# Что планируется сделать

1. Реализовать функции SCrypt и DES на С++
2. Попробовать улучшить обе с использованием вставок машинных кодах

# Обоснование и планируемые результаты

Криптографическая хэш-функция SCrypt является «медленной на железе» (PBKDF). Используется в основном для хеширования паролей и другой очень ценной информации). Она сконструирована так, чтобы хеширование пароля было максимально затруднено. Это сделано с тем, чтобы злоумышленник, нелегальным способом получивший хэш пароля, испытывал максимальные затруднения при подборе коллизии (т. е. какой-либо, пусть и отличной от настоящего пароля, строки с таким же хэшем) с помощью метода грубой силы. А обычные пользователи не чувствуют особых неудобств, т. к. вход в систему – операция не слишком частая, и лишние 100-1000 мс не слишком критичны для них.

Другими словами, функция SCrypt требует больших затрат памяти и времени. Можно уменьшить затраты одного за счет увеличения затрат другого, но алгоритм специально подобран так, чтобы существенного выигрыша это не дало.

С другой стороны, криптографическая функция DES является «быстрой на железе» (и используется для потокового шифрования, в котором скорость шифрования очень важна). Она довольно старая и считается не слишком стойкой, но в данной работе используется в большей степени в иллюстративных целях.

Итак, как указано выше, функция SCrypt должна быть трудной для вычисления, а функция DES – требовать существенно меньших затрат. Отсюда следует предположение, что функция DES должна лучше (или, по крайней мере, не хуже) поддаваться оптимизации. Попробуем их оптимизировать их с использованием вставок на машинных кодах. Возможны следующие варианты:

* Функции оптимизируются примерно в равной степени или SCrypt оптимизируется хуже. Это будет означать, что функция SCrypt выполняет свою задачу: не слишком хорошо оптимизируется таким способом.
* Функция SCrypt оптимизируется лучше. Это будет означать, что функция SCrypt поддается оптимизации лучше, чем функция, которая задумывалась как «быстрая», то есть хорошо поддается атаке методом грубой силы при использовании машинных кодов. В этом случае следует добавить что-то, что позволит затруднить низкоуровневую реализацию.

Почти наверняка реализуется первый вариант. В этом случае будет подтверждено соответствие алгоритма SCrypt своим задачам. Также было бы интересно сравнить выигрыш (благодаря оптимизации) алгоритма SCrypt с выигрышем более старого алгоритма PBKDF2, а также более новых Bcrypt и Argon2.

# Свежие исследования аналогичной тематики

<https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e> – Компилятивная статья 2019 года. Сравниваются SCrypt, Bcrypt, Argon2, PBKDF2. Делается вывод, что Argon2 – наиболее надежный.

# Литература по теме

<https://www.researchgate.net/publication/331431690_Demystifying_Crypto-Mining_Analysis_and_Optimizations_of_Memory-Hard_PoW_Algorithms>

<https://www.tarsnap.com/scrypt/scrypt.pdf>

<https://www.researchgate.net/publication/329384376_Optimizing_a_Password_Hashing_Function_with_Hardware-Accelerated_Symmetric_Encryption>

<https://eprint.iacr.org/2016/989.pdf>

<https://www.itspy.cz/wp-content/uploads/2019/10/32_prace.pdf>