Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

Лабораторная работа № 4

по дисциплине «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

КРИПТОГРАФИЧЕСКИЕ БИБЛИОТЕКИ

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	6
Студенты:	Сидоров Даниил,
	Дюков Богдан
Преподаватели:	Авдеенко Татьяна Владимировна,
	Кутузова Ирина Александровна.

Новосибирск

2026

1. Цель работы

Познакомиться с существующими криптографическими библиотеками. Научиться использовать сторонние криптографические библиотеки при разработке собственных приложений.

2. Задача

I. Найти криптографическую библиотеку.
II. Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.

1. Шифровать и дешифровать выбранный пользователем файл с использованием одного или нескольких симметричных алгоритмов шифрования:
   1) результаты шифрования и дешифрования должны сохраняться в файлы;
   2) требуемые параметры шифрования, такие как ключ, вектор инициализации и пр., должны считываться из файла.
2. Шифровать и дешифровать выбранный пользователем файл с использованием одного асимметричного алгоритма шифрования:
   1) реализовать процедуру генерации пары открытый–закрытый ключ;
   2) результаты шифрования и дешифрования должны сохраняться в файлы;
   3) требуемые параметры шифрования должны считываться из файла.
3. Вычислять и проверять электронную цифровую подпись для выбранного пользователем файла с использованием одного из алгоритмов:
   1) результаты вычисления подписи должны сохраняться в файлы;
   2) требуемые параметры вычисления и проверки подписи должны считываться из файла.
4. Вычислять хэш-значения для выбранного пользователем файла по одному или нескольким алгоритмам хэширования, при этом вычисленное хэш-значение должно сохраняться в файл.

III. Протестировать правильность работы реализованного приложения.

3. Метод решения задачи

Криптографическая библиотека, которая использовалась при выполнении лабораторной работы, - System.Security.Cryptography C#. Используемые алгоритмы:

1) Симметричные алгоритмы шифрования – DES, AES;
2) Асимметричные алгоритмы шифрования – RSA;
3) Вычисление и проверка цифровой подписи – DSA и ECDSA;
4) Вычисление хэш-значений – SHA512, MD5, RIPEMD-160.

Симметричное шифрование

DES (стандарт шифрования данных) и AES (расширенный стандарт шифрования) представляют собой симметричные блочные алгоритмы шифрования. Открытый

текст в них будет зашифрован в блоках одинаковой длины. AES был разработан для замены DES, который больше не считался достаточно безопасным для современных приложений.

Требуемыми параметрами как для шифрования, так и для дешифрования являются **ключ** и **вектор инициализации** (дополнительный вход, который используется вместе с ключом для шифрования или дешифрования).

DES (класс DESCryptoServiceProvider) поддерживает фиксированную длину ключа и вектора инициализации - оба 64 бита. AES (класс AesCryptoServiceProvider) поддерживает размеры ключей 128, 192 и 256 бит, а размер вектора инициализации фиксированный - 128 бит.

В C# алгоритмы блочного шифрования DES и AES поддерживают 5 режимов, а именно ECB, CBC, CFB, OFB, CTS. В нашем случае использовался режим по умолчанию – CBC (именно этот режим поддерживает вектор инициализации).

Действия, которые нужно выполнить при шифровании ассиметричным алгоритмом:

1. **Генерация ключа**. Создается ключ, который будет использоваться для шифрования и дешифрования данных. Важно, чтобы этот ключ был достаточно сложным и случайным, чтобы предотвратить его угадывание или взлом.
2. **Выбор вектора инициализации (IV)**. IV - случайное число, которое используется для начальной инициализации процесса шифрования. IV обеспечивает, чтобы даже одинаковые сообщения шифровались по-разному, что увеличивает безопасность.
3. **Шифрование**. Исходный текст преобразуется в шифротекст с использованием ключа и IV. Этот процесс обычно включает в себя серию математических операций, которые применяются к каждому блоку исходного текста.
4. **Передача и расшифровка данных**. Зашифрованные данные передаются получателю, который может использовать тот же ключ и IV для преобразования шифротекста обратно в исходный текст.

Асимметричное шифрование

RSA (Rivest-Shamir-Adleman) — это алгоритм асимметричного шифрования, который используется для безопасной передачи данных в интернете. Он основан на сложности факторизации больших чисел. В RSA используются два ключа: открытый и закрытый. Открытый ключ используется для шифрования данных, а закрытый ключ - для их расшифровки.

Класс RSACryptoServiceProvider, реализующий алгоритм RSA в используемой библиотеке, поддерживает размеры ключей от 384 бит до 16384 бит с шагом в 8 бит.

Действия, которые нужно выполнить при шифровании асимметричным алгоритмом:

1. **Генерация ключей**. Создаются два ключа - открытый и закрытый. Открытый ключ распространяется и доступен всем, в то время как закрытый ключ держится в секрете.
2. **Шифрование**. Отправитель использует открытый ключ получателя для шифрования сообщения. Зашифрованное сообщение затем отправляется получателю.
3. **Расшифровка**. Получатель использует свой закрытый ключ для расшифровки зашифрованного сообщения.

Если получатель хочет ответить, он повторяет процесс, описанный выше, но уже использует открытый ключ отправителя.

Вычисление и проверка цифровой подписи

DSA (Digital Signature Algorithm) и ECDSA (Elliptic Curve Digital Signature Algorithm) — это два алгоритма, используемые для создания и проверки цифровых подписей.

DSA основан на сложности вычисления дискретных логарифмов в конечном поле. ECDSA является вариант DSA, который использует эллиптические кривые. Это делает ECDSA более эффективным, поскольку он обеспечивает тот же уровень безопасности, что и DSA, но с меньшими ключами.

Оба алгоритма используют пару ключей: открытый и закрытый. Закрытый ключ используется для генерации цифровой подписи, а открытый ключ - для ее проверки.

Класс ECDSA из библиотеки поддерживает различные размеры ключей. Размер ключа зависит от выбранной эллиптической кривой. Некоторые общие размеры ключей: 256 бит, 384 бит, 521 бит. Закрытый ключ представляет собой число, которое используется в качестве секретного множителя для генерации открытого ключа. Открытый ключ представляет собой точку на эллиптической кривой. Эта точка представляется двумя координатами. Поэтому открытый ключ выводится как два числа, разделенных запятой.

Существуют две разные версии алгоритма DSA. Наша версия, описанная в FIPS 186-2, поддерживает длину ключа от 512 бит до 1024 бит с шагом в 64 бита. Открытый ключ представляет собой число, вычисленное на основе закрытого ключа и некоторых общедоступных параметров (P, Q, G). Закрытый ключ представляет собой случайное число, меньшее параметра Q.

Также оба алгоритма принимают на вход выбранный алгоритм хэширования. ECDSA может использоваться с различными алгоритмами хеширования, тогда как

DSA версия, описанная в FIPS 186-2, требует использования SHA-1 в качестве хэш – алгоритма.

Действия, которые нужно выполнить при вычислении и проверке электронной цифровой подписи:

1. **Выбор алгоритма хеширования**. Выбирается алгоритм хеширования, который будет использоваться для создания хэша от исходного сообщения. Это может быть любой криптографический алгоритм хеширования, такой как SHA-256, MD5, RIPEMD-160 и др.
2. **Генерация ключей**. Создаются два ключа - открытый и закрытый. Открытый ключ распространяется и доступен всем, в то время как закрытый ключ держится в секрете.
3. **Создание подписи**. Для создания подписи сначала вычисляется хэш от исходного сообщения с использованием выбранного алгоритма хеширования. Затем этот хэш подписывается с использованием закрытого ключа. Когда говорится, что “хэш подписывается с помощью закрытого ключа”, имеется в виду, что используется алгоритм цифровой подписи, который принимает хэш и закрытый ключ в качестве входных данных и выдает цифровую подпись. Примером такого алгоритма является RSA.
4. **Проверка подписи**. Для проверки подписи сначала вычисляется хэш от того же сообщения с использованием того же алгоритма хеширования. Затем с помощью открытого ключа проверяется, соответствует ли подпись этому хэшу.

Цифровая подпись играет важную роль в обеспечении безопасности данных. Она обеспечивает **аутентификацию** (получатель может проверить источник сообщения), **целостность** (получатель может проверить, что сообщение не было изменено после подписания) и **невозможность отказа** (отправитель не может ложно утверждать, что он не подписывал сообщение).

Вычисление хэш-значений

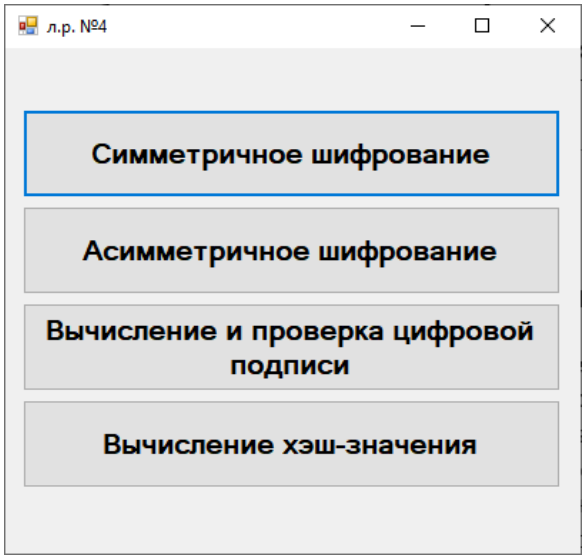
Вычисление хэш-значений с помощью алгоритмов SHA512, MD5 и RIPEMD160 основано на преобразовании входных данных в уникальное хэш-значение фиксированной длины.

SHA512 преобразует текст любой длины в строку фиксированного размера, каждый вывод создает длину в 512 бит. MD5 используется для создания хэш-функции, производящей 128-битное хэш-значение. RIPEMD160 используется в стандарте Bitcoin и производит выходные данные длиной 160 бит.

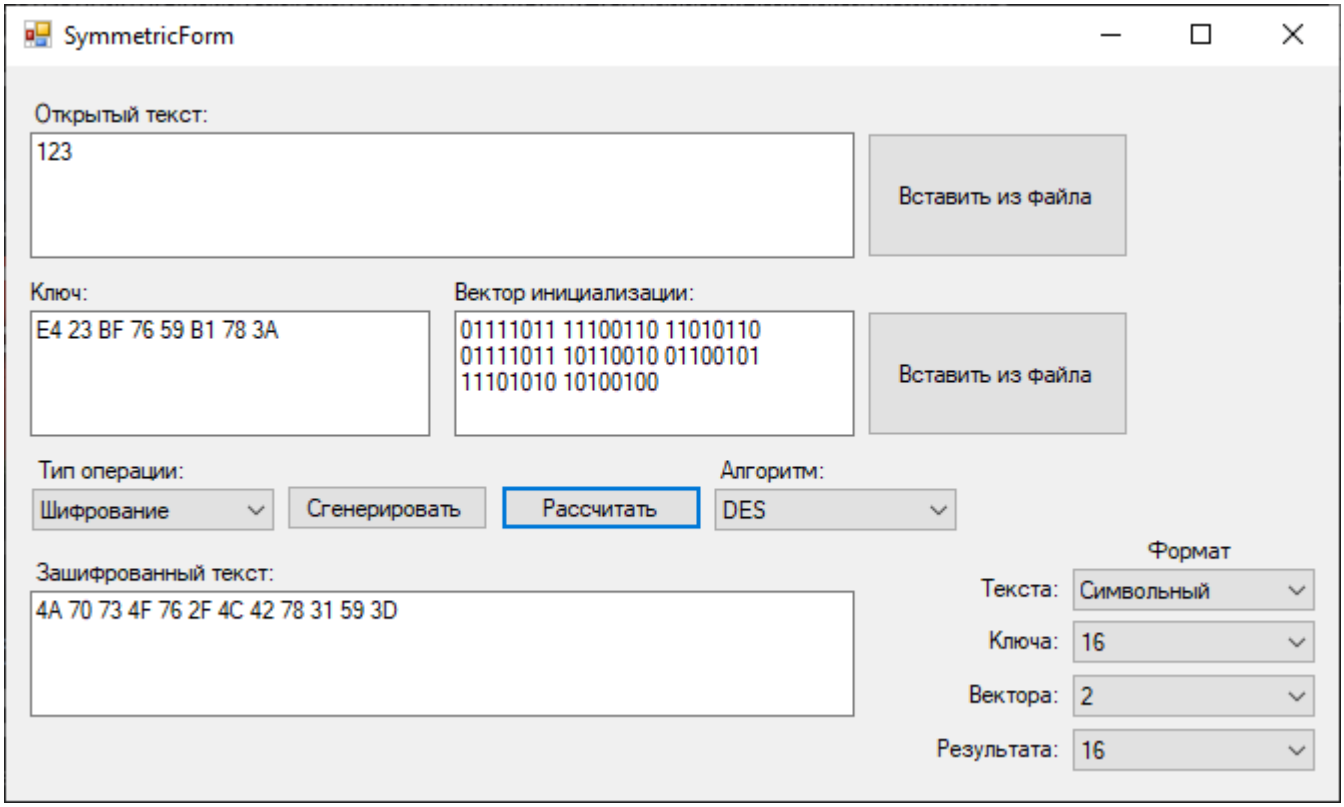
Хэш-значения могут использоваться только один раз для аутентификации данных или цифровых подписей, или они могут храниться для быстрого поиска в хэш-таблице.

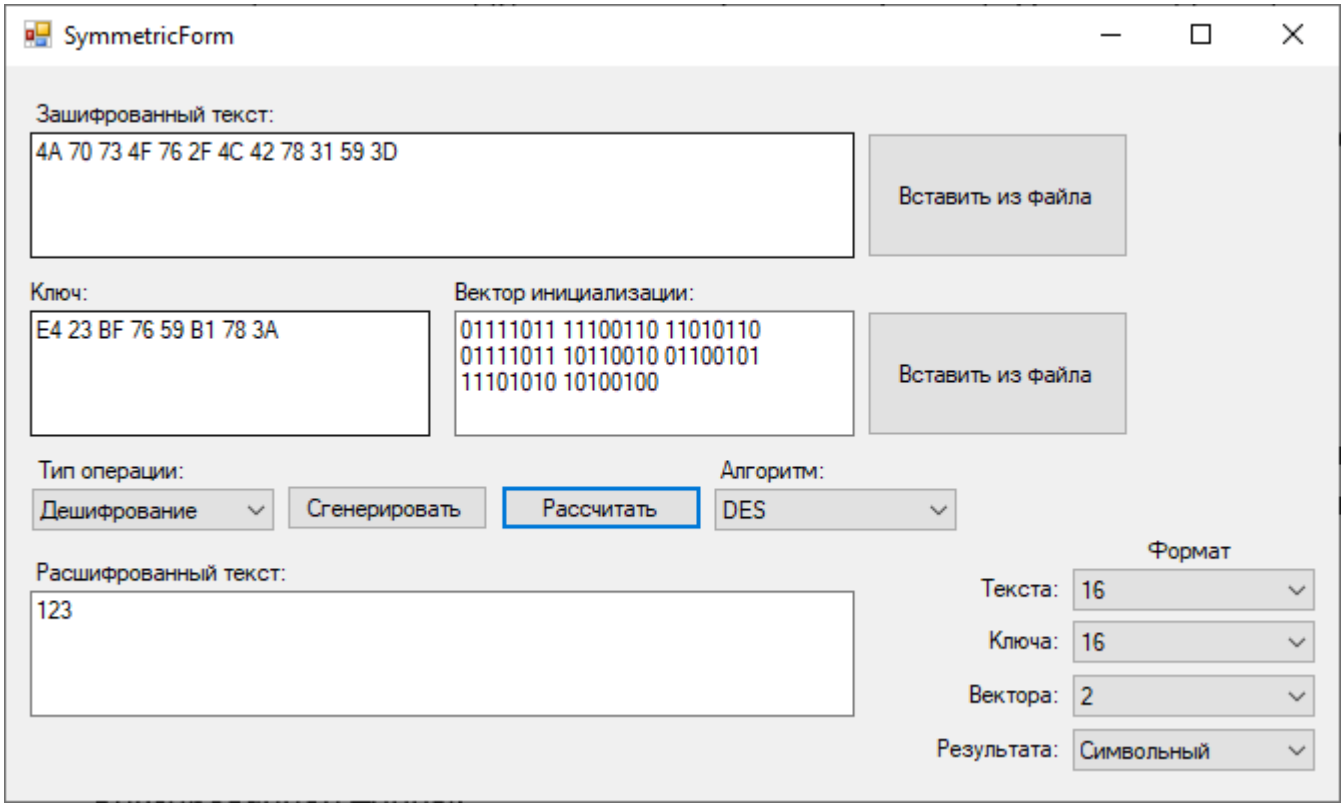
4. Разработанное программное средство

Разработанное программное средство представляет собой приложение Windows Forms. Интерфейс главного меню:

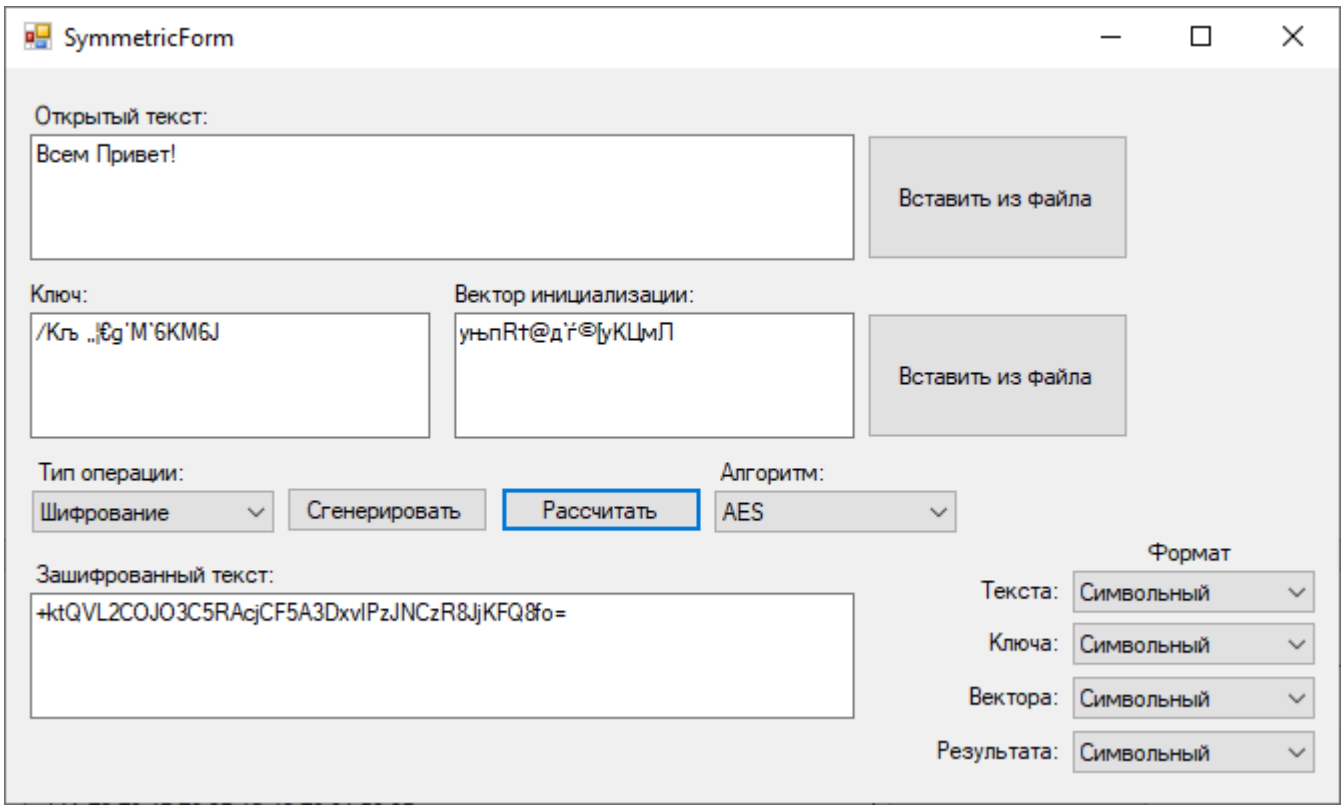


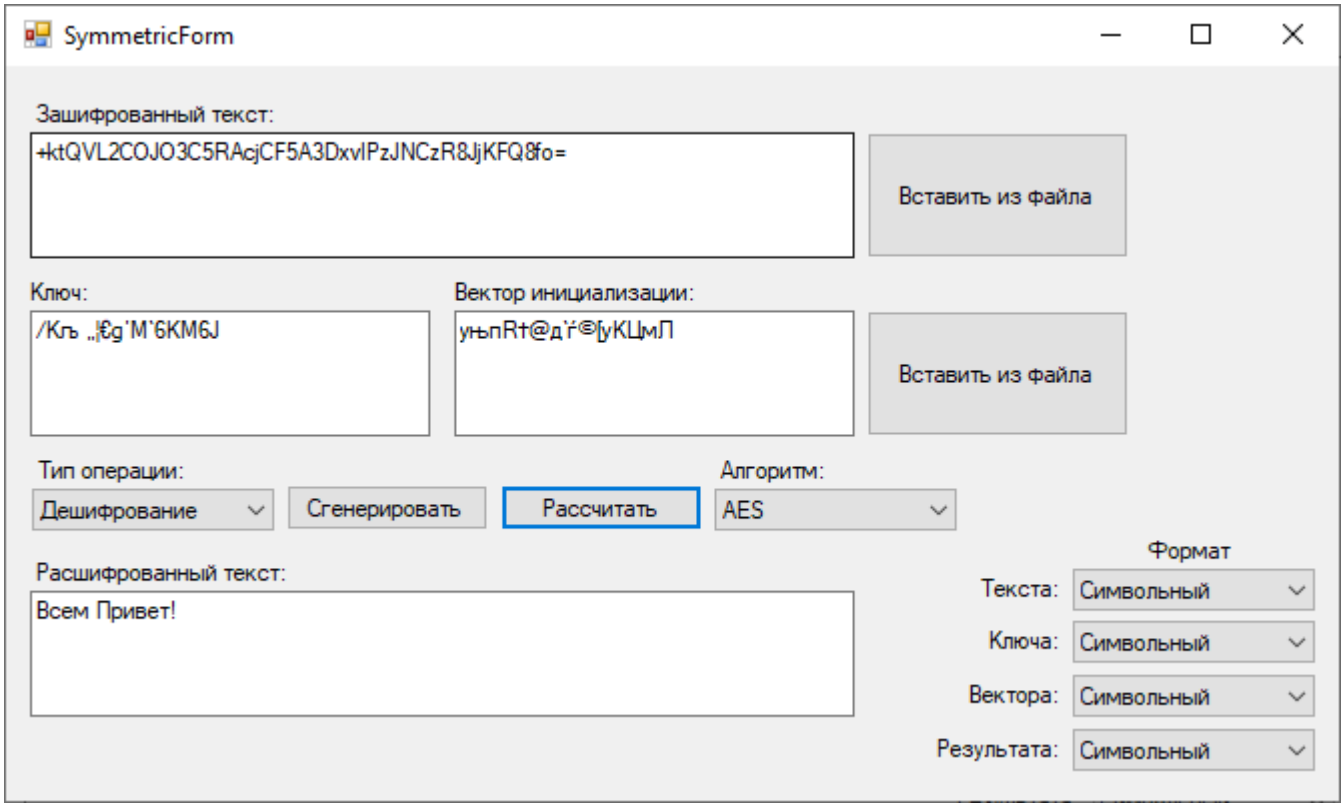
Перейдем к окну с симметричным шифрованием, сразу выполним шифрование и дешифрование алгоритмом DES:





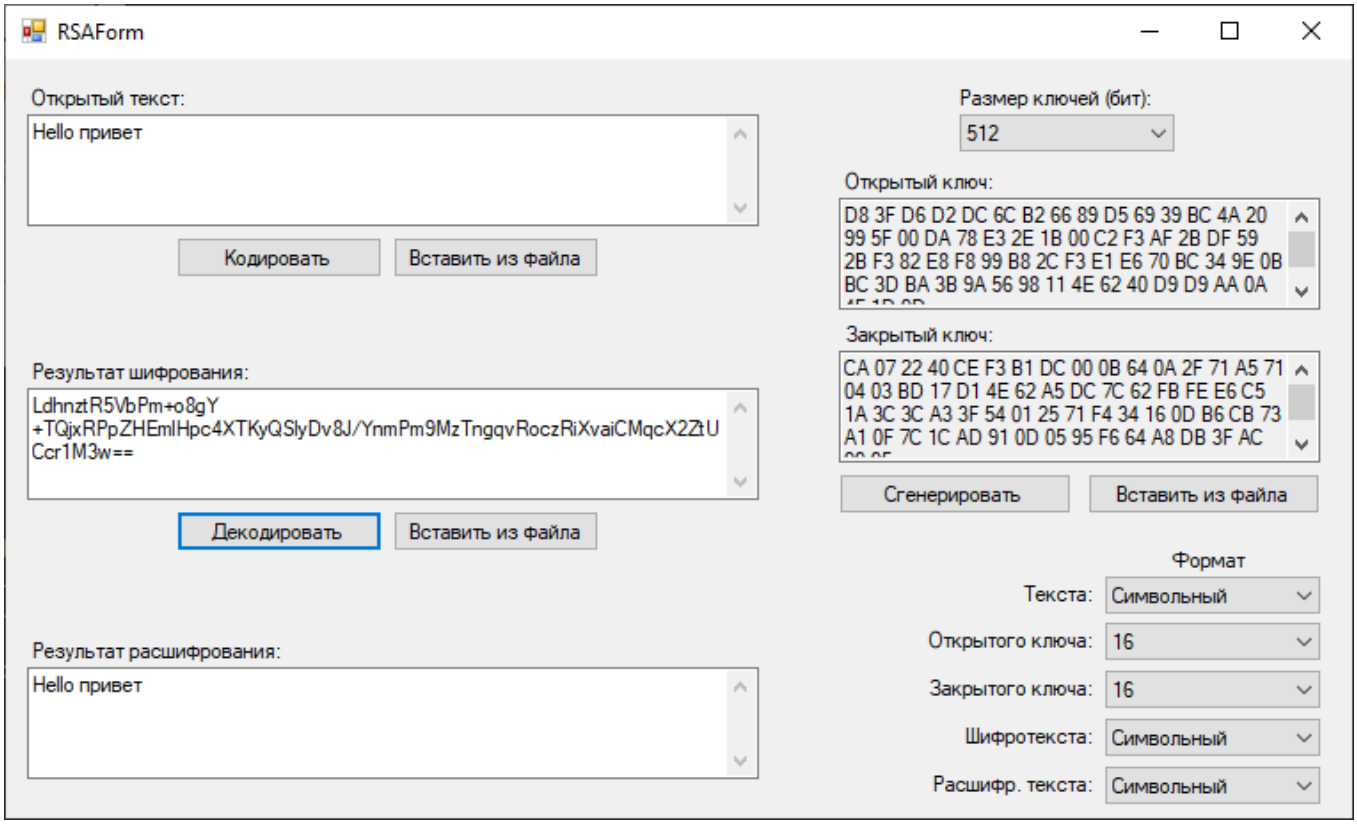
Выполним аналогичные действия, но с использованием алгоритма AES:



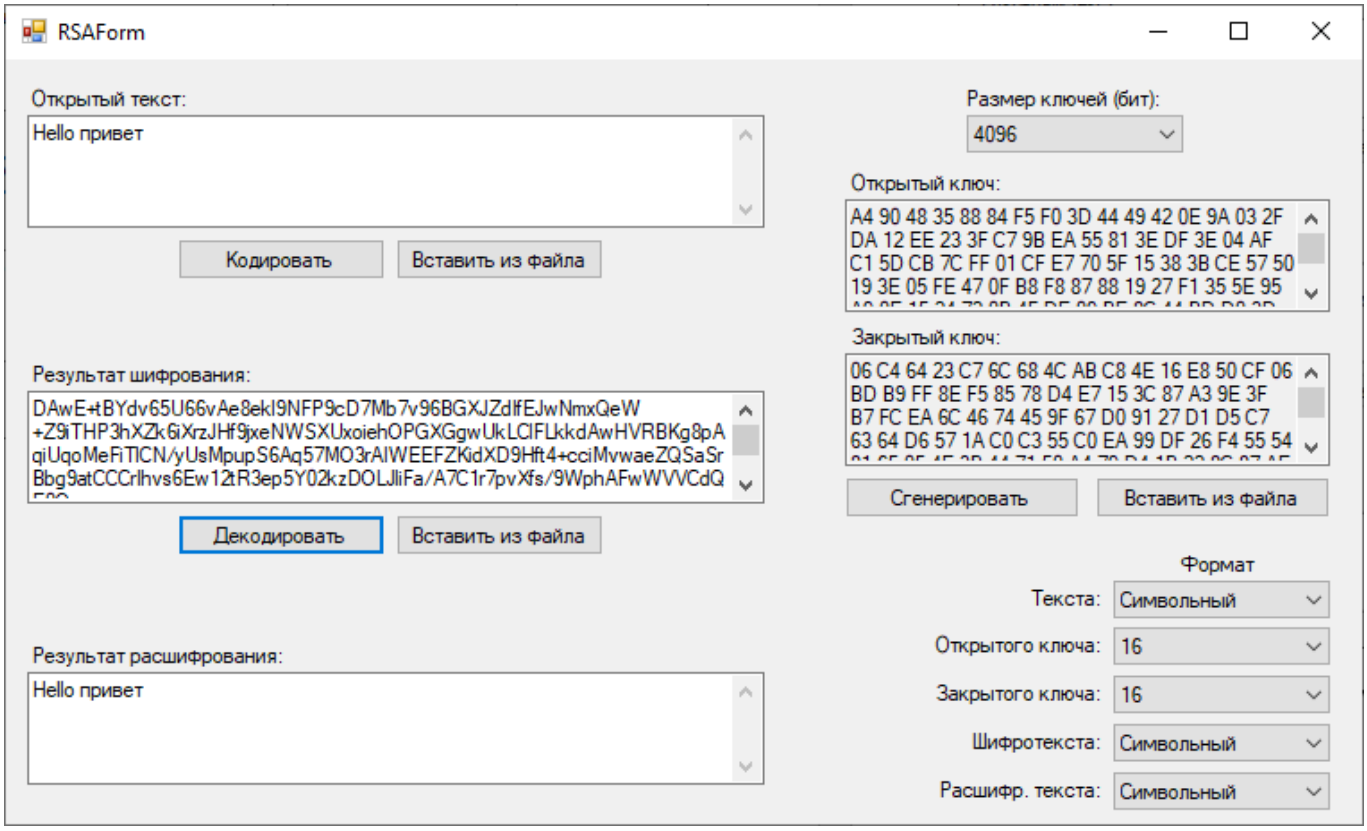


Во время шифрования открытый текст, ключ, вектор инициализации и шифротекст сохраняются в соответствующие файлы. Результат дешифрования также сохраняется в файл.

Перейдем к асимметричному шифрованию. Зашифруем и расшифруем произвольный текст:

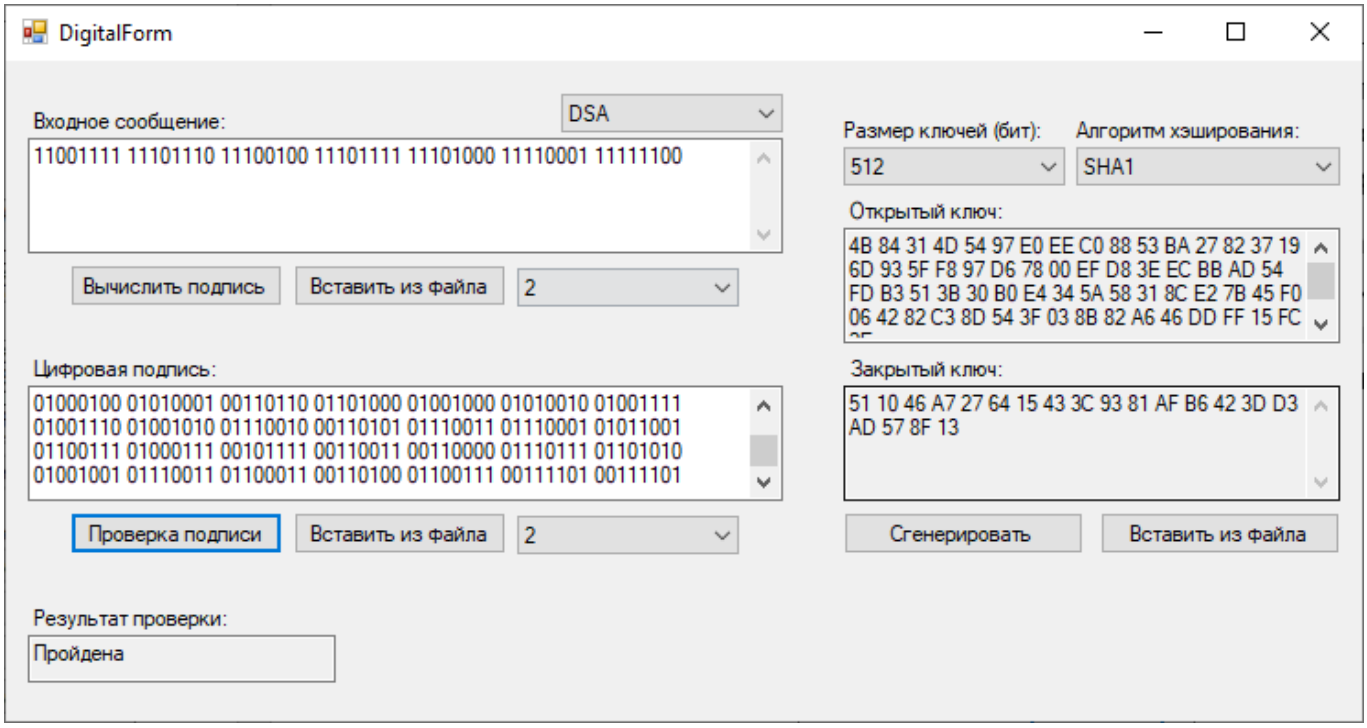


Попробуем другой размер ключей:

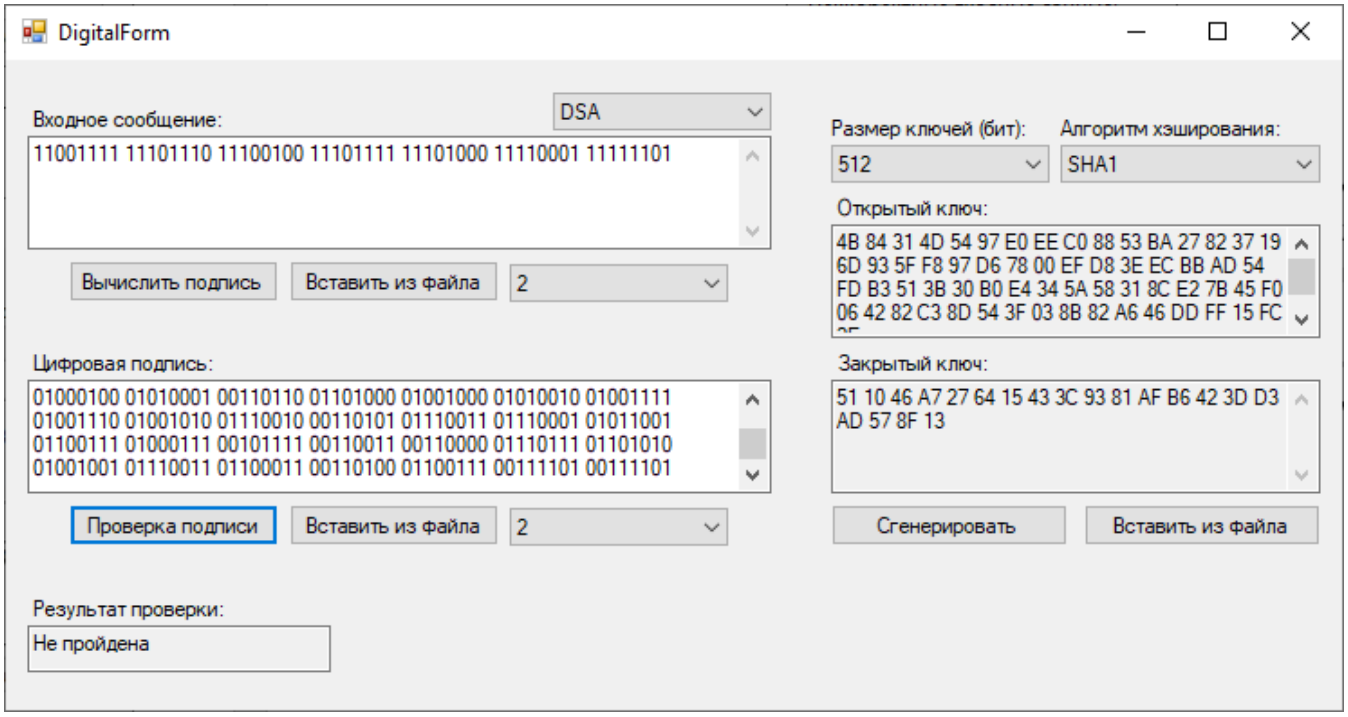


Во время шифрования открытый текст, ключи и шифротекст сохраняются в соответствующие файлы. Ключи сохраняются в формате XML. Результат дешифрования также сохраняется в файл.

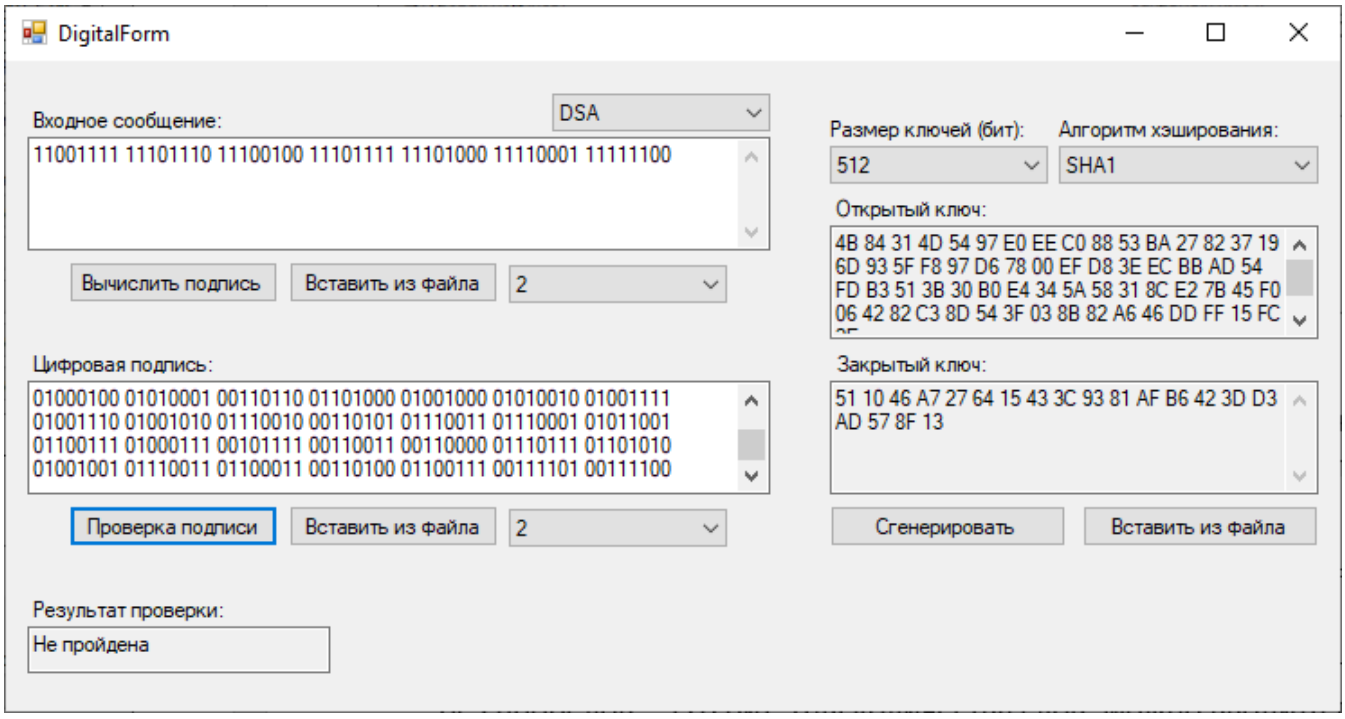
Перейдем к цифровой подписи. С помощью DSA вычислим цифровую подпись и сразу проверим её:



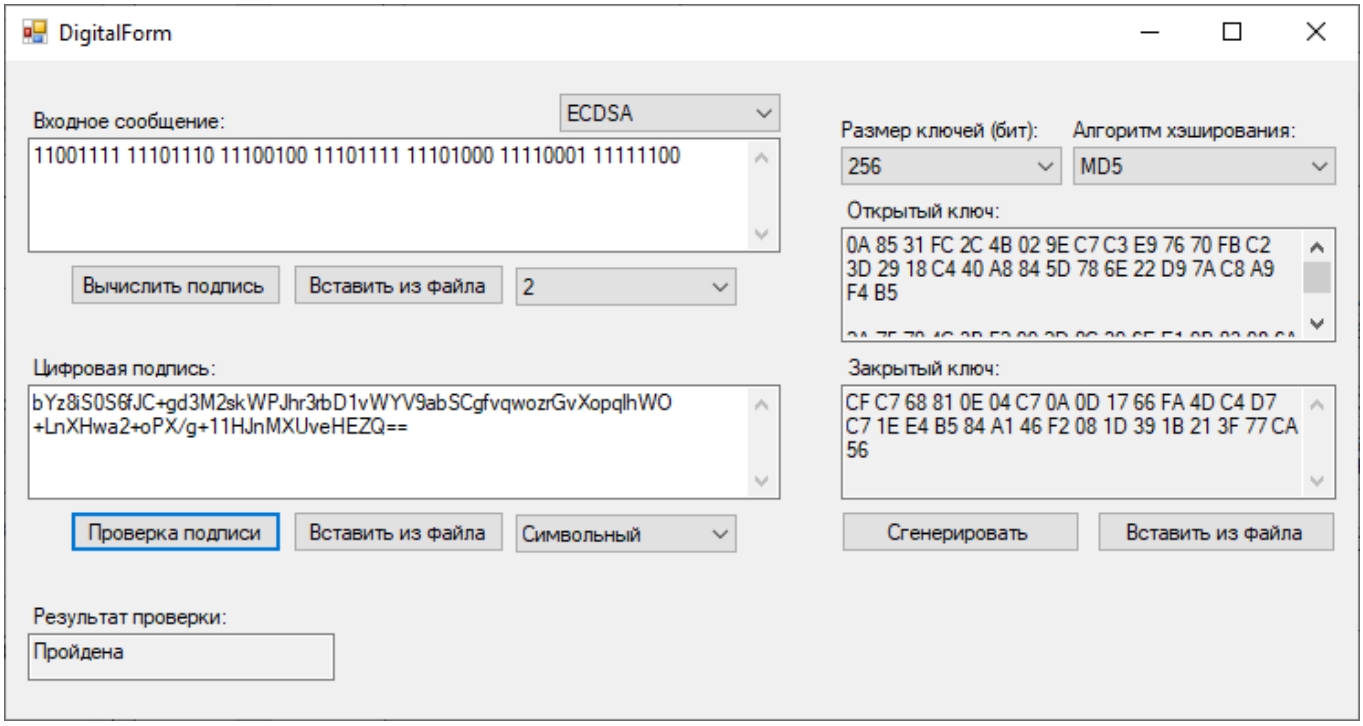
Убедимся, что малейшая модификация исходного сообщения (модификация последнего бита) приводит к отрицательному результату проверки:

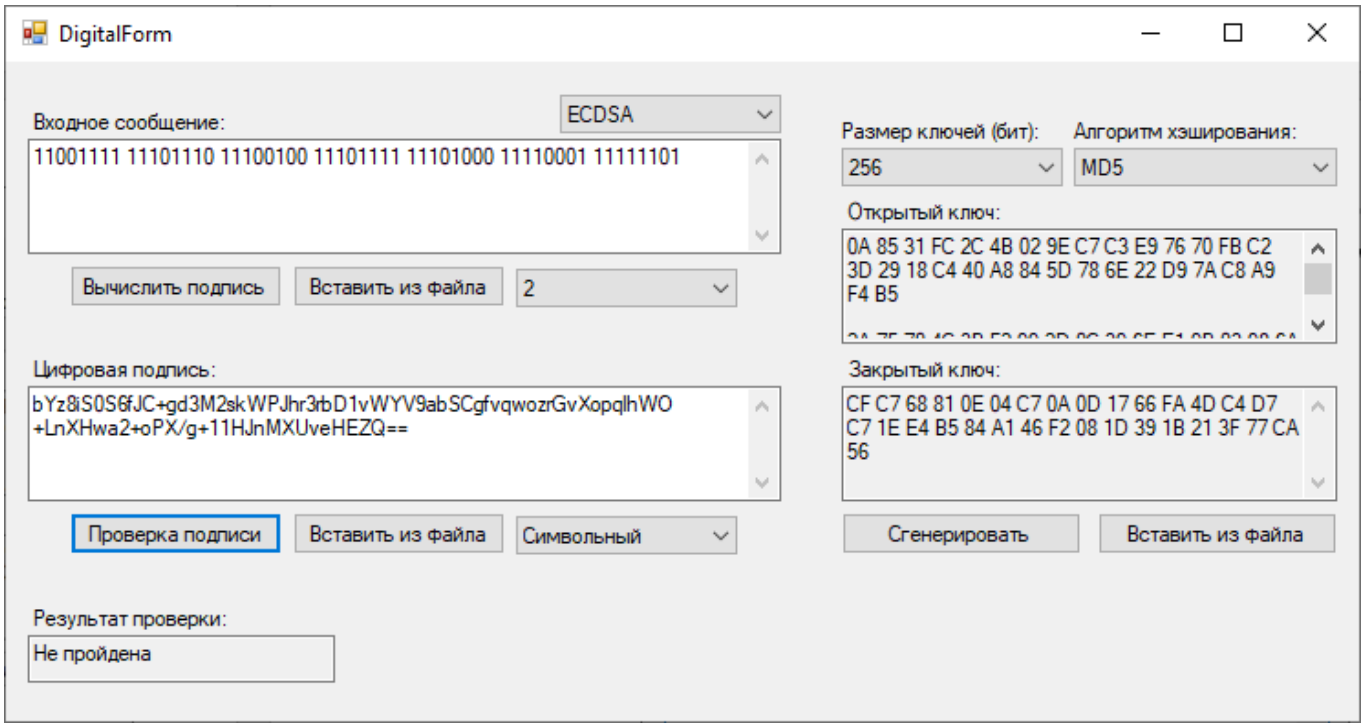


Аналогично с модификацией цифровой подписи (модификация последнего бита):



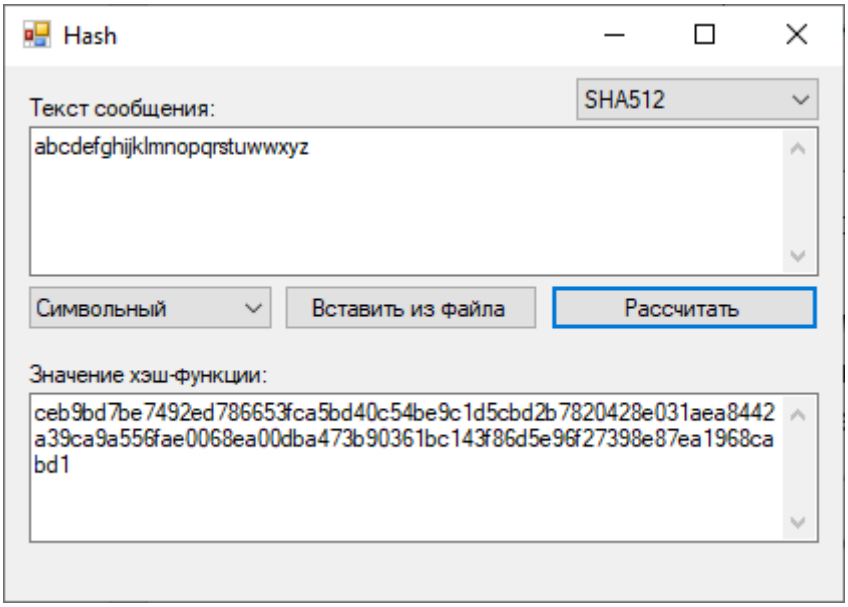
Вычислим цифровую подпись с помощью ECDSA, проверим ее, а потом проверим корректность проверки при модификации исходного сообщения:

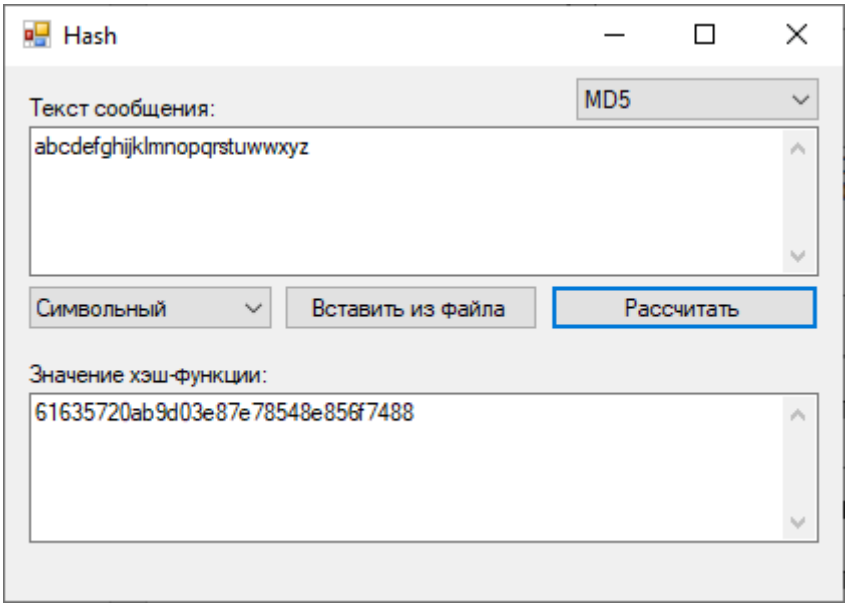


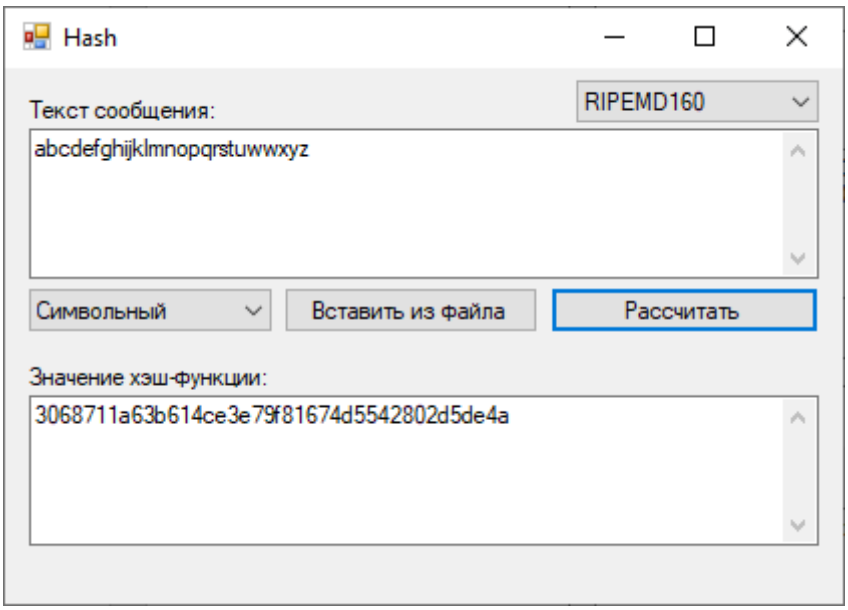


При вычислении подписи в соответствующие файлы сохраняются исходное сообщение, полученная подпись и ключи (формат XML и JSON для DSA и ECDSA соответственно). После проверки подписи результат проверки также сохраняется в файл.

Перейдем к вычислению хэш-значений. Получим хэши для одного и того же сообщения разными алгоритмами:







Исходный текст и вычисленное хэш-значение сохраняются в соответствующие файлы.