Статья Делаем любой симметричный алгоритм шифрования асимметричным

Интро

Приветствую вас пользователи дамаги, я решил поучавствовать во втором конкурсе статей и описать тут идею которая зарадилась у меня ещё давно, эта идея позволит решить проблему оффлайн криптолокеров, в этой статье мы разберём способ шифрования информации любым симметричным алгоритмом в ассиметричном виде, а так-же разберём способ реализации этого и в конце я немного расскажу про подводные камни связанные с написанием криптолокеров и подобного софта.

Проблема оффлайн криптолокеров в том, что им приходится использовать RSA как ассиметричный алгоритм, а он как известно работает медленнее рса и вообще он уже очень заезженный. в этой статье я как раз буду описывать способ решения данной проблемы используя связку из двух алгоритмов шифрования. Примеры кода так-же будут в статье.

Началось всё с того, что я начал изучать криптографию на основе эллептических кривых, в процессе изучения я перечитал
кипу статей, таки смог начать понимать как это всё устроенно и придумал очень хороший на мой взгляд способ быстро и надёжно
шифровать информацию любым симметричным алгоритмом! Алгоритм Диффи Хелльмена и Еллептические кривые (или **curve25519**) можно применить не совсем так, как задумывали их авторы. Об этом собственно и будет статья.

Поговорим пока про шифрование. На мой взгляд в криптолокерах шифрование должно быть минимальным но надёжным, от минимализма зависит скорость. Если делать локер со связками разных шифров, то шифровать такой локер будет до выхода новой винды, лучшая связка шифров на мой взгляд это: **Salsa20 + AES + curve25519** алгоритм стоит выбирать не слишком громоздкий, про вес софта забывать тоже не стоит. Не стоит использовать всякие vector и тп в шифровании как это любят делать некоторые локеры, это создаёт упадок в скорости, лучше вообще искать или писать алгоритм шифрования на си, там обычно максимум нужно ключ в struct пихать от чего легко можно избавиться.

Немного про подводные камни. Если вы начнёте использовать **ECC (Elliptic curve cryptography)**, то у вас будет упадок в скорости по сравнению с **curve25519** (улучшенная версия ECC, там быстрее работает алгоритм Диффи Хелльмена). Ещё не пытайтесь сам это реализовывать, там чёрт ногу сломит.

В конце статьи вы поймёте почему можно использовать любой симметричный алгоритм шифрования, но стоит выбирать криптоустойчевые алгоритмы, так как всякие ксоры легко взламываются.

Надеюсь статья окажется для вас полезной, всех тех кому она понравистя больше остальных прошу проголосовать за мою статью на конкурсе
статей номер 2 =)

Теория
Начнём с терминов

Шифрование: изменение информации с целью скрытия сути сообщения от всех кроме
получателя. Это всякие алгоритмы подписи, хеширования, шифрования и тп...
Симметричное шифрование: шифрование информации определённым ключем, такие алгоритмы
принимают на вход информацию и ключ, а отдают зашифрованную информацию этим ключём.

Ассиметричное шифрование: шифрование на основе публичного и приватного ключа,
публичным ключём можно зашифровать информацию, приватным расшифровать. Имея публичный ключ
расшифровать информацию 'нормальными' способа не получится.

Алгоритм Диффи Хелльмена: алгоритм который позволяет создать 1 ключ на основе твоего приватного ключа и
публичного ключа собеседника, если он сделает это со своим приватным ключём и моим публичным то получится тот-же
самый ключ.

Эллептические кривые: аналог RSA, алгоритм для подписи сообщений, с помощью приватного ключа можно подписать сообщения, а с помощью публичного проверить подпись.
Это можно использовать для проверки отправителя, но мы будем использовать их для шифрования.

Что нужно для шифрования используя эту идею:
1) Эллептические кривые
2) Алгоритм Диффи Хелльмена
3) Любой симметричный алгоритм шифрования

Пример
Пример использования алгоритма Диффи Хелльмена:
Алиса и Кирито создали пару ключей:
Публичный ключ алисы: APUBLIC
Приватный ключ алисы: APRIVATE

Публичный ключ Кирито: KPUBLIC
Приватный ключ Кирито: KPRIVATE

APUBLIC + KPRIVATE = FEDEADXXXXXXXX
APRIVATE + KPUBLIC = FEDEADXXXXXXXX

Этот алгоритм хотели использовать с целью защищённой передачи информации через интернет. Если кто-то перехватит публичные ключи, то ничего сделать с ними он не сможет.

Но у меня возникла другая мысль. А что если хранить в коде публичный ключ и при каждом шифровании информации
генерировать новую пару ключей, на основе сохранённого публичного ключа и сгенерированного ключа создать ключ используя
алгоритм Диффи Хелльмена и сохранить сгенерированный публичный ключ в зашифрованной версии сообщения. С помощью
этого способо можно использовать любой симметричный алгоритм шифрования в ассиметричном виде!

Практика
Вот примерный на c++
C++:

```
//функция принимает буффер, длинну буффера и публичный ключ
byte* encrypt(byte* buffer, int buflen, byte* pubkey){
    byte* local_priv_key, local_pub_key, secret;
    //генерируем два ключа
    ECCGenKeys(local_priv_key, local_pub_key);
    //создаём сессионный ключ на основе вшитого публичного ключа и сгенерированного
приватного
    ECCGenSessionKey(secret, pubkey, local_private_key);

    byte* temp_buffer = (byte*)alloc(buflen);
    byte* return_buffer = (byte*)alloc(buflen + 32);
    copy(temp_buffer, buflen, buffer);
    //шифруем буффер с использованием сгенерированного сессионного ключа
    aes_encrypt(temp_buffer, secret);
    copy(return_buffer, 32, local_pub_key);
    copy(return_buffer + 32, buflen, temp_buffer);
    memfree(temp_buffer);

    memfree(local_priv_key);
    memfree(local_pub_key);
    memfree(secret);
    return return_buffer;
}
```

После выполнения этого кода у нас получается буффер, первые 32 байта которого это публичный ключ,
а останое это зашифрованная информация.

Для расшифровки нам нужно использовать сохранённый приватный ключ и публичный ключ (первые 32 байта буффера)
Примерно так будем выглядить код расшифровки

C++:

```
//функция принимает буффер, длинну буффера и приватный ключ
byte* decrypt(byte* buffer, *buflen, byte* prikey){
    byte* secret, byte* data = (byte*)alloc(buflen - 32), byte* pubkey = (byte*)alloc(32);//
выделение памяти...

    copy(data, buffer + 32, buflen - 32);
    copy(pubkey, buffer, 32);

    //восстанавливаем сессионный ключ на основе нашего приватного и публичного что сохранён в
сообщении
    ECCGenSessionKey(secret, pubkey, prikey);
    //расшифровываем сообщение полученным ключём
    aes_decrypt(data, secret);

    memfree(pubkey);
    memfree(secret);
    return data;
}
```

После выполнения этого кода мы получим расшифрованный вариант сообщения.

Получается что можно использовать любой симметричный алгоритм шифрования в ассиметричном виде, хоть **AES, Salsa20, GOST**, да хоть **XOR**

Заключение

В заключении хочу сказать что для криптолокеров лучше использовать такой тип шифрования:
1) создать ключ на файл
2) зашифровать файл
3) зашифровать этот ключ мастер ключём
4) записать зашифрованный ключ в конец файла
5) хранить зашифрованную версию файлового мастер ключа на пк

Это исключает возможность брутфорса мастер ключа и эта система будет быстро отрабатывать. А вот мастер ключ уже можно шифровать способом описанным в этой статье.
Так-же не стоит использовать крипто апи в локерах так как он вызывает детекты.

А по поводу шифрования файлов не стоит делать упор на скорость, стоит шифровать файл полосками, либо шифровать первые 5кб файла, их хватит с головой. Правда если юзер опытный и имеет более старую версию этого бекапа то ничего ему не мешает взять те же первые 5 кб файла и вставить их в зашифрованную версию. Если повезёт то файл будет 'расшифрован'.

Расскажу ещё про свою прошлою ошибку в написании локера, я брал блок в 32 байта хотя нормальным блоком для алгоритма был
блок в 64 байта, по итогу когда софт проверяли его смогли расшифровать, не делайте таких глупых ошибок =)
Так-же не стоит мудрить с алгоритмами шифрования, шифровать файл двумя алгоритмами и тп, это бессмысленно, разве
что только для понтов которые серьёзным людям не нужны, если вы так делаете то максимум кого вы найдёте так это школоту.
На мой взгляд самыми хорошими алгоритмами шифрования для локеров являются: **Salsa20, AES, Lucifer, curve25519**.

И лучше не используйте эллептические кривые, используйте **curve25519**. Алгоритм Диффи Хелльмена отрабатывает там куда быстрее
чем в эллептических кривых. Ещё не реализовывайте криптографию сами, ничем хорошим это не закончится, любая ошибка тут критична.

Вот на этом хочу закончить свою статью, желаю вам успехов в ваших начинаниях в криптографии и разработке ПО, не делайте глупостей и всегда старайтесь делать софт адекватно а не ляпать лишь бы работало, иначе будет очередной высер которых и так уже полно на рынке, уникальности как таковой у вас не будет, а значит и профита серьёзного не будет, единственный нормальный софт из локеров на данный момент это Ривл и мой софт, хотя у него и так дофига моментов которые стоило бы поправить, на этом пожалуй будет конец этой статьи.