Також з метою мінімізації витоку інформації використовуються алгоритми сталого часу (constant-time algorithms), алгоритми, які використовують кеш процесора (cache-aware algorithms) та шаблони доступу (access patterns - способи, якими система зчитує та записує інформацію в постійну пам'ять). Розробники відмічають, що засоби протидії можуть бути неповними і якщо ви підозрюєте витік інформації, потрібно повідомити їх про це. Взагалі розробниками вітаються внески у покращення бібліотеки, наприклад, пошук та виправлення помилок чи написання тест кейсів.

2 Порівняння

Тестування проводилось в трьох режимах. Генерувався потік даних у 8 гігабайт, але розподілялися по блокам по-різному.

2.1 Найменший блок

В цьому режимі ми генеруємо багато блоків по 128 бітів (тобто 16 байтів), і кожному з цих блоків генеруємо свій ключ і свій вектор ініціалізації. В цьому режимі багато часу йде на ініціалізацію контекстів і розгортання самого шифру. Вимір відбувається в секундах.

	AES-256	SHA3-256	DSA 1024
PyCrypto	551.92	180.1	-
OpenSSL	55.76	35.21	270.88
Crypto++	242.64	126.44	-

2.2 Середній блок

В цьому режимі блоки відкритого тексту вже трохи більшої довжини, а саме блоки відкритого тексту вже йдуть по 2048 байтів. Вимір відбувається в мілісекундах.

	AES-256	SHA3-256	DSA 1024
PyCrypto	5997.5	3862.9	155213.5
OpenSSL	1585.5	1020.8	2880.5
Crypto++	3539.2	3454.1	46167.9

2.3 Найбільший блок

В цьому режимі весь відкритий текст йде одним блоком розміром у 8 гігабайт.

	AES-256	SHA3-256	DSA 1024
PyCrypto	1904.03	2383.09	2687.42
OpenSSL	1191	730.14	763.03
Crypto++	1189.11	2539.8	837.06

3 Реалізація

3.1 PyCryptoDome

```
def generate_data(key_size, salt_size, plain_text_block_size, blocks_count):
        keys = [get_random_bytes(key_size) for i in range(blocks_count)]
        salts = [get_random_bytes(salt_size) for i in range(blocks_count)]
        texts = [get_random_bytes(plain_text_block_size) for i in range(blocks_count)]

        return {'keys': keys, 'salts': salts, 'texts': texts}

def test_AES(data, blocks_count):
        for index in range(blocks_count):
                cipher = AES.new(data['keys'][index], AES.MODE_CBC,
            iv=data['salts'][index])
                cipher.encrypt(data['texts'][index])

def test_SHA3(data, blocks_count):
    for index in range(blocks_count):
        hash_obj = SHA3_256.new()
        hash_obj.update(data['texts'][index])

def test_DSA(data, blocks_count):
    signer = DSS.new(data['dsa_key'], 'fips-186-3')

    for index in range(blocks_count):
        signer.sign(SHA3_256.new(data['texts'][index]))
```

3.2 OpenSSL

```
std::pair<unsigned char*, int> opensslAESEncrypt(const unsigned char* plainText,
    const unsigned char* key, const unsigned char* salt, bool needReturn)
{
    auto plainTextLength = strlen((char*)plainText);

    EVP_CIPHER_CTX* cipherContext = EVP_CIPHER_CTX_new();

    auto cipherText = new unsigned char[plainTextLength * 2];
    int length;

    EVP_EncryptInit_ex(cipherContext, EVP_aes_256_cbc(), NULL, key, salt);

    EVP_EncryptUpdate(cipherContext, cipherText, &length, plainText, plainTextLength);
    int cipherTextLength = length;

    EVP_EncryptFinal_ex(cipherContext, cipherText + length, &length);
    cipherTextLength += length;

    EVP_CIPHER_CTX_free(cipherContext);

    if (needReturn == false)
    {
        delete[] cipherText;
        cipherText = nullptr;
    }

    return std::make_pair(cipherText, cipherTextLength);
}

unsigned char* opensslSHA3(const unsigned char* plainText, bool needReturn)
{
    auto plainTextLength = strlen((char*)plainText);

    EVP_MD_CTX* digestContext = EVP_MD_CTX_new();

    EVP_DigestInit_ex(digestContext, EVP_sha256(), NULL);
    EVP_DigestUpdate(digestContext, plainText, plainTextLength);

    unsigned int digestLength = SHA256_DIGEST_LENGTH;
```

```
    auto digest = static_cast<unsigned char*>(OPENSSL_malloc(digestLength));

    EVP_DigestFinal_ex(digestContext, digest, &digestLength);

    EVP_MD_CTX_free(digestContext);

    if (needReturn == false)
    {
        OPENSSL_free(digest);
    }

    return digest;
}

EVP_PKEY* opensslDSAGenerateKey(unsigned int keyLength)
{
    EVP_PKEY_CTX* paramsContext = EVP_PKEY_CTX_new_id(EVP_PKEY_DSA, NULL);
    EVP_PKEY_paramgen_init(paramsContext);

    EVP_PKEY_CTX_set_dsa_paramgen_bits(paramsContext, keyLength);

    EVP_PKEY* params = NULL;
    EVP_PKEY_paramgen(paramsContext, &params);

    EVP_PKEY_CTX* keyContext = EVP_PKEY_CTX_new(params, NULL);
    EVP_PKEY_keygen_init(keyContext);


    EVP_PKEY* key = NULL;
    EVP_PKEY_keygen(keyContext, &key);

    //fprintf(stdout, "Generating public/private key pair:\n");
    //EVP_PKEY_print_private_fp(stdout, key, 4, NULL);
    //fprintf(stdout, "\n");

    EVP_PKEY_CTX_free(paramsContext);
    EVP_PKEY_free(params);
    EVP_PKEY_CTX_free(keyContext);

    return key;
```

```
}

std::pair<unsigned char*, int> opensslDSASign(
    unsigned char* plainText, EVP_PKEY* key, bool needReturn
)
{
    auto plainTextLength = strlen((char*)plainText);

    EVP_MD_CTX* digestContext = EVP_MD_CTX_create();
    const EVP_MD* digestFunction = EVP_sha256();

    EVP_DigestInit_ex(digestContext, digestFunction, NULL);

    EVP_DigestSignInit(digestContext, NULL, digestFunction, NULL, key);

    EVP_DigestSignUpdate(digestContext, plainText, plainTextLength);

    size_t signLength = 0;
    EVP_DigestSignFinal(digestContext, NULL, &signLength);
    unsigned char* sign = static_cast<unsigned char*>(OPENSSL_malloc(signLength));

    EVP_MD_CTX_free(digestContext);

    if (needReturn == false)
    {
        OPENSSL_free(sign);
    }

    return std::make_pair(nullptr, signLength);
}
```

3.3 Crypto++

```
void cryptoppAESEncrypt(
    const unsigned char* plainText, const unsigned char* plainKey,
    const unsigned char* salt
)
{
    HexEncoder encoder(new FileSink(std::cout));
    SecByteBlock key(AES::MAX_KEYLENGTH);
    SecByteBlock iv(AES::BLOCKSIZE);

    std::string cipher;

    CBC_Mode< AES >::Encryption e;
    e.SetKeyWithIV(plainKey, key.size(), salt);

    StringSource s(static_cast<std::string>((char*)plainText), true,
        new StreamTransformationFilter(e,
            new StringSink(cipher)
        ) // StreamTransformationFilter
    ); // StringSource
}


void cryptoppSHA3(const unsigned char* plainText)
{
    HexEncoder encoder(new FileSink(std::cout));

    std::string message = static_cast<std::string>((char*)plainText);
    std::string digest;

    SHA3_256 hash;
    hash.Update((const byte*)message.data(), message.size());
    digest.resize(hash.DigestSize());
    hash.Final((byte*)&digest[0]);
}

std::pair<DSA::PublicKey, DSA::PrivateKey> cryptoppDSAGenerateKey(
    unsigned int keyLength, AutoSeededRandomPool &prng
)
{
```

```
    HexEncoder encoder(new FileSink(std::cout));
    DSA::PrivateKey privateKey;
    privateKey.GenerateRandomWithKeySize(prng, keyLength);

    DSA::PublicKey publicKey;
    publicKey.AssignFrom(privateKey);

    return std::make_pair(publicKey, privateKey);
}

void cryptoppDSASign(
    unsigned char* plainText, std::pair<DSA::PublicKey, DSA::PrivateKey> key,
    AutoSeededRandomPool& prng
)
{
    std::string message = static_cast<std::string>((char*)plainText);
    std::string signature;
    std::string output;

    DSA::Signer signer(key.second);
    StringSource(message, true,
        new SignerFilter(prng, signer, new StringSink(signature))
    );
}
```