**Security Remote Password Protocol**

Протокол SRP позволяет пользователю идентифицировать себя на сервере, не передавая своего пароля, то есть подтвердить тот факт, что он знает свой пароль, и только этот факт.

Основные параметры, используемые в данном протоколе:

* *q* и *N* = 2*q* + 1 выбираются так, что *N* и *q* простые. N должно быть достаточно большим, чтобы [дискретное логарифмирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BB%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) по модулю *N* было практически неосуществимо. Все вычисления производятся по (*mod N*),
* *g* — генератор мультипликативной группы (генератор по модулю N => для любого 0 < X < N существует и единственный x такой, что g^x % N = X),
* *k* — параметр, получаемый на обеих сторонах, например, *k* = *H*(*N*, *g*) в ревизии 6а (в ревизии 6 *k* было постоянным и равно 3),
* *s* — [соль](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) (также *модификатор*) — [строка](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) данных, которая передаётся [хеш-функции](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) вместе с [паролем](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D0%BB%D1%8C). Используется для удлинения строки пароля, чтобы увеличить сложность [взлома](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%BB%D0%BE%D0%BC_%D0%BF%D0%B0%D1%80%D0%BE%D0%BB%D1%8F),
* *I* — идентификатор пользователя в системе сервера (username),
* *p* — пароль пользователя, соответствующий *I (*password*)*,
* *H*() — [криптографическая односторонняя хеш-функция](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F), например, [SHA-256](https://ru.wikipedia.org/wiki/SHA-256),
* *x* — секретный ключ, *x* = *H*(*s*, *p*),
* *v* — верификатор пароля на стороне сервера, *v* = *g*x. Верификатор по математическим свойствам схож с открытым ключом, так как он легко получается из пароля, а обратная операция является вычислительно неразрешимой,
* *u* — произвольный параметр для кодирования (публичная функция побелки),
* *a*, *b* — секретные одноразовые числа (сессионные приватные ключи),
* *A*, *B* — сессионные публичные ключи.

Сервер хранит пароли, используя следующую формулу:

* *x* = *H*(*s*, *p*) (s выбирается произвольным образом),
* *v* = *g*x (вычисление верификатора пароля),

После этого сервер хранит пару (*I*, *s*, *v*) в своей базе данных.

**Аутентификация происходит по следующей схеме:**

1. Клиент -> Сервер:  *I*, *A* = *g*a (идентифицируется, *a* - произвольное)
2. Сервер -> Клиент:  *s*, *B* = *kv* + *g*b (посылает сохраненное *s*, произвольное *b*)

На обеих сторонах рассчитывается: *u* = *H*(*A*, *B*).

*На стороне клиента:*

1. *x* = *H*(*s*, *p*) (пользователь вводит пароль)
2. *S* = (*B* - *kg*x)(a + ux) (вычисляется ключ сессии)
3. *K* = *H*(*S*) (K - это искомый ключ для шифрования)

*На стороне сервера:*

1. *S* = (*Av*u)b (вычисление ключа сессии)
2. *K* = *H*(*S*) (*K* - это искомый ключ для шифрования)

Теперь обе стороны имеют общий секретный ключ K. Для завершения аутентификации, им необходимо свериться, что их ключи совпадают. Один из возможных способов:

Клиент -> Сервер: *M* = *H*(*H*(*N*) xor *H*(*g*), *H*(*I*), *s*, *A*, *B*, *K*) и проверка на стороне сервера

Сервер -> Клиент: *H*(*A*, *M*, *K*) и проверка на стороне клиента.

Для реализации данного вида аутентификации Security Remote Password Protocol возможно использовать уже имеющуюся в открытом виде библиотеку *"NETSRP"*. Лицензия на исходный код - *"GNU Lesser General Public License".* Располагается по адресу *"https://github.com/cncfanatics/SRP"*. Библиотека представляет собой пять cs-файлов. Два основных из них: *"Client.cs"* и *"Server.cs"* - это классы, которые реализуются на стороне клиента (*Client.cs*) и сервера (*Server.cs*) для выполнения основных операций по вычислению представленных выше значений и выражений по их расчету. *"Constants.cs"* – класс, хранящий константные величины значений N и g, определяемые по умолчанию для случая, когда они не будут заданы. Класс *"General.cs"* содержит методы для расчета секретного ключа *x* = *H*(*s*, *p*), расчета хэш-функции и вычисления ключа сессии S. Также в состав библиотеки входит класс *"BigInteger.cs"*, отвечающий за работу с большими целыми значениями.

**Методика использования библиотеки, для реализации схемы аутентификации**

1. На стороне клиента создается экземпляр класса *User*, через сигнатуру конструктора определяются:

- значения имени пользователя (*username*),

- пароля пользователя (*password*).

Также возможно задать вручную значения *N* и *g*. Если они не будут заданы, то будет выполнено подключение данных параметров, определяемых по умолчанию (*"DefaulValues.cs"*).

В конструкторе класса выполняется расчет величины сессионного публичного ключа A и параметра *k* (хэш-функция от N и g)*.*

2. Серверу отсылается имя пользователя (*username*) и величина сессионного публичного ключа A, которую возвращает метод *StartAuthentication()*.

3. На стороне сервера создается экземпляр класса *Server*. В сигнатуре конструктора данного класса должны быть заданы следующие параметры:

- имя пользователя (*username*),

- соль,

- верификатор пароля на стороне сервера,

- сессионный публичный ключ A,

также возможно задать вручную значения *N* и *g,* но они не обязательны.

В конструкторе класса выполняется расчет значения M, для сравнения со значением, полученным от клиента (метод *Verification)*. Также рассчитывается величина H\_AMK,как хэш-функции от A, M, K для отсылки клиенту в знак подтверждения аутоидентификации. Для расчета этих величин в конструкторе определяются следующие параметры:

- сессионного публичного ключа B

- параметра *k* (хэш-функция от N и g)*,*

*-*  *u* (хэш-функция от A и B),

- ключа сессии S,

- ключа для шифрования *K*,

- окончательного значения M для сравнения с этим же значением, полученным от клиента *user\_M*.

Клиенту отсылаются только значения B и s (соль).

4. На клиентской стороне, на основе полученных значений B и s (соли), через вызов метода *Resulting\_M\_for\_comparision,* производится вычисление значения M. В данном методе результирующее значение M получается посредством последовательного вычисления следующих величин:

- *u* (хэш-функции от A и B),

- секретного ключа *x* = *H*(*s*, *p*),

- верификатор пароля *v* = *g*x*(mod N),*

*-* ключа сессии S,

- ключа для шифрования *K*

- значения M, для отсылки серверной стороне

- значение H\_AMK, хэш – функция от величин A, M, K.

Значение M отсылается серверной стороне.

5. На стороне сервера производится сравнение полученной величины M от клиента и рассчитанной величины M (метод *Verification*). Если данные значения совпадают, то аутентификация произошла успешно и клиенту высылается величина H\_AMK,как хэш-функции от A, M, K. На стороне клиента производиться сравнение полученной величины H\_AMK от сервера и производится сравнение (по длине байтового массива и побайтово) с этой же величиной, полученной клиентом (метод *Verification*). Если все прошло успешно, флаг аутентификации меняется на: *authenticated = true*.

