

Разработка системы электронной подписи на Python с использованием библиотеки Cryptography

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данном исследовании рассматривается разработка программы для подписания документов электронной подписью. Программа реализована на языке программирования Python с использованием библиотеки Cryptography. Результатом исследования является готовая система для электронного подписания документов, а также описание её работы.

Ключевые слова: python, cryptography, кибербезопасность

Development of an electronic signature system in Python using the Cryptography library

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

This study examines the development of a program for signing documents with an electronic signature. The program is implemented in the Python programming language using the Cryptography library. The result of the research is a ready-made system for electronic document signing, as well as a description of its operation.

Keywords: python, cryptography, cybersecurity

1 Введение

1.1 Актуальность

Актуальность разработки системы электронной подписи на Python обусловлена растущей потребностью в обеспечении безопасности и подлинности документов в цифровом мире. В условиях увеличения объема электронных коммуникаций и документооборота, защита данных от несанкционированного доступа и подделки становится критически важной. Электронная подпись позволяет гарантировать целостность и аутентичность документов, что особенно важно в таких областях, как юриспруденция, финансы, здравоохранение и государственное управление. Использование современных криптографических алгоритмов и библиотек, таких как “cryptography”, обеспечивает высокий уровень безопасности и надежности, что делает разработку таких систем актуальной и востребованной задачей.

Кроме того, возможность выявления подмены подписей и проверки их подлинности позволяет предотвратить мошеннические действия и повысить доверие к электронным документам.

1.2 Обзор исследований

В.Х.Н. Аль-Рубайе в своём труде рассмотрел некоторые из самых популярных методов шифрования и атак, а также программную реализацию простых алгоритмов шифрования на примере алгоритма шифра ограждения рельсов [1]. Д.Н. Кустов и А.А. Конин описали алгоритм шифрования и процесс его реализации на языке программирования Python [2]. М.В. Бабичева и Б.С. Савченко в совместной работе изучили уязвимости блочных шифров, которые широко применяются в настоящее время для закрытия информации в инфокоммуникационных системах [3]. Е. Сейткулов, Р. Оспанов, Н. Ташатов и другие посветили исследование программной реализации методов анализа и синтеза криптографических S-блоков [4]. Ю.А.Костиков и А.М. Романенков рассмотрели методы организации безопасных механизмов авторизации и аутентификации [5].

1.3 Цель исследования

Целью исследования является создание и описание системы электронной подписи на языке программирования Python с использованием библиотеки cryptography.

2 Материалы и методы

Для реализации системы используется язык программирования Python и библиотека cryptography. В качестве IDE выступает PyCharm.

3 Результаты и обсуждения

Приложение состоит из четырёх скриптов: генерация ключей и сертификатов, подписание документа, подмена подписи и проверка на подлинность. Перед непосредственной разработкой устанавливается библиотека cryptography через терминал IDLE.

После установки инструментов происходит написание первого скрипта, отвечающего за генерацию ключей и сертификатов (рис. 1).

В данном шаге происходит генерация ключей и сертификатов для двух пользователей. Второй пользователь необходим для проверки работоспособности системы на опознавание подмены подписи. Этот процесс включает несколько ключевых этапов, которые обеспечивают безопасность и аутентификацию в системе электронной подписи.

```

def generate_keys_and_certificate(user_name):
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )

    public_key = private_key.public_key()

    subject = issuer = x509.Name([
        x509.NameAttribute(NameOID.COUNTRY_NAME, value=u"US"),
        x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, value=u"California"),
        x509.NameAttribute(NameOID.LOCALITY_NAME, value=u"San Francisco"),
        x509.NameAttribute(NameOID.ORGANIZATION_NAME, value=u"My Company"),
        x509.NameAttribute(NameOID.COMMON_NAME, value=u"mysite.com"),
    ])

    builder = CertificateBuilder(
        subject_name=subject,
        issuer_name=issuer,
        public_key=public_key,
        serial_number=x509.random_serial_number(),
        not_valid_before=datetime.utcnow(),
        not_valid_after=datetime.utcnow() + timedelta(days=10),
    )

    cert = builder.sign(
        private_key=private_key,
        algorithm=hashes.SHA256(),
    )

    with open(f"{user_name}_private_key.pem", "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))

    with open(f"{user_name}_public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    with open(f"{user_name}_public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))

    with open(f"{user_name}_certificate.pem", "wb") as f:
        f.write(cert.public_bytes(serialization.Encoding.PEM))

    generate_keys_and_certificate("user1")
    generate_keys_and_certificate("user2")

```

Рисунок 1. Скрипт генерации ключей и сертификатов

В начале скрипта происходит генерация приватного ключа. Приватный ключ — это секретный ключ, который используется для создания цифровой подписи. Он генерируется с использованием алгоритма RSA (Rivest-Shamir-Adleman), который является одним из самых популярных алгоритмов асимметричного шифрования. Описываемый ключ сохраняется в зашифрованном виде, чтобы предотвратить его утечку или несанкционированное использование. Для реализации вышесказанного используется функция “rsa.generate_private_key” для генерации приватного ключа с использованием алгоритма RSA. Параметры “public_exponent=65537” и “key_size=2048” задают общепринятые значения для RSA-ключей.

После генерации приватного ключа создается публичный ключ. Публичный ключ — это открытый ключ, который используется для проверки цифровой подписи. Он генерируется на основе приватного ключа и может быть свободно распространяем, так как он не содержит секретной информации. Данный ключ также сохраняется в формате PEM (Privacy-Enhanced Mail), который является стандартом для хранения криптографических ключей и сертификатов. Генерация ключа на основе приватного ключа осуществляется с помощью метода “public_key()”.

Следующим шагом является генерация сертификата. Сертификат — это электронный документ, который подтверждает подлинность публичного ключа. Он содержит информацию о владельце ключа, такую как имя, организация и страна. Сертификат подписывается приватным ключом, что гарантирует его целостность и подлинность. Сертификат также сохраняется в формате PEM. Для построения сертификатов создается объект “CertificateBuilder”. В этот объект добавляется информация о владельце ключа, такая как страна, регион, город, организация и общее имя. Сертификат подписывается приватным ключом с использованием алгоритма SHA-256.

В результате выполнения описываемого скрипта на выходе создаются три файла в формате “.pem” - приватный и публичный ключ, а также сертификат. Эти файлы могут быть использованы для подписания и проверки документов.

В следующем скрипте “sign_document.py” происходит подписание документа пользователем 1 (рис. 2). Этот процесс включает несколько ключевых этапов, которые обеспечивают создание цифровой подписи для документа.

```
def sign_document(document_path, private_key_path, signature_path):  
  
    with open(document_path, "rb") as f:  
        document_data = f.read()  
  
    with open(private_key_path, "rb") as f:  
        private_key = serialization.load_pem_private_key(  
            f.read(),  
            password=None,  
        )  
  
    signature = private_key.sign(  
        document_data,  
        padding.PKCS1v15(),  
        hashes.SHA256()  
    )  
  
    with open(signature_path, "wb") as f:  
        f.write(signature)  
  
sign_document( document_path: "document.txt", private_key_path: "user1_private_key.pem", signature_path: "user1_signature.bin")
```

Рисунок 2. Скрипт подписание файла

Вначале скрипта происходит чтение документа, который необходимо подписать. В качестве документа выбирается текстовый файл, содержащий информацию, которую нужно защитить с помощью цифровой подписи. Документ читается в бинарном режиме, чтобы сохранить его исходное содержимое без изменений.

После чтения документа происходит чтение приватного ключа пользователя 1. Приватный ключ используется для создания цифровой подписи. Он читается из файла, который был создан на предыдущем шаге. Приватный ключ десериализуется из формата PEM с помощью функции “serialization.load_pem_private_key, чтобы его можно было использовать для создания подписи.

Следующим шагом является создание цифровой подписи. Подпись создается с использованием алгоритма RSA и хеш-функции SHA-256. Данное действие происходит при выполнении метода “sign” приватного ключа. Метод “sign” принимает три аргумента: данные документа, объект “padding” и объект “hashes”. Объект “padding.PKCS1v15()” задает алгоритм паддинга, а объект “hashes.SHA256()” задает хеш-функцию.

После подписания файла подпись сохраняется в бинарном формате, чтобы ее можно было использовать для последующей проверки. Файл с

подписью может быть передан вместе с документом для проверки его подлинности.

Для подмены подписи используется скрипт “replace_signature.py”, в котором происходит подмена подписи пользователя 1 подписью пользователя 2 (рис. 3).

```
def replace_signature(document_path, private_key_path, signature_path):  
  
    with open(document_path, "rb") as f:  
        document_data = f.read()  
  
    with open(private_key_path, "rb") as f:  
        private_key = serialization.load_pem_private_key(  
            f.read(),  
            password=None,  
        )  
  
    new_signature = private_key.sign(  
        document_data,  
        padding.PKCS1v15(),  
        hashes.SHA256()  
    )  
  
    with open(signature_path, "wb") as f:  
        f.write(new_signature)  
  
replace_signature( document_path: "document.txt", private_key_path: "user2_private_key.pem", signature_path: "user1_signature.bin")
```

Рисунок 3. Скрипт подмены подписей

Функция “replace_signature” принимает три аргумента: путь к документу, путь к приватному ключу пользователя 2 и путь к файлу подписи. Вначале документ читается в бинарном режиме с помощью функции “open” и метода “read”. Затем приватный ключ пользователя 2 читается из файла с помощью таких же методов. Полученный ключ десериализуется из формата PEM с помощью функции “serialization.load_pem_private_key”. После считывания ключей происходит создание новой подписи с использованием метода “sign” приватного ключа пользователя 2. Данный метод принимает три аргумента: данные документа, объект “padding” и объект “hashes”. В конце происходит сохранение подменной подписи.

В последнем скрипте “verify_signature.py” выполняется проверка подписи документа и выявление подмены (рис. 4). Кроме проверки подмены подписи происходит и проверка содержимого документа. Это означает то, что если после подписания были внесены как-либо изменения в файле, то программа выдаст аналогичную ошибку, если бы были подменены подписи.

```
def verify_signature(document_path, signature_path, public_key_path):  
    with open(document_path, "rb") as f:  
        document_data = f.read()  
  
    with open(signature_path, "rb") as f:  
        signature = f.read()  
  
    with open(public_key_path, "rb") as f:  
        public_key = serialization.load_pem_public_key(  
            f.read()  
        )  
  
    try:  
        public_key.verify(  
            signature,  
            document_data,  
            padding.PKCS1v15(),  
            hashes.SHA256()  
        )  
        print("Подпись верна.")  
    except Exception as e:  
        print("Подпись неверна:", e)  
  
verify_signature(document_path: "document.txt", signature_path: "user1_signature.bin", public_key_path: "user1_public_key.pem")
```

Рисунок 4. Скрипт проверки подлинности подписи

В данном скрипте происходит открытие и чтение документа, приватного и публичного ключей. Затем подпись проверяется с использованием метода “verify” публичного ключа. Метод “verify” принимает четыре аргумента: подпись, данные документа, объект “padding” и объект “hashes”. Объект “padding.PKCS1v15()” задает алгоритм паддинга, а объект “hashes.SHA256()” задает хеш-функцию.

Если подпись верна, выводится сообщение “Подпись верна.”. Если подпись неверна, выводится сообщение “Подпись неверна.”.

Программа реализована и готова к тестированию (рис. 5). В первом тесте успешно прошли проверку подписанный файл и приложенная к нему подпись. Во втором случае, подпись у файла была подменена на другую, что привело к соответствующему результату программы. В третьем тесте подпись не менялась, но был изменён уже подписанный файл, что привело к результату из второго теста.

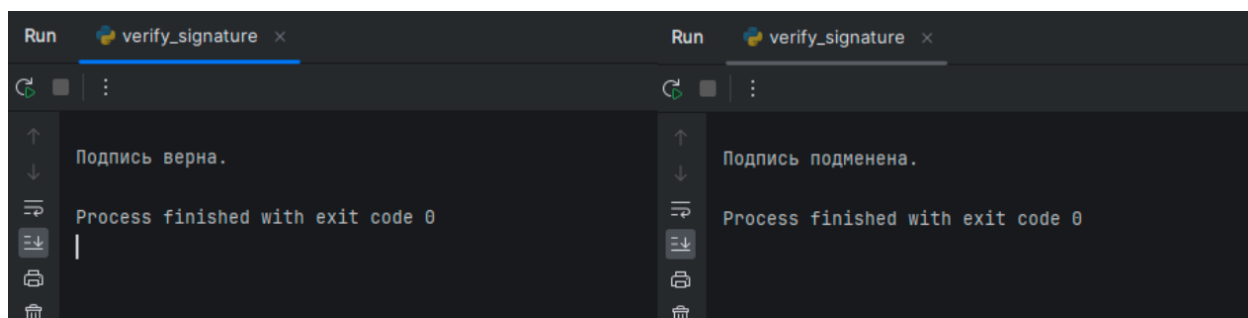


Рисунок 5. Результат работы программы – слева успешная проверка подлинности, справа выявление подмены или факта редактирования файла после подписания

Вывод

В ходе выполнения данной работы была разработана программа для подписания документов электронной подписью на языке программирования Python. Программа включает генерацию ключей и сертификатов, подписание документа, подмену подписи и проверку подлинности подписи. В ходе тестирования были получены результаты, подтверждающие правильность и корректность работы системы.

Библиографический список

1. Аль-Рубайе В.Х.Н. Методы криптографии // В сборнике: Академическая наука на службе обществу. Сборник статей Международного научно-исследовательского конкурса. Петрозаводск, 2022. С. 153-161.
2. Кустов Д.Н., Конин А.А. Создание модифицированного шифрования атбаша на языке Python // В сборнике: Информационные технологии в процессе подготовки современного специалиста. межвузовский сборник научных трудов. Липецк, 2023. С. 52-55.
3. Бабичева М.В., Савченко Б.С. Анализ шифров с наличием s-блока дифференциальным методом // Вестник Донецкого национального университета. Серия Г: Технические науки. 2018. № 4. С. 42-48.
4. Сейткулов Е., Оспанов Р., Ташатов Н., Ералиева Б., Сисенов Н. Программный инструмент для анализа и синтеза криптографических s-блоков // Вестник Казахской академии транспорта и коммуникаций им. М. Тынышпаева. 2023. № 3 (126). С. 257-266.
5. Костиков Ю.А., Романенков А.М. Методы организации безопасных механизмов авторизации и аутентификации // В сборнике: Сборник научных трудов кафедры прикладной математики и программирования по итогам работы постоянно действующего семинара «Теория систем». М., 2022. С. 172-195.