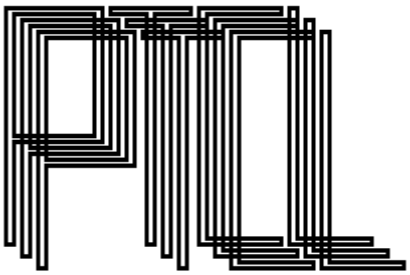


# PlexusTCL Crypter

версия 4.51 от 27 мая 2020 года

**ВАЖНЫЕ ПРЕДУПРЕЖДЕНИЯ**

**1.) ЕСЛИ ВЫ НЕ УВЕРЕНЫ В СТАБИЛЬНОСТИ РАБОТЫ ПРОГРАММЫ ИЛИ ВАШЕГО КОМПЬЮТЕРА, ТО ПЕРЕД ШИФРОВАНИЕМ ФАЙЛА, ОБЯЗАТЕЛЬНО СДЕЛАЙТЕ ЕГО РЕЗЕРВНУЮ КОПИЮ.**

**2.) ПРОГРАММА НЕ ГАРАНТИРУЕТ ТАЙНУ, ЕСЛИ ВЫ ДОПУСКАЕТЕ ВОЗМОЖНОСТЬ АТАК ПО СТОРОННИМ КАНАЛАМ, ТАКИХ КАК ОТПЕЧАТОК (ДАМП) ОПЕРАТИВНОЙ ПАМЯТИ ВО ВРЕМЯ РАБОТЫ ПРОГРАММЫ, ПОПАДАНИЕ КЛЮЧЕЙ ШИФРОВАНИЯ В ФАЙЛ ПОДКАЧКИ, ЗАРАЖЕНИЕ КОМПЬЮТЕРА ВРЕДОНОСНОЙ ПРОГРАММОЙ И Т.Д.**

**3.) ЧТОБЫ ОБЕСПЕЧИТЬ МАКСИМАЛЬНО ВОЗМОЖНЫЙ УРОВЕНЬ БЕЗОПАСНОСТИ ПРИ ПРИМЕНЕНИИ НАСТОЯЩЕГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ПРОКОНСУЛЬТИРУЙТЕСЬ СО СПЕЦИАЛИСТАМИ В ОБЛАСТИ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ.**

Программное обеспечение «PlexusTCL Crypter» предназначено для криптографической защиты информации, путем шифрования файлов, размером до 2 Гб включительно. Файлы могут быть обработаны пятью криптографическими алгоритмами по выбору пользователя, а именно ARC4, Rijndael (далее AES), Serpent, Blowfish и Threefish, с использованием ключевого файла или введенной в качестве пароля строки.

## Программа и алгоритмы

Программное обеспечение представляет собой исполняемый файл с интегрированным GUI (графическим интерфейсом пользователя), в котором реализованы алгоритмы шифрования и система управления вводом/выводом информации в виде вызываемых функций.

Все реализации криптографических алгоритмов, а именно ARC4, AES, Serpent, Blowfish и Threefish, протестированы с помощью тестовых векторов и полностью соответствуют своим математическим описаниям или опубликованным стандартам. Математическое описание шифра ARC4 (на самом деле RC4, от Ronald Cipher 4) никогда не было опубликовано ни своим создателем, Роном Райвестом, ни кем либо из сотрудников компании RSA Security, так как по прежнему является коммерческой тайной, но программная реализация алгоритма ARC4 полностью соответствует анонимно опубликованному в 1994 году

исходному коду на языке C (Си). Компиляция этого исходного кода давала программу, которая принимая случайные ключ шифрования и поток открытого текста, всегда давала такой же шифротекст как и лицензионный RC4, а значит полностью совместима с лицензионными продуктами RSA Security, поддерживающими этот шифр. Само сочетание букв «RC4» является торговой маркой, принадлежащей RSA Security, по этому в среде «free software» алгоритм шифрования принято называть ARC4, т.е (англ. Alleged RC4) предполагаемый RC4.

При использовании блочного шифра из программы «PlexusTCL Crypter», важно учитывать тот факт, что шифр AES отличается от шифра Threefish тем, что Threefish спроектирован как 64 битный шифр, т.е оперирующий 64 битными (8 байтными) числами как самостоятельными единицами, в то время как AES оперирует блоками данных, состоящими из 8 битных (1 байтных) значений. AES шифрует открытый текст блоками битов, обрабатывая по 8 битов (1 байту) за операцию, в то время как Threefish принимает 64 бита (8 байтов) открытого текста за одно большое число, и оперирует им как одним целым. По этому, скорость работы Threefish на 64 битных процессорах, в разы больше скорости его работы на 32 битных процессорах, но скорость работы на 32 битных процессорах, в 2 — 3 раза ниже, чем скорость работы AES. По этому, в случае если важна скорость шифрования, рекомендуется использовать Threefish на 64 битных процессорах, а AES на 8, 16 и 32 битных. К тому же, реализация шифра AES не оптимизирована, по этому не рекомендуется использовать его для обработки больших файлов (20, 50, 100 Мб и т.д), так как обработка может занять десятки минут, что не относится к остальным шифрам в программе.

Использовать шифры Blowfish и Serpent можно на любом 32 или 64 битном процессоре, так как эти шифры показывают почти одинаковую производительность на обоих, к тому же их реализации очень быстрые по сравнению с Threefish и тем более AES. Blowfish шифрует данные блоками по 32 бита, шифруя сразу 2 части открытого текста в виде 64 битного блока данных, принимая на вход левую и правую 32 битные части данных по отдельности, заменяя их шифротекстом, как и любые другие шифры спроектированные на основе сети Фейстеля. Так как оба шифра спроектированы 32 битным, это может сказаться на производительности при использовании шифра на 64 битных платформах в лучшую сторону, но разработчик не заметил разницы. Скорость работы обоих шифров на 32 битном и 64 битном процессорах почти одинаковая.

Самым быстрым из всех шифров программы «PlexusTCL Crypter», да и вообще в мире, является шифр ARC4. Он был спроектирован американским математиком Рональдом Райвестом уже очень давно, и не использовался наверное только в операционных системах, созданных до появления шифра. Он интегрирован в программу не только из-за того, что трудно найти надежный и проверенный временем шифр, работающий с такой высокой скоростью, но и из-за его исторической значимости. Его можно спокойно использовать только с ключами длиной 2048 битов (256 байтов) и никогда не использовать один ключ дважды.

Все четыре блочных шифра, а именно AES, Threefish, Blowfish и Serpent, работают в режиме CFB (режим обратной связи по шифротексту), что обеспечивает достаточно надежный уровень безопасности применения блочного шифра, превращая блочный шифр в поточный. Не стоит беспокоится на этот счет, так как правильно использованный блочный шифр в виде поточного, ничем не уступает самому криптостойкому блочному шифру. При шифровании данных в режиме CFB, блочный шифр вообще не используется для шифрования данных а используется для генерации псевдослучайной последовательности битов, которая складывается по модулю 2 с каждым битом шифруемых данных.

Чтобы зашифровать что-либо в помощью блочного шифра, работающего в режиме CFB, сначала необходимо сгенерировать случайную или псевдослучайную последовательность, называемую IV (вектором инициализации), зашифровать IV с помощью блочного шифра, чтобы получившуюся последовательность побитно сложить по модулю 2 с шифруемым блоком данных, получая шифротекст. После окончания шифрования первого блока данных, в качестве IV при шифровании каждого последующего блока, используется предыдущий зашифрованный блок, что обеспечивает лавинное изменение всего шифротекста при изменении даже 1 бита ключа шифрования, шифруемого блока или IV. Если выразить режим CFB символами, то получится что-то вроде:

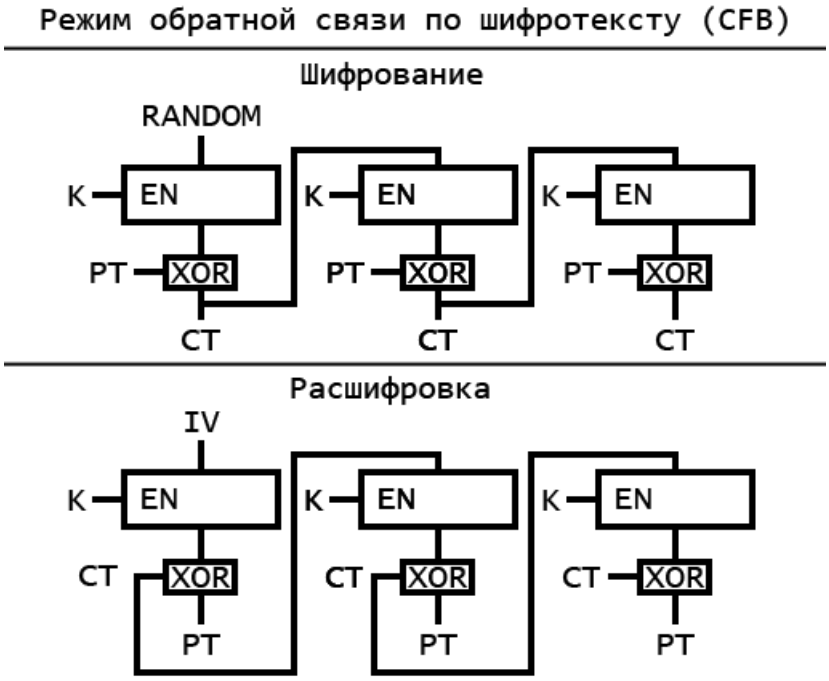
```
iv   = random(time(now));
c[0] = e(iv, key);
c[i] = p[i] xor e(c[i-1], key); от i=1 до i=n;
```

*где:*

- *random – генератор псевдослучайных чисел вашей ОС*
- *time – системное время вашей ОС в секундах*
- *now – реальное время*
- *xor – операция побитового исключающего ИЛИ*
- *key – ключ шифрования*
- *iv – вектор инициализации*
- *n – длина шифротекста в блоках*
- *i – увеличивающийся на единицу счетчик*
- *p – открытый текст*
- *c – шифротекст*
- *e – функция шифрования*

Даже при шифровании файла размером 2 Гб, полностью состоящего из нулевых битов, используя в качестве ключа шифрования и IV последовательность нулей, зашифрованный файл будет выглядеть нагромождением случайных данных, не имеющих никакой закономерности. В программе «PlexusTCL Crypter», режим CFB реализован так, что **в качестве IV, используется зашифрованная псевдослучайная последовательность**, записываемая в файл перед первым зашифрованным блоком, что необходимо для расшифровки и нисколько не сказывается на криптостойкости, так как знание IV ничем не поможет при взломе шифротекста без знания ключа, а ключ всегда должен быть самым охраняемым секретом. Знание того, что IV

записан в файл первым зашифрованным блоком, никак не поможет взломать шифротекст, если не известен ключ.




При шифровании двух открытых текстов, первые шифруемые блоки которых совпадают, уникальный для каждого открытого текста вектор инициализации, используется в режиме CFB для сокрытия этого совпадения, если для шифрования используется один и тот-же ключ. Псевдослучайный для каждого открытого текста IV, позволяет использовать «долгосрочный ключ», т.е один ключ для шифрования множества открытых текстов, даже если они совпадают. Так как в настоящей программе в качестве IV используется зашифрованная псевдослучайная последовательность, генерируемая используемой ОС, то это представляет угрозу только в том случае, если будет атакован ГПСЧ (генератор псевдослучайных чисел). К примеру, если в ОС остановлено системное время, то генератор псевдослучайных чисел будет бесконечно генерировать одно и то же число от 0 до 255 (от 0x00 до 0xFF), так как «зерном» для генератора выступает реальное системное время в секундах. Если злоумышленник знает открытый текст, шифрование которого дает IV, то злоумышленник может попытаться восстановить ключ, и расшифровать все сообщения, зашифрованные тем же ключом. Если ГПСЧ будет генерировать одни и те же числа, то шифрование одинаковых блоков открытого текста при использовании одного и того же ключа, будет давать одинаковые шифротексты, что дает возможность взломать шифротекст. Чтобы этого избежать, нужно следить за тем, что делает прочее программное обеспечение на используемом компьютере (не подменяет ли оно что-нибудь), избегать использования «пиратских» копий программного обеспечения которые могут быть заражены вредоносным кодом и никогда ничего не шифровать в случае получения сообщения «Критическая ошибка ГПСЧ!», что означает некорректную работу ГПСЧ (три псевдослучайных байта IV равны, а такого никогда не должно быть). Чтобы увеличить случайность и одновременно уменьшить

вероятность повторения значений в IV, его первые два байта складываются по модулю два с координатами курсора мыши по осям X и Y, которые вычисляются в момент нажатия кнопки «Старт», что вносит в IV элемент неопределенности (злоумышленник не может сказать, в каком именно положении был курсор мыши в момент нажатия кнопки).

Поточный шифр ARC4, работает в режиме OFB (режим обратной связи по выходу), при использовании которого, в зависимости от ключа шифрования генерируется поток псевдослучайных битов, каждый бит которого складывается по модулю 2 с каждым битом шифруемого текста, давая на выходе шифротекст. Режим OFB это тот же режим CFB, но отличается тем, что в режиме OFB шифротекст не используется для генерации ключа шифрования открытого текста. Поточные шифры как класс шифров, работают в десятки раз быстрее блочных, именно по этому один из них и был включен в программу (блочные шифры относительно долго шифруют большие файлы). При шифровании файла алгоритмом ARC4, нельзя использовать один ключ шифрования дважды, так как использование этого шифра исключает использование вектора инициализации (это не блочный шифр)!

Так же, в пакете «PlexusTCL Crypter», начиная с версии 2.73, присутствует бонус, а именно утилита sha256sum.exe, которая вычисляет SHA-2-256 контрольную сумму как текстов так и файлов размером до 2 Гб включительно. Утилита, как и ее исходный код, распространяется свободно и бесплатно, а в архиве она присутствует на случай, если у пользователя нет программы для вычисления контрольных сумм строк и файлов. Программа принимает на вход три аргумента, а вычисление контрольной суммы строки «PlexusTCL» и ее печать в табличном виде, будет выглядеть так:

```
[user@machine]~$:./sha256sum -t -s «PlexusTCL»

2D 92 4A CF 99 37 74 AC 55 3D C7 A7 6C AD 3D DD
64 4D 93 91 E3 24 58 24 C1 21 FD 66 EE F8 0F EC
```

Утилита sha256sum принимает на вход три аргумента, а именно «-s/t», «-s/f» и простую строку. Строковые эквиваленты первых двух аргументов выглядят как «--string/table» и «--string/file». Первый аргумент позволяет выбрать, в каком виде вы хотите получить контрольную сумму строки или файла, в строковом или табличном. Аргумент «-s/--string» указывает на то, что контрольная сумма будет напечатана в виде строки, а аргумент «-t/--table» на печать контрольной суммы в виде таблицы, как в примере выше. Второй аргумент позволяет явно указать, контрольную сумму чего вычислить, введенной в виде третьего аргумента строки, или файла, именем которого и является третий аргумент. Чтобы вычислить контрольную сумму файла, нужно использовать аргумент «-f/--file», а для вычисления контрольной суммы строки, аргумент «-s/--string». В качестве исходного кода самого алгоритма SHA-2-256 взята реализация от USA NIST (Национальный Институт Стандартов и Технологий США), которая была протестирована с использованием официальных тестовых векторов и полностью безопасна в использовании. Алгоритм был многократно проверен множеством

криптоаналитиков со всего мира, и в нем не было найдено ни уязвимостей, ни лазеек. Сам алгоритм SHA-2-256 был разработан и запатентован USA NSA (Агентство Национальной Безопасности США), что ограничивает его использование, но не запрещает использовать его для домашних целей.

## CryCon

**Cry**pter for **Con**sole – консольная программа-фильтр, аналог графической программы «PlexusTCL Crypter», из которого удален криптостойкий генератор паролей и уничтожитель обрабатываемого файла. Все написанное ранее про программу «PlexusTCL Crypter» и ее алгоритмы, кроме интеграции в IV координат курсора мыши, справедливо и для программы CryCon. Как и любая консольная программа, CryCon принимает агрументы, которые интерпретируются программой, и в зависимости от них, программа выполняет какие-либо операции.

Первым аргументом программы CryCon всегда выступает строка, указывающая на используемый алгоритм шифрования, если этот аргумент не «-h» или «--help», который указывает на то, что нужно вывести короткую справку. Пользователь сам указывает, какой алгоритм шифрования следует использовать для шифрования или расшифровки файла в виде короткого (буквенного) или длинного (строкового) аргумента. Аргументы, соответствующие алгоритмам шифрования, указаны в таблице ниже, и могут быть переданы программе только в маленьком (строковом) регистре.

| алгоритм шифрования | буквенный аргумент | строковый аргумент |
|---|---|---|
| ARC4 | -a | --arc4 |
| AES (Rijndael) | -r | --aes |
| Serpent | -s | --serpent |
| Blowfish | -b | --blowfish |
| Threefish | -t | --threefish |

Второй аргумент, как и последующие, интерпретируются программой в зависимости от выбранного алгоритма. Например, вторым аргументом при выборе алгоритма ARC4, является имя обрабатываемого файла, третьим имя файла назначения и четвертым имя ключевого файла или строковый ключ. Так как при выборе алгоритма ARC4, аргументы указывающие на то, какую операцию выполнять и какой длины ключ использовать, не указываются, из этого следует, что для шифрования файла secret.dat в файл en.secret.dat с использованием в качестве ключа, данные из файла key.sk, правильные аргументы при запуске программы CryCon будут выглядеть так:

**[user@machine]~$:./crycon --arc4 secret.dat en.secret.dat key.sk**

Второй аргумент при выборе любого другого алгоритма, а именно AES, Serpent, Blowfish или Threefish, всегда указывает на то, какую

операцию выполнить, шифрование или расшифровку. Этот аргумент может быть коротким (буквенным) или длинным (строковым), и выглядит буквенный аргумент как «-e» и «-d», а строковый как «--encrypt» и «--decrypt».

Третий аргумент интерпретируется в зависимости от того, какой алгоритм был выбран. Если был выбран алгоритм AES или Serpent, то третий аргумент интерпретируется как указание на то, какой длины ключ следует использовать, 128, 192 или 256 битный. Этот аргумент может быть коротким (буквенным), а именно «-a», «-b», «-c», или длинным (строковым), а именно «--128», «--192» или «--256». Если был выбран алгоритм Blowfish или Threefish, то третий аргумент интерпретируется как имя обрабатываемого файла, потому что при использовании этих алгоритмов длина ключа не указывается. Все дело в том, что в программе CryCon, алгоритм Threefish реализован только в его 512-битной (средней) версии, а алгоритм Blowfish, это алгоритм с переменной длиной ключа, которая всегда максимальна. Из этого следует, что аргументы для запуска программы, при выборе алгоритма AES или Serpent при 256-битном ключе, будут такими:

**[user@machine]~$:./crycon --aes --encrypt --256 secret.dat en.secret.dat key.sk**

**[user@machine]~$:./crycon --serpent --encrypt --256 secret.dat en.secret.dat key.sk**

а при выборе алгоритма Blowfish или Threefish такими:

**[user@machine]~$:./crycon --blowfish --encrypt secret.dat en.secret.dat key.sk**

**[user@machine]~$:./crycon --threefish --encrypt secret.dat en.secret.dat key.sk**

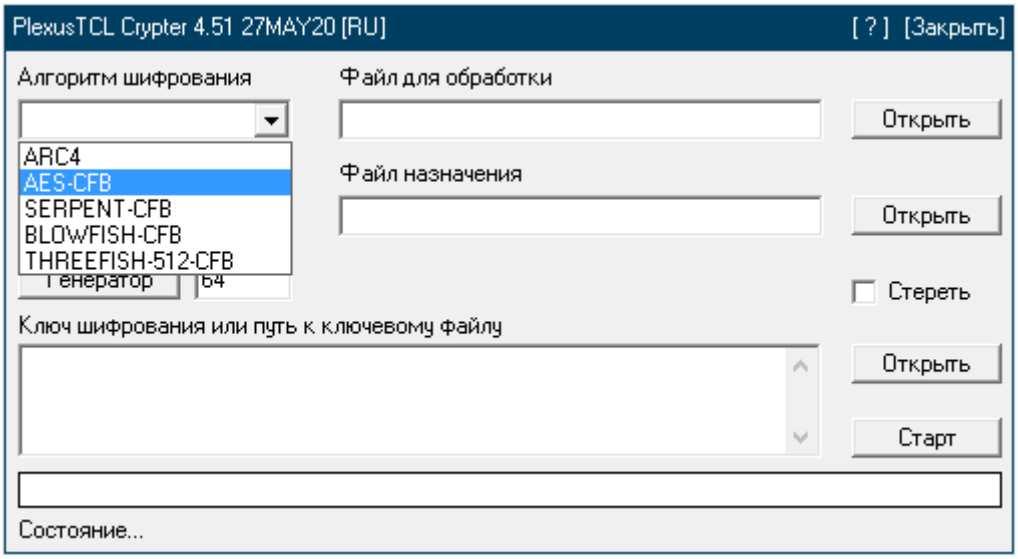
В случае, если во время работы программы произошла ошибка, например такая как: закончилось место на диске, файл для обработки не был открыт (значит что-то мешает), введенный аргумент некорректен (написан неправильно), длина ключа в ключевом файле мала и т.д, то программа уведомит об этом выводом текстового сообщения и прервет все операции.

## Графический интерфейс пользователя

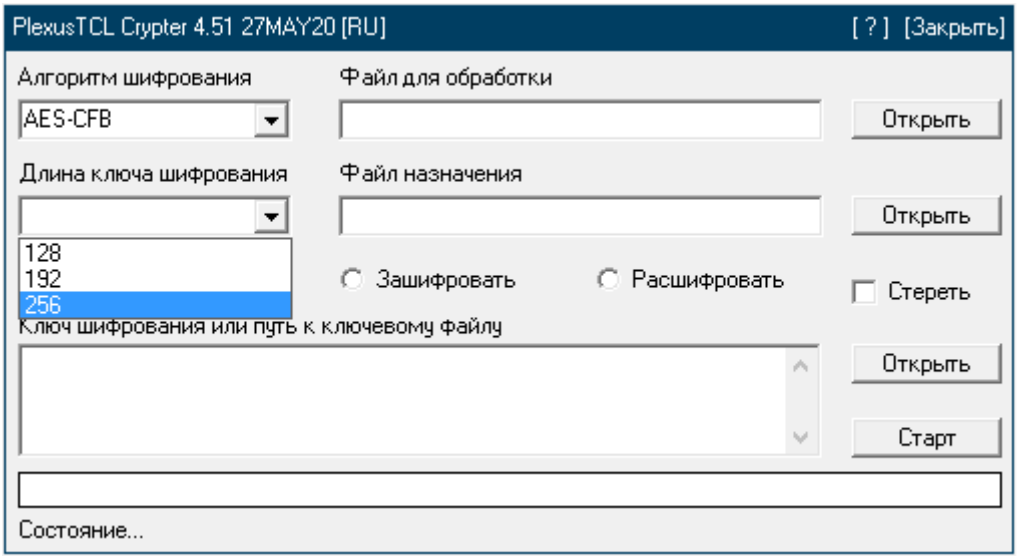
GUI существует, чтобы управлять логикой работы программы с помощью компьютерной мыши, так как не все пользователи успешно работают в командной строке. К тому же, GUI бывает красив, элегантен, строг, интуитивно понятен и просто приятен в работе с ним. Чтобы использовать программу «PlexusTCL Crypter» для обработки файла, нужно выполнить следующие действия.

1.) Запустите программу PlexusTCL Crypter 4.51.exe

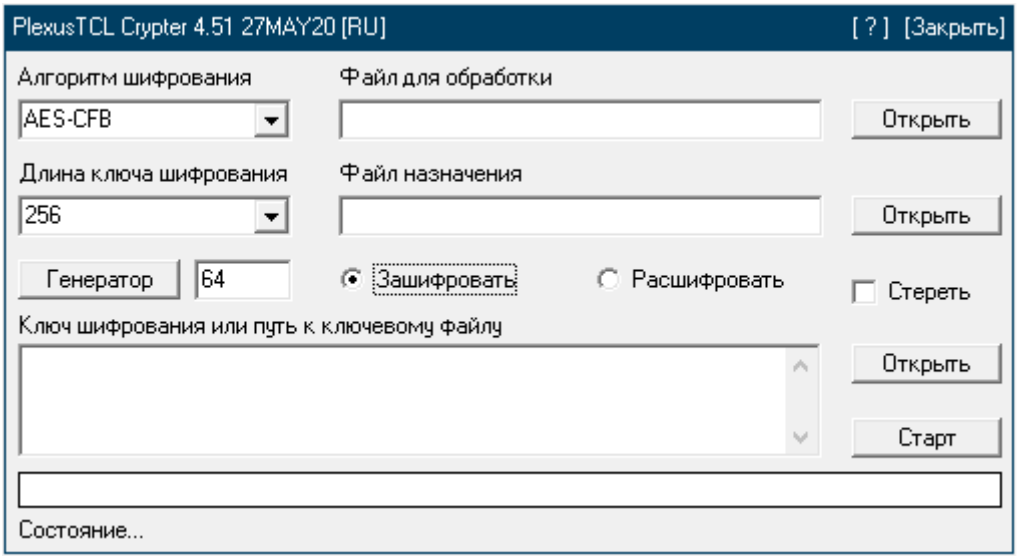
2.) Выберите нужный вам алгоритм шифрования.



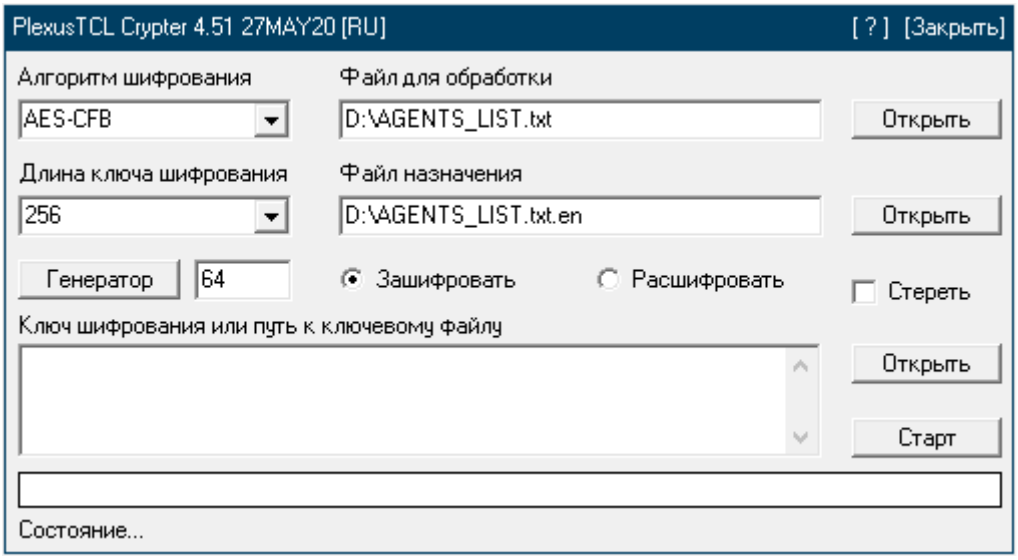

3.) Если появилось поле для выбора длины ключа шифрования, выберете нужную вам длину ключа. Чем больше выбрана длина ключа шифрования, тем больше криптостойкость, но шифрование будет длится дольше.



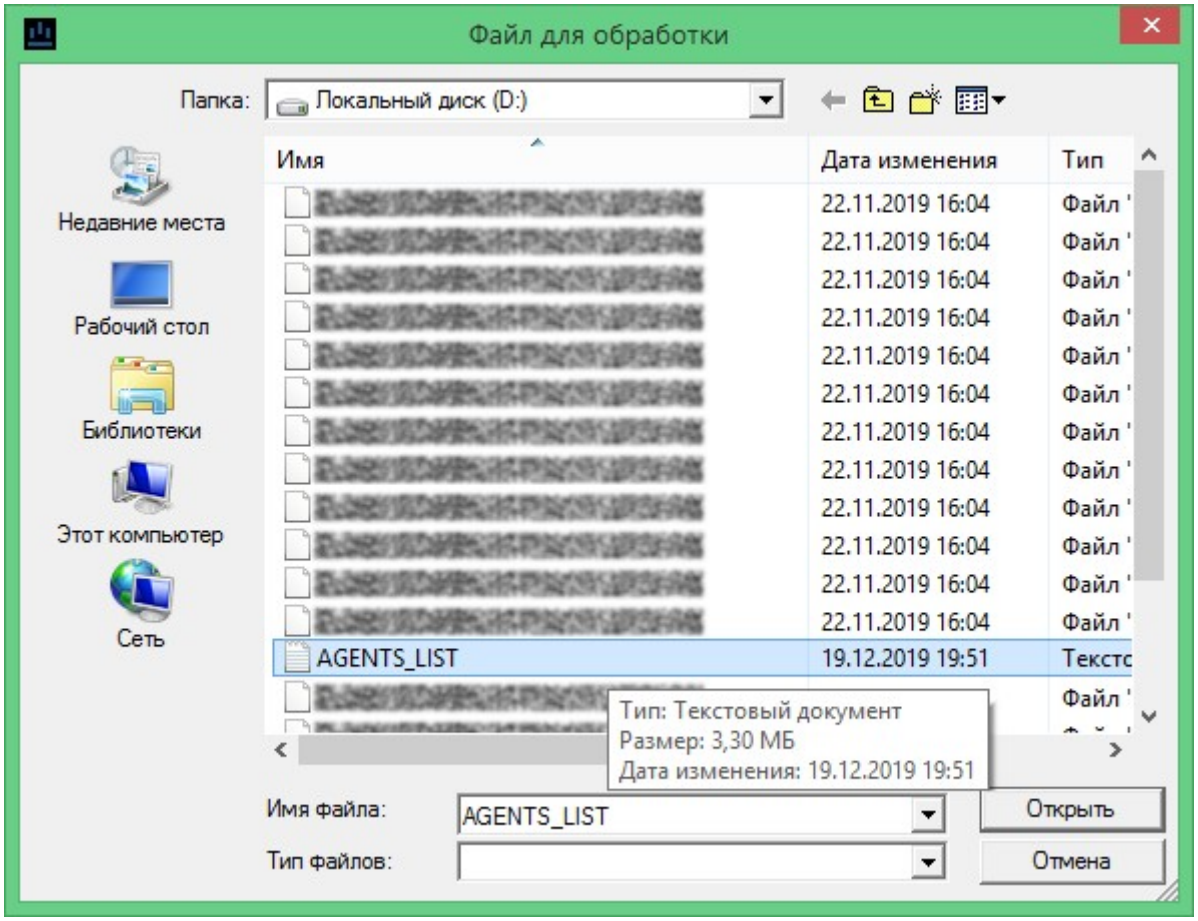

4.) Выберите необходимое действие (зашифровать/расшифровать). Если вы используете поточный шифр ARC4, то поле для выбора длины ключа шифрования и варианты (зашифровать/расшифровать) не появятся.




5.) Введите в поля озаглавленные как «Файл для обработки» и «Файл назначения» названия файлов, который хотите обработать и в который будут записаны обработанные данные. Строки в полях не должны совпадать!

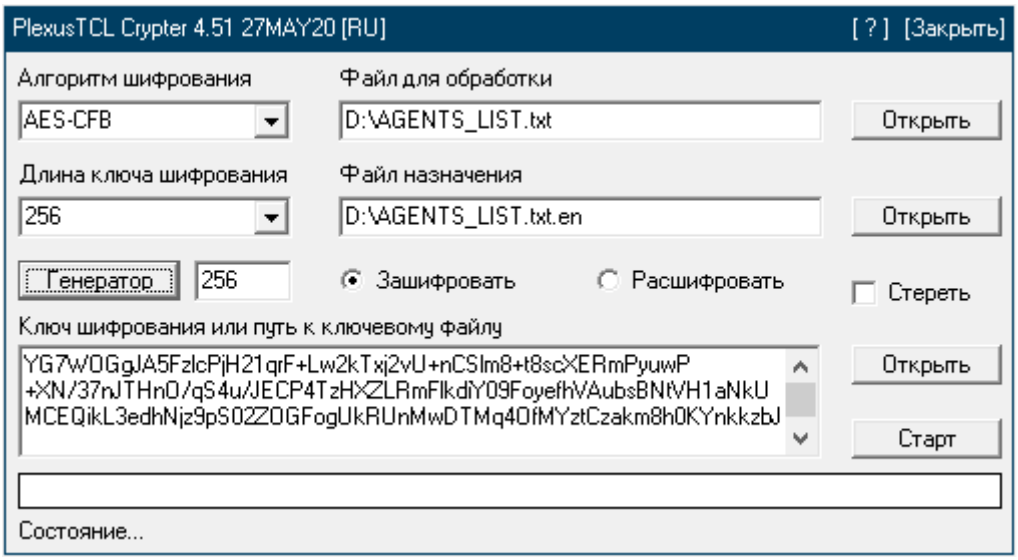



Нужно вводить название файла с его расширением если обрабатываемый файл лежит в одном каталоге с программой, которая будет с ним работать, так как программа не может определять расширения файлов с которыми работает. Если файл лежит в другом каталоге, нужно вводить название файла вместе с полным путем к нему. Если вы не желаете вводить имя файла с полным путем к нему, то вы можете просто вызвать диалоговое окно выбора файла, нажав на кнопку с надписью «Открыть» справа от соответствующего поля.

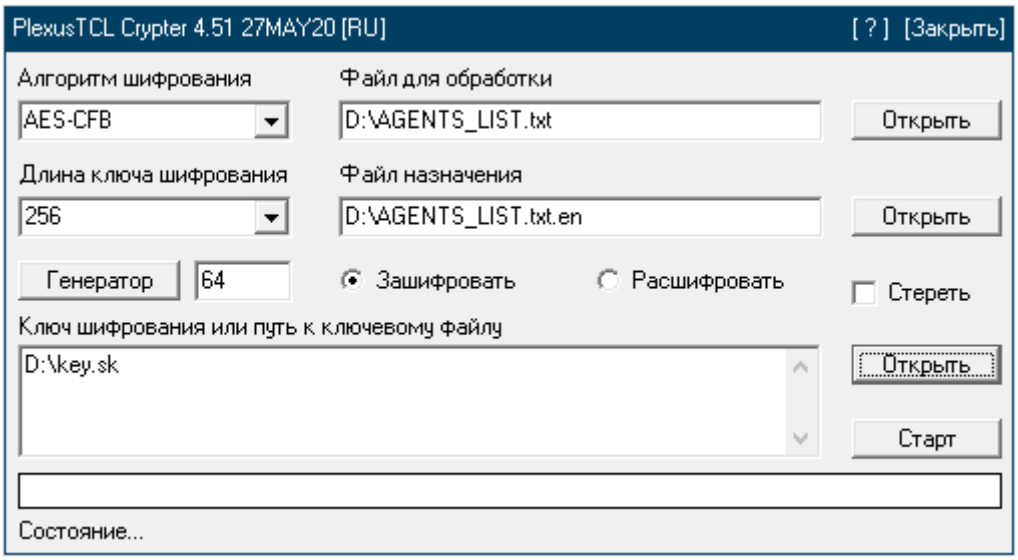



Выбранным для обработки файлом, считается тот файл, название которого появилось в поле «Имя файла» и который одновременно стал выделенным. Чтобы утвердить файл для обработки, нажмите «Открыть», и его имя вместе с полным путем к нему, появится в соответствующем поле. Имя файла, в который будут сохранены обработанные данные, вводится в поле «Файл назначения» или выбирается так же, как и файл для обработки, после нажатия на кнопку «Открыть» справа от соответствующего поля. Каждая кнопка соответствует полю, находящемуся слева от нее.

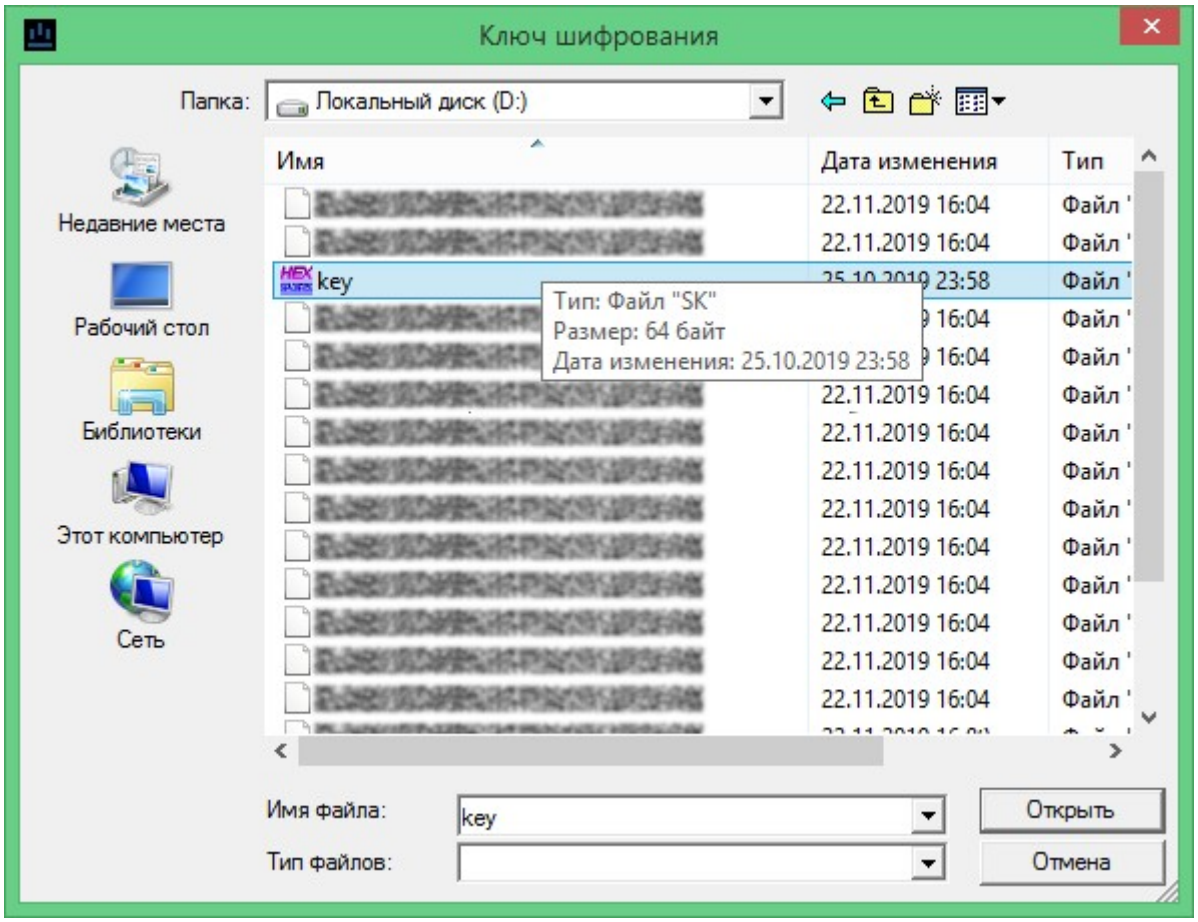
6.) Если вы хотите использовать в качестве ключа шифрования простую строку или ключевую фразу, введите ее в поле озаглавленное как «Ключ шифрования или путь к ключевому файлу». Не стоит беспокоится на счет безопасности использования строки в качестве ключа шифрования, так как строка не используется в качестве ключа шифрования. Строка будет преобразована в ключ шифрования функцией формирования ключа на основе алгоритма хеширования SHA-2-256. В примере ниже, в качестве ключа шифрования, используется строка состоящая из 256 псевдослучайных заглавных, строчных латинских букв, арабских цифр и специальных символов, полученная с помощью встроенного генератора, который может генерировать строки от 8 до 256 символов.




Если вы хотите использовать в качестве ключа шифрования, данные из файла (файл может быть любым, но его размер должен быть больше длины ключа шифрования в байтах или равен ей), введите в поле название файла.

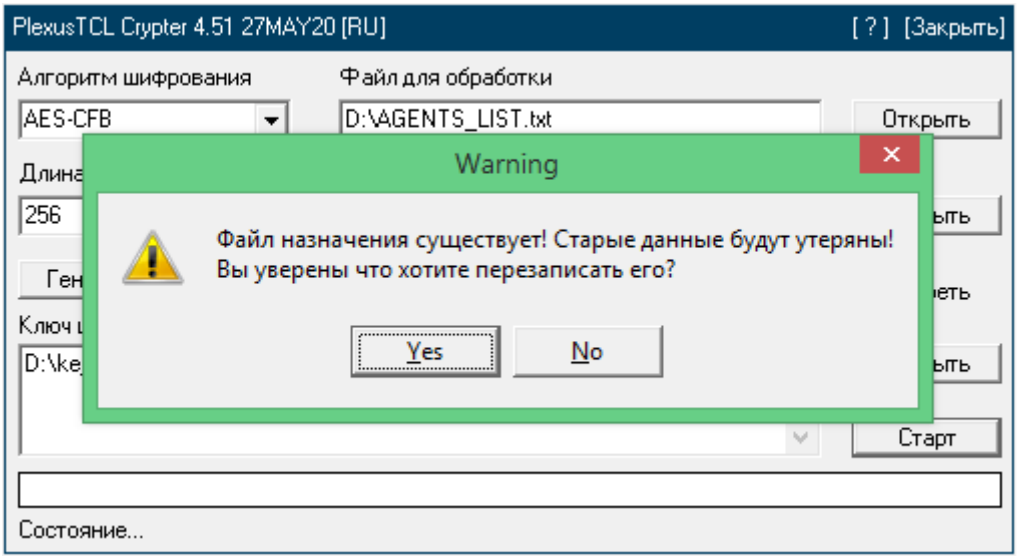



Вы так же можете вызвать диалоговое окно выбора ключевого файла (как на рисунке ниже), если хотите выбрать один из множества файлов, нажав на «Открыть» рядом с полем для ввода ключа. Файл считается утвержденным так-же, как в примере с выбором обрабатываемого файла.



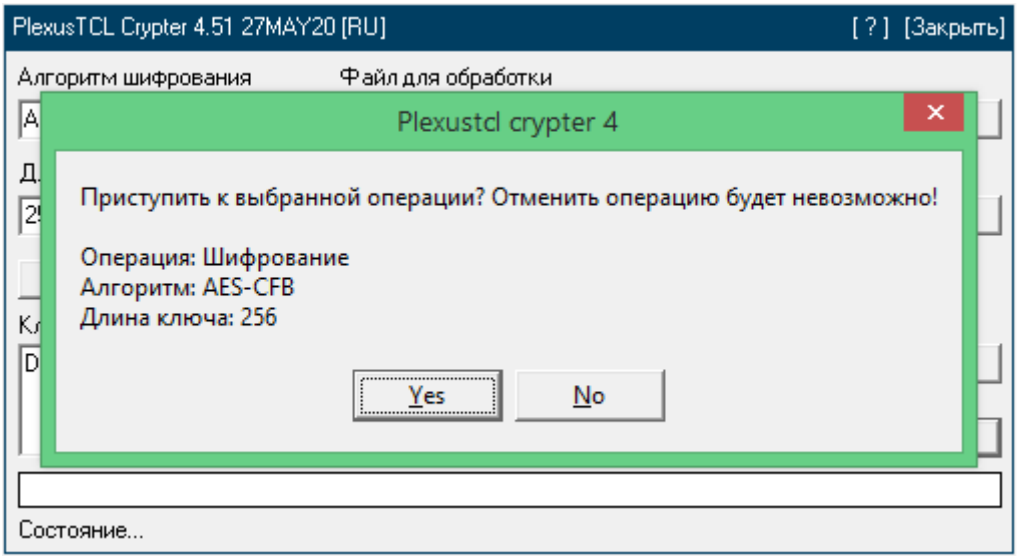

7.) Нажмите «Старт», чтобы начать операцию шифрования/расшифровки. Перед этим, вы можете поставить галочку в поле «Стереть», если желаете уничтожить файл для обработки после того, как он будет обработан. Если галочка установлена, то после окончания всех операций, файл для обработки будет перезаписан нулями, его размер будет усечен до нуля и он будет удален стандартной функцией DeleteFile.

В случае, если вы допустили ошибку, программа уведомит об этом выводом текстового сообщения. Так же, программа уведомит о том, что в качестве файла назначения был выбран существующий файл, и предложит перезаписать его.

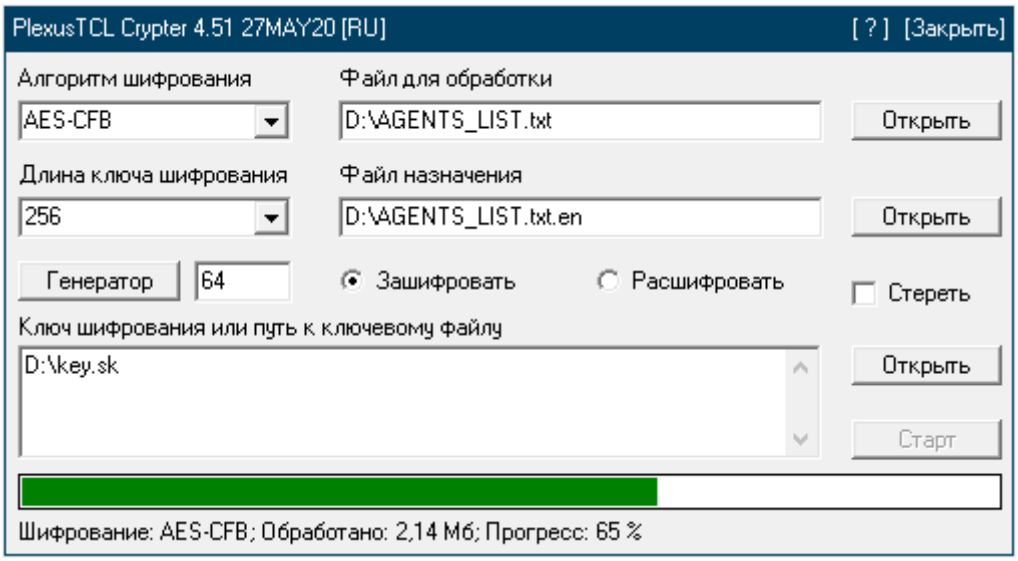



Если вы согласитесь на перезапись существующего файла, программа немедленно уничтожит все данные в файле назначения путем записи обработанных данных из файла для обработки, поверх старых данных. В случае совпадения строк в любых двух и более полях, а именно в полях «Файл для обработки», «Файл назначения» и «Ключ шифрования или путь к ключевому файлу», программа прервет операцию и уведомит о равенстве строк, потому что оно недопустимо.

Если ошибки не были допущены и программа готова к обработке файла, программа уведомит об этом выводом диалогового окна. Обратите внимание на то, что **отмена выбираемой операции невозможна** (придется ждать окончания обработки файла).




Выполняемая операция, используемый алгоритм, количество обработанных данных и процент прогресса, будут видны в самом нижнем поле, как на рисунке ниже.




Когда операция будет завершена, программа уведомит об этом выводом сообщения. Обратите внимание на то, что на время выполнения операции, программа отключает кнопку «Старт». Это сделано для того, чтобы у пользователя не было возможности запустить обработку одного файла сразу в двух потоках одновременно.

8.) Чтобы расшифровать файл, заполните поля так же как и для шифрования, но выберите «Расшифровать» перед нажатием кнопки «Старт».

## Исходные коды

Программа crycon, как и утилита sha256sum, написаны на языке программирования C (Си) и скомпилированы в исполняемые файлы компилятором TCC (Tiny C Compiler) версии 0.9.27. GUI написан на языках программирования C/C++/Pascal (Си, Си++ и Паскаль) и скомпилирован в исполняемый файл компилятором C++ Builder версии 6.0. Все исполняемые файлы в пакете «PlexusTCL Crypter» и их исходные коды, распространяются свободно и бесплатно.

Ниже указаны SHA-2-256 контрольные суммы всех трех программ входящих в «PlexusTCL Crypter»:

| **Название файла** | **SHA-2-256 контрольная сумма** |
|---|---|
| PlexusTCL Crypter 4.51.exe | 69C6F5C109A6BAFDCDD42F2B3166AC85<br>E02C8002E7347DF4F34E1E6CE0F640FE |
| crycon.exe | 51E2ABBE9388B4DA2C1D3ADA8BCC9DBE<br>7F4D84FA684FEE6D1158CDD0012F271F |
| sha256sum.exe | 83CD0D3A42E11DE25D1C5EEEDBE00F60<br>4B2F59FAD4EDF3378E6125DB99BE028A |