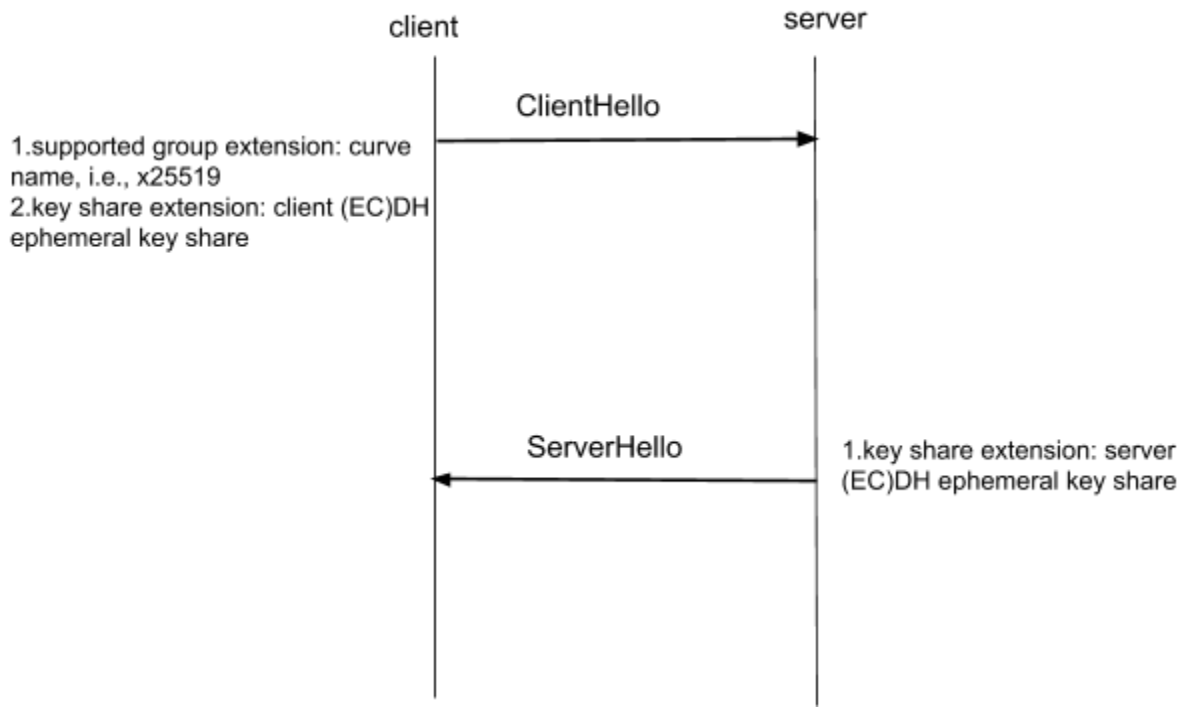
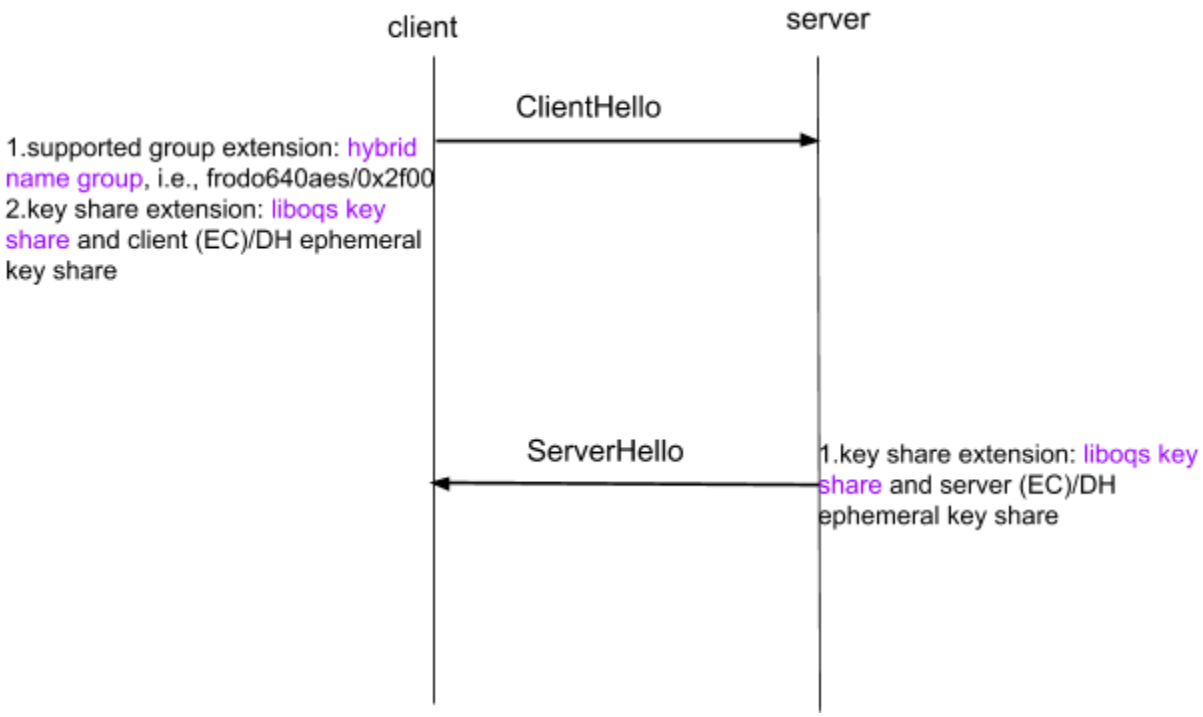
OpenQKD Network liboqs/OpenSSL integration

1 TLS 1.3 key exchange

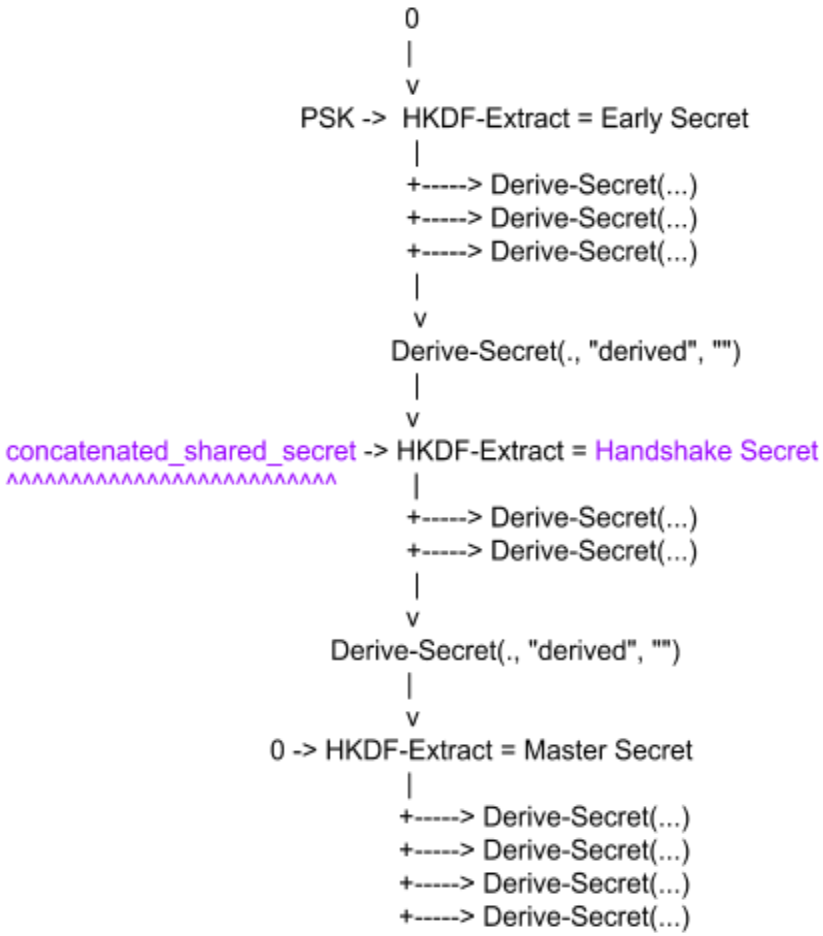


2 TLS 1.3 liboqs hybrid key exchange

2.1 TLS 1.3 liboqs handshake

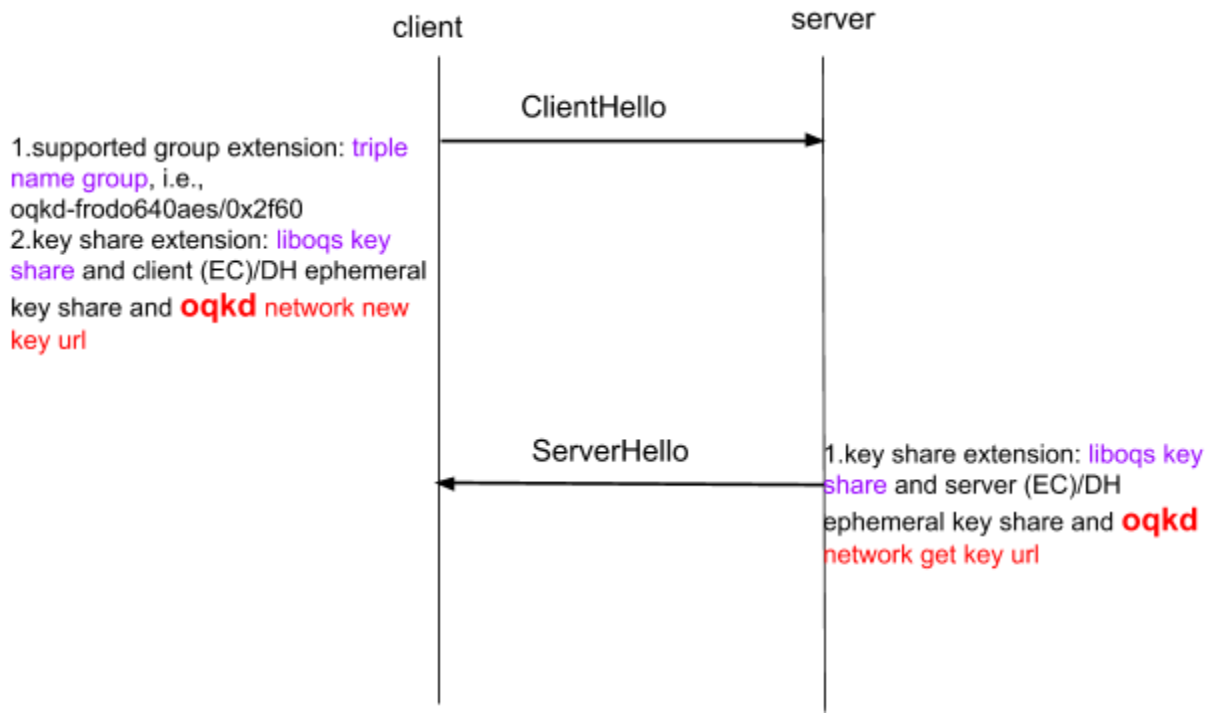


2.2 Liboqs hybrid handshake secret calculation



concatenated_shared_secret = (EC)DH key || liboqs key

3 TLS 1.3 oqkd network + liboqs + (EC)/DH triple key exchange



Please note that the OpenQKD Network key is **NOT** sent in the SSL ClientHello/ServerHello message, instead the OpenQKD Network new_key_url/get_key_url is sent in ClientHello/Server respectively. Server news OpenQKD Network key based on new_key_url from client in ClientHello and returns the get_key_url to client in ServerHello. Client then gets the OpenQKD Network key with get_key_url. Section 6 illustrates the whole process.

concatenated_shared_secret = (EC)DH key || liboqs key || **oqkd network key**

4 New OpenSSL APIs

4.1 Client side

```
void SSL_set_oqkd_new_key_url_callback(SSL *s, int
(*callback)(char**url, int* len))
```

```
void SSL_set_oqkd_get_key_callback(SSL *s, int (*callback)(char*
get_key_url, char** key, int* keylen))
```

4.2 Server side

```
void SSL_set_oqkd_new_key_callback(SSL *s, int (*callback)(char*
new_key_url, char** key, int* keylen, char**get_key_url))
```

5 OpenQKD Network library/libopenqkd

5.1 client side

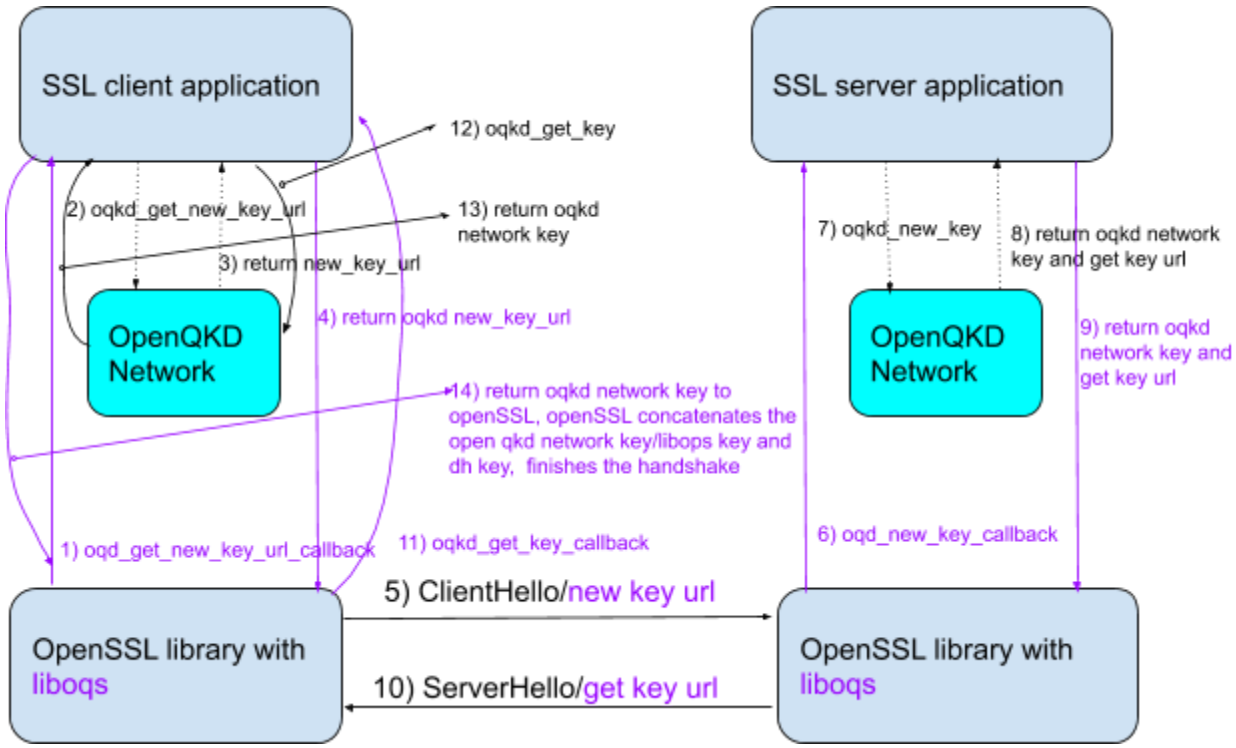
```
int oqkd_get_new_key_url(char** new_key_url);
```

```
int oqkd_get_key(char* get_key_url, char**key, int* key_len);
```

5.2 server side

```
int oqkd_new_key(char* new_key_url, char**key, int* key_len, char**
get_key_url);
```

6 Overall process



Client needs to set liboqs algorithm and OpenQKD Network callbacks to initiate SSL handshake. It creates SSL_CTX first, then calls SSL_CTX_set1_groups_list to set the liboqs algorithm for example p256_oqkd_fordo640aes, and calls SSL_set_oqkd_new_key_url_callback and SSL_set_oqkd_get_key_callback API to set OpenQKD Network callbacks.

Server sets SSL_set_oqkd_new_key_callback after creating SSL object. Then the following steps are executed when the client starts SSL handshake.

Step 1: OpenSSL library on client side invokes *oqkd_new_key_url* callback that is provided in SSL client application.

Step 2: SSL client application calls oqkd_get_new_key_url provided by libopenqkd.

Step 3: OpenQKD Network returns the new key url to SSL client application.

Step 4: oqkd_new_key_url callback in client application returns oqkd_new_key_url to OpenSSL library.

Step 5: OpenSSL library adds the new key url to ClientHello KeyShare extension and sends the ClientHello to the server.

Step 6: Upon receiving the ClientHello, OpenSSL library in server invokes oqkd_new_key_calback with the new key url from the ClientHello keyshare extension.

Step 7: The new key callback provided by server application invokes oqkd_new_key with new key url.

Step 8: OpenQKD Network on server side news/generates the key via OpenQKD Network new key API, and returns oqkd key and get_key url to server application.

Step 9: oqkd_new_key callback in server application returns the oqkd key and get_key url to OpenSSL library.

Step 10: OpenSSL library puts the get_key url into KeyShare extension in ServerHello and calculates the shared secret by concatenating (EC) DH key/liboqs key/openQKD Network key.

Step 11:Upon receiving ServerHello, client side OpenSSL library extracts the get_key_url and invokes oqkd_get_key callback with get_key_url.

Step 12: oqkd_get_key callback provided by client application calls qkd_get_key with get_key_url.

Step 13: OpenQKD Network on the client side returns the OpenQKD Network key to the client application via OpenQKD Network get key API.

Step 14: oqkd_get_key callback provided by client application returns the OpenQKD Network key to OpenSSL library. OpenSSL library calculates the shared secret by concatenating (EC) DH key/liboqs key/OpenQKD Network key, continues and finishes the SSL handshake process with SSL server.

7 SSL application change

- Client calls *SSL_CTX_set1_groups_list/SSL_set1_groups_list* to set the algorithm, for example *p256_oqkd_frodo640aes*. Please note that SSL_CTX_set1_groups_list/SSL_set1_groups_list are existing SSL API.
- Client sets *oqkd_new_key_url_callback* where *oqkd_get_key_url* is called
- Client sets *oqkd_get_key_callback* where *oqkd_get_key* is called
- Server sets *oqkd_new_key_callback* where *oqkd_new_key* is called
- Application links with ***libopenqkd*** library and *libcurl/libjson-c*.

8 Sample applications

8.1 updated openssl s_client and s_server

*./apps/openssl s_client -groups **p256_oqkd_frodo640aes** -CAfile ~/openquantumsafe/openssl/ecdsa_CA.crt --connect 192.168.2.235:4443*

./apps/openssl s_server -cert ~/openquantumsafe/openssl/ec_srv.crt -key ~/openquantumsafe/openssl/ec_srv.key -tls1_3 -accept 4443

8.2 s_client change

```
/*OQKD*/
SSL_set_oqkd_new_key_url_callback(con, oqkd_new_key_url_callback);
SSL_set_oqkd_get_key_callback(con, oqkd_get_key_callback);
```

8.3 s_server change

```
/*OQKD*/
SSL_set_oqkd_new_key_callback(con, oqkd_new_key_callback);
```

8.4 common callbacks

```
int oqkd_new_key_url_callback(char** url, int *len) {
   if (oqkd_get_new_key_url(url) == 0) {
       printf("oqkd_new_key_url is:%s\n", *url);
       *len = strlen(*url);
       return 0;
   } else {
       printf("oqkd_new_key_url fails!\n");
       return -1;
   }
}

/*new_key_url is zero terminated, get_key_url is also zero terminated, key
is NOT zero terminated*/
int oqkd_new_key_callback(char* new_key_url, char** key, int *key_len,
char** get_key_url) {
   // call openQKD to get new key with new_key_url
   if (oqkd_new_key(new_key_url, key, key_len, get_key_url) == 0) {
       printf("oqkd_new_key succeeds, key_len:%d, get_key_url:%s\n",
*key_len, *get_key_url);
       return 0;
   } else {
       printf("oqkd_new_key fails!\n");
       return -1;
   }
}

/*get_key_url is zero terminated*/
int oqkd_get_key_callback(char* get_key_url, char** key, int *key_len) {
  if (oqkd_get_key(get_key_url, key, key_len) == 0) {
      printf("oqkd_get_key succeeds, key_len:%d\n", *key_len);
      return 0;
  } else {
      printf("oqkd_get_key fails!\n");
      return -1;
  }
}
```

9 Post-quantum algorithms

https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

NIST post-quantum cryptography algorithms round 3 finals

Algorithm name	NID
p256_oqkd_kyber512	0X2F70
p384_oqkd_kyber768	0X2F71
p521_oqkd_kyber1024	0X2F72
p256_oqkd_kyber90s512	0X2F73
p384_oqkd_kyber90s768	0X2F74
p521_oqkd_kyber90s1024	0X2F75
p256_oqkd_ntru_hps2048509	0X2F76
p384_oqkd_ntru_hps2048677	0X2F77
p521_oqkd_ntru_hps4096821	0X2F78
p384_oqkd_ntru_hrss701	0X2F79
p256_oqkd_lightsaber	0X2F7A
p384_oqkd_saber	0X2F7B
p521_oqkd_firesaber	0X2F7C

10 Build and run

```
mkdir qkd-test
cd qkd-test
git clone https://github.com/Open-QKD-Network/openssl.git
cd openssl
git checkout openqkd
cd ../
git clone https://github.com/Open-QKD-Network/liboqs.git
cd liboqs
git checkout openqkd
mkdir build
cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX=../../openssl/oqs ..
ninja
ninja install
cd ../../openssl
# Build openssl with debug (you can use gdb to debug openssl)
./Configure no-shared linux-x86_64 -lm no-asm -g3 -O0
-fno-omit-frame-pointer -fno-inline-functions
# Build openssl without debug
./Configure no-shared linux-x86_64 -lm
make

# Start openqkd network
# checkout the code from disable-spring-auth
https://github.com/Open-QKD-Network/qkd-net/tree/disable-spring-auth
# and disable authentication by adding false in last line of kms.conf

more ~/.qkd/kms.conf
http://localhost:9992/uaa/oauth/token
http://localhost:8095/api/newkey
http://localhost:8095/api/getkey
B
false

#Assume server IP address is 192.168.2.235
#On server side
./apps/openssl s_server -cert ./certs/ec_srv.crt -key
./certs/ec_srv.key -tls1_3 -accept 4443
# On client side
./apps/openssl s_client -groups p384_oqkd_saber -CAfile
./certs/ecdsa_CA.crt --connect 192.168.2.235:4443
```