Build QwyitTalk™

Suggested QwyitTalk™ Prototype Development Plan

1. Identify an appropriate client/server area for demonstration, including the following properties
    a. Best fit development platform
    b. Best fit open source code, with easy integration of QT™ code and libraries
    c. Best fit benchmarking against *normal* operation (not any 'secure' versions, as these may/will have special implementation guidelines: compare QT™ to insecure)
    d. Best fit optimization so as to accomplish claimed benchmarks (QT™ should add only a small overhead without any hardware or software modifications to insecure operation)
2. Update Server (QDS), with the following capabilities
    a. Listen on port 4180 (HTTPX)
        i. Perform VSU - Web (HTTP and HTTPS), email and SMS capability
    b. Perform AH
    c. Key storage/update (DB, file, etc.)
    d. A best-of-breed random number generator (NIST approved recommended)
3. Update Client, with the following capabilities
    a. Listen on port 4180 (HTTPX)
        i. Perform VSU - Web (HTTP and HTTPS), email and SMS capability
    b. Perform AH
    c. Key storage/update (DB, file, etc.)
    d. Local version (OS embedded) best-of-breed random number generator (NIST nice)
4. Build installs
    a. Client includes building/creating an SDK that others could use as a guide for insertion into other products, as well as building their own client versions of the same demo product (to see for themselves how easy it is to use/build, as well as independent benchmarks)
    b. The Server build will stay 'proprietary' (even though well documented), as it is envisioned that the server is licensed for others to operate their own independent QDSs, as well as federate the trust of our central QDS to anyone who wants to connect
        i. This means that a follow-on activity (phase) to a successful demo is to define (reference guide like this one), then build an SDK that connects QDSs together (how they message for client searches, do a more-robust version of the VSU (more factors of authentication as mentioned in this VSU guide), etc.
5. Test
6. Benchmark

There are libraries available for the fundamental QT™ functions

For the complete technical QT™ implementation details, see *The QwyitTalk™ Reference Guide*

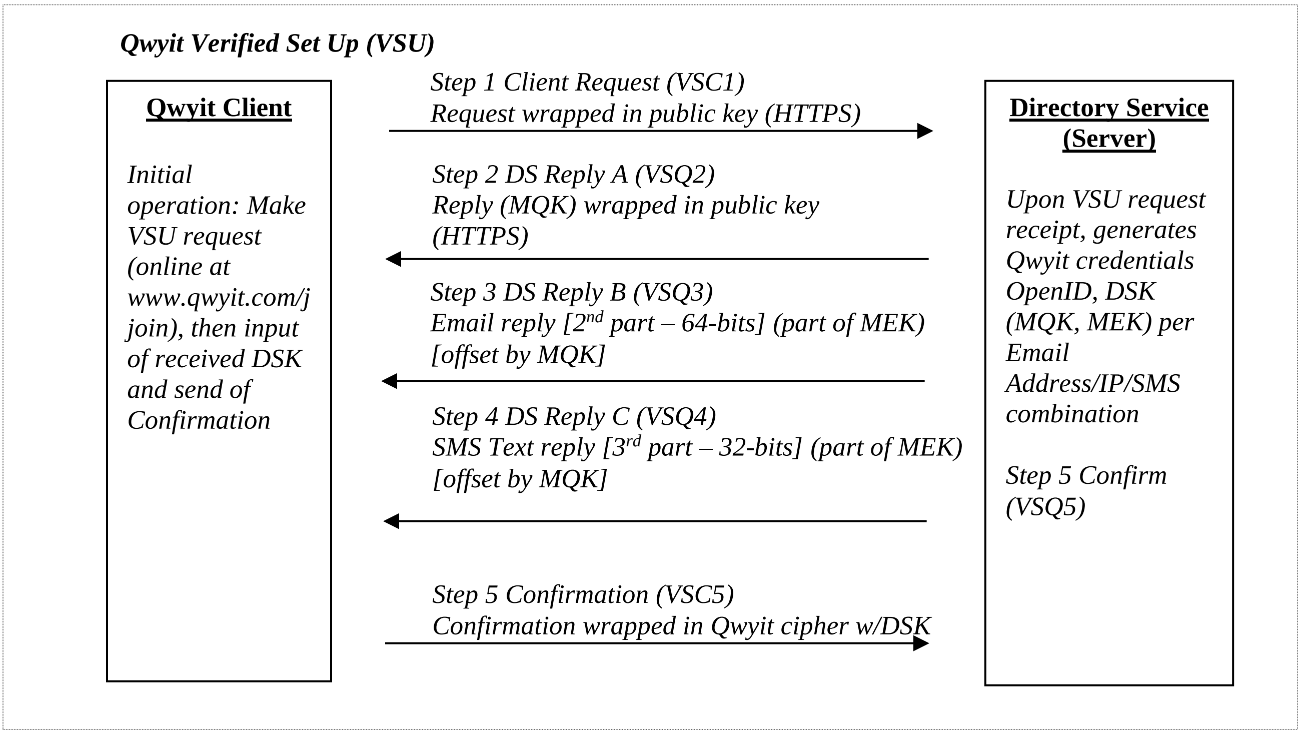
Use Case/Design

Here are the steps *every* QwyitTalk™ participant will enact, regardless of market, application or device:

NOTE: For clarity, all participants are called a “Device” (numbered as to different ones), and the QwyitTalk™ central location is the QwyitTalk™ Directory Server, “QDS”.

Join the QwyitTalk community – Perform a Verified Setup (VSU)

Note: The Step details (data formats, QwyitTalk™ processing, etc.) can be found in the QwyitTalk™ Tech Reference Guide


Qwyit Verified Set Up (VSU)
Qwyit Client
Initial operation: Make VSU request (online at www.qwyit.com/join), then input of received DSK and send of Confirmation
Step 1 Client Request (VSC1)
Request wrapped in public key (HTTPS)
Step 2 DS Reply A (VSQ2)
Reply (MQK) wrapped in public key (HTTPS)
Step 3 DS Reply B (VSQ3)
Email reply [2nd part – 64-bits] (part of MEK) [offset by MQK]
Step 4 DS Reply C (VSQ4)
SMS Text reply [3rd part – 32-bits] (part of MEK) [offset by MQK]
Step 5 Confirmation (VSC5)
Confirmation wrapped in Qwyit cipher w/DSK
Directory Service (Server)
Upon VSU request receipt, generates Qwyit credentials OpenID, DSK (MQK, MEK) per Email Address/IP/SMS combination
Step 5 Confirm (VSQ5)

Device1:

- Decide to join the QwyitTalk™ secure communication community (all Qwyit-enabled participants)

- Obtain a Qwyit-enabled application, or device or any other Qwyit-capable thing (a networked stereo, an IoT refrigerator, etc.)

- Open a web browser (whether Qwyit-enabled or not – and whether or not on the same device as the to-be-joined Qwyit-enabled thing)

- Go to www.qwyit.com/join (or after landing at www.qwyit.com, select the “Join” button)

- Enter the required and optional information
  - For Version 1, the only required information are a valid, owned/operated email address and valid, owned/operated phone number

        - This is an email address that Device1 can check, open and read
        - This is a phone number that can be accessed, SMS msg receipt capable, and read
        - These must be publicly accessible, as the QDS will send keys over public networks to both
            - For systems needing private VSU distribution, see Qwyit LLC for details

- Select "Submit" to join the QwyitTalk™ secure communication community, and securely communicate using Qwyit-enabled apps, devices, etc.

Qwyit Directory Server (QDS):

- Reply in HTTPS to the original browser VSU request w/partial DSK

- Reply in email w/partial DSK

- Reply in SMS phone text w/partial DSK

- Store Device1's information and DSK in the QDS Main QwyitTalk™ DB, temporarily (open-source, large data storage and manipulation capable, *performance is main/only criterion*)

Device1:

- User opens QwyitTalk™ application on Device (first application: QwyitFone™)
    - QwyitFone™ will know there is no initial DSK, opens up VSU input display/information

- User collects DSK from browser displayed reply
    - Enters into correct place in QwyitFone™ app

- User opens and collect from email (cut & paste preferable, to cut down on errors!)
    - Enters into correct place in QwyitFone™ app

- User opens and collects from SMS (Cut & paste, if possible)
    - Enters into correct place in QwyitFone™ app

- User selects "Join" (or "Submit") in QwyitFone™ app
    - App formats entry into correct DSK and stores
        - If possible, asks User if wants to better secure the storage of their key by entering and remembering a PIN (preferably a 6-digit, hex PIN)
            - User selects Y/N, enters when prompted

- QwyitFone™ app stores DSK (using PIN where available)

- QwyitFone™ app replies to QDS w/confirmation message
    - App notifies User it has confirmed joining the QwyitTalk™/QwyitFone Community!

- Error processing upon unconfirmed/improper confirmation TBD

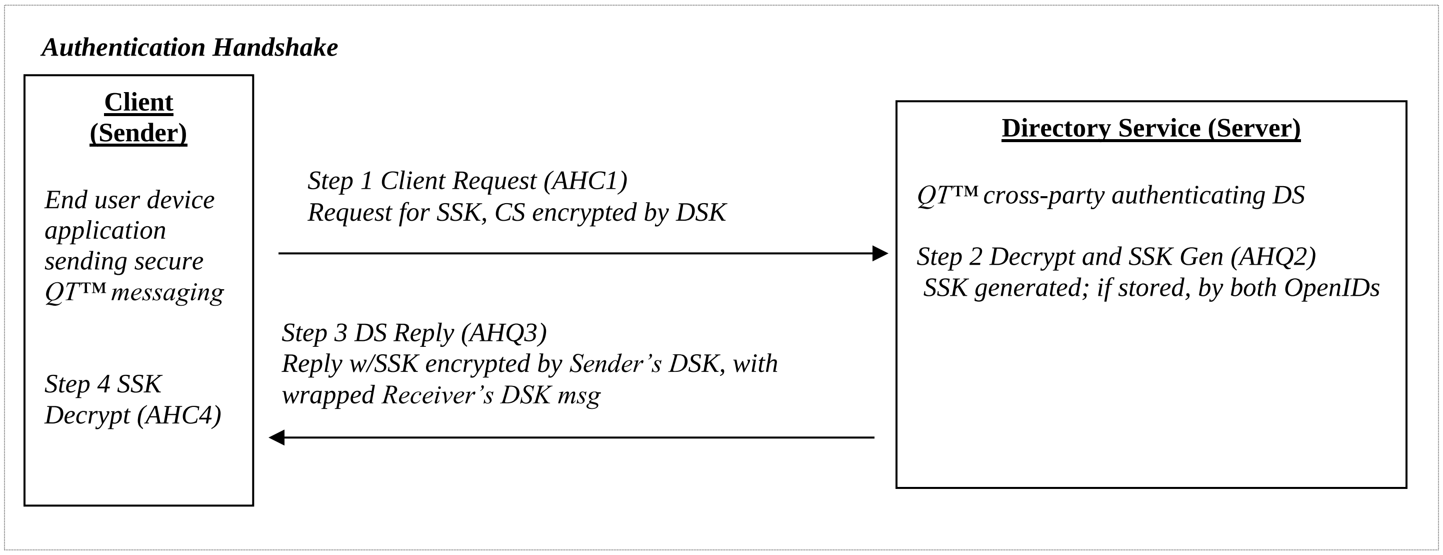
QDS:

- Upon receipt of proper confirmation, move Device1's information and DSK from the temporary tables to the permanent QDS Main QwyitTalk™ Key Distribution System DB tables

- Error processing upon unconfirmed/improper confirmation TBD

Prepare to Communicate P2P - Perform an Authenticated Handshake per communication session

Note: The Step details (data formats, QwyitTalk™ processing, etc.) can be found in the QwyitTalk™ Tech Reference Guide


Authentication Handshake
Client (Sender)
End user device application sending secure QT™ messaging
Step 4 SSK Decrypt (AHC4)
Step 1 Client Request (AHC1)
Request for SSK, CS encrypted by DSK
Step 3 DS Reply (AHQ3)
Reply w/SSK encrypted by Sender's DSK, with wrapped Receiver's DSK msg
Directory Service (Server)
QT™ cross-party authenticating DS
Step 2 Decrypt and SSK Gen (AHQ2)
SSK generated; if stored, by both OpenIDs

Device1:

- Decide to communicate w/another member of the QwyitTalk™ Community

- Initiate an AH request w/the QDS to talk to Device2
    - Send the Device1 OpenID, the intended communication destination Device2 OpenID, and the encrypted Confirmation Salt (CS) to the Server to initiate an AH

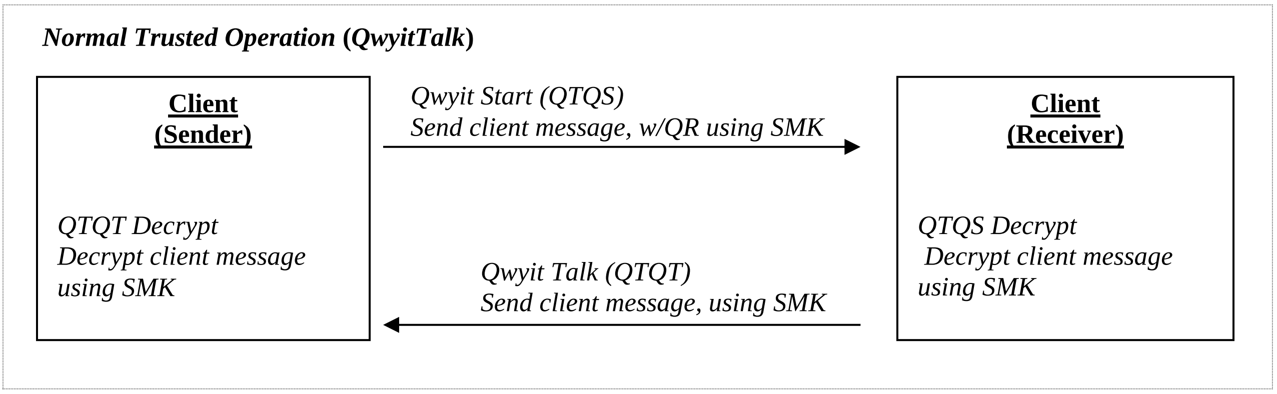
QDS:

- Reply with the session-level Session Start Key (SSK) to Device1

- Include the same SSK, double encrypted for Device2

P2P QwyitTalk™ Messaging – QwyitTalk cipher

Note: The Step details (data formats, QwyitTalk™ processing, etc.) can be found in the QwyitTalk™ Tech Reference Guide


Normal Trusted Operation (QwyitTalk)
Client (Sender)
QTQT Decrypt
Decrypt client message using SMK
Qwyit Start (QTQS)
Send client message, w/QR using SMK
Qwyit Talk (QTQT)
Send client message, using SMK
Client (Receiver)
QTQS Decrypt
Decrypt client message using SMK

Device1:

- Contact Device2 (w/QTQS initial Qwyit encrypted message)
    - Include already formatted and entered contents for first message
    - App-specific display of the content 'conversation'

Device2:

- Reply to Device1
    - QT messaging (no content requirements – back and forth, msg order, number of same-device consecutive msgs, etc. – anything is acceptable and QT readable
    - App-specific display of the content 'conversation'

Appendix 1 – QwyitTalk™ VSU, AH and Messaging Example

The following are an example of the process flow for the complete QT™ system. This output is created by the *QDS-Full-Example.exe* application available from Qwyit LLC at www.qwyit.com.

QT™ VSU Example

The VSU involves any Client who wants to join the QT™ secure messaging system, contacting the Qwyit Directory Server (QDS). This process is most likely found, performed and managed within a Client Application participating in the QT™ community. The Client and QDS process steps, and output, are as follows:

Client1

I just contacted the Qwyit.com QDS to begin a VSU and join the Qwyit Community!

QDS

```
I just received a VSU start from Client1 (who submitted a Qwyit.com website form)
First, I'll create a new ID for this Client1
The Client1 ID is: 71A2913C34ED7413
Click Continue in Client1...

Next, I'll create some new keys for this Client1
The Client1 MQK is: 7F3FB0566CA68F6264E8D990DE5CE9399DEFAE32395E1D13C82FEDEE35C4FFF9
The Client1 EOK is: C0BCEBE8CB26B8FA
The Client1 SOK is: B7D4DD28
Click Continue in Client1...

Next, I'll form the MOK and MEK using the PDAF on the EOK and MQK for this Client1
The Client1 MOK is: 7B789794ABE2DEF642B7A90876D613A5B66EC643A74160BC6B729994498237D0
The Client1 MEK is: DED71FBEE58CDD9BF1146299678A16C7A0712F10218B0ED2A7EDD65D24B36F70
Click Continue in Client1...

Now, I'll ready the keys for securely sending to this Client1
The Client1 EOK || SOK is: C0BCEBE8CB26B8FAB7D4DD28
The Client1 EOK || SOK encrypted is: 3FEB9B3E27CC375C1BBCA6B8
Now, The keys are secured and ready to send in email and SMS to this Client1
```

The Client1 EOK encrypted (EOKe) is: 3FEB9B3E27CC375C
The Client1 SOK encrypted (SOKe) is: 1BBCA6B8
Click Continue in Client1...

Lastly, The keys are sent in HTTPS (MQK), email (EOKe) and SMS (SOKe) to this Client1
[VSQ2: 71A2913C34ED7413, 7F3FB0566CA68F6264E8D990DE5CE9399DEFAE32395E1D13C82FEDEE35C4FFF9] sent in HTTPS to this Client1
[VSQ3: 71A2913C34ED7413, 3FEB9B3E27CC375C] sent in email to this Client1
[VSQ4: 71A2913C34ED7413, 1BBCA6B8] sent in SMS to this Client1
VSU processing now switches to Client1
Click Continue in Client1...

Client1

VSU processing now continues here in Client1
First, all three values from the HTTPS browser window, an email and a text msg SMS will be cut & pasted into the Client1 Qwyit application, and stored
The Client1 EOKe || SOKe is: 3FEB9B3E27CC375C1BBCA6B8
The Client1 EOK || SOK decrypted is: C0BCEBE8CB26B8FAB7D4DD28
Click Continue in Client1...

The Client1 MOK is: 7B789794ABE2DEF642B7A90876D613A5B66EC643A74160BC6B729994498237D0
The Client1 MEK is: DED71FBEE58CDD9BF1146299678A16C7A0712F10218B0ED2A7EDD65D24B36F70
Now all the master Client1 keys (MQK, MEK) have been decrypted and are stored in Client1
Click Continue in Client1...

Next, reply with VSC5 message to QDS using the new keys
First step is to create the Confirmation Salt using the PDAF and the last 128-bits of the MQK and MEK
The Client1 Confirmation Salt is: 7B3DCB65174CEC050DFCCCDC74FDEC22
Click Continue in Client1...

Next, use the QwyitTalk cipher to send the Confirmation Salt in an encrypted VSC5 message back to the QDS
The Client1 VAC5 message key is: 3339537D8F09106BCC4B5DCC1B3DCAF8A0A1977099FA4507DA6FD79E19370835
The Client1 VAC5 ciphertext is: j{†±ÂKF• ýr• Ppú-ÐåÝ³Ü=‰J“Ò^z:g
Click Continue in Client1...

Now send the encrypted VSC5 confirmation message back to the QDS
[VSC5: 71A2913C34ED7413, DBADC51DEC597FCB536F68315F4D63AFE7A169939B7F0849A5ECE330D77382B2, j{†±ÂKF• ýr• Ppú-ÐåÝ³Ü=‰J“Ò^z:g] sent using Port 4180 to the QDS
VSU confirmation processing now switches to the QDS
Click Continue in Client1...

QDS

VSU processing now continues here in QDS
First step is to create the Confirmation Salt using the PDAF and the last 128-bits of the MQK and MEK
The Client1 Confirmation Salt is: 7B3DCB65174CEC050DFCCCDC74FDEC22
Click Continue in Client1...

Next, use the QwyitTalk cipher to decrypt the received Confirmation Salt from the VSC5 message
The Client1 VAC5 received message key is: 3339537D8F09106BCC4B5DCC1B3DCAF8A0A1977099FA4507DA6FD79E19370835
The Client1 VAC5 received plaintext, which is the Confirmation Salt, is: 7B3DCB65174CEC050DFCCCDC74FDEC22
Click Continue in Client1...

Now compare the received with the computed...
As you can see, they are the same. Client1 infor is stored here in the QDS, and is now a Qwyit Community member!
VSU PROCESSING COMPLETE!
Select CLEAR for next demonstration

Client1

VSU PROCESSING COMPLETE!
Select CLEAR for next demonstration

QT™ AH Example

The AH involves any Client who wants to securely message with any other QT™ secure messaging system Client(s). In this example, the ‘app’ is configured to perform P2P secure messaging where Client1 contacts the Qwyit Directory Server (QDS) and receives an encrypted SSK, as well as a wrapped encrypted SSK for their intended recipient, Client2. This process is performed without any end-user participation, simply as the start of their messaging to Client2. The Client1, QDS and Client2 process steps, and output, are as follows:

Client1

I am going to message Client2. First, I need to perform an AH with the QDS in order to get a shared start key for use with Client2
Here is my current Client1 MQK : 554E3EE1B25DC81611E3A3FE76EF63D561BA19B1489901FCD07B1899B03C44F0
Here is my current Client1 MEK : A6DB598412DDF19E16160728B19E2788BDB88EFE7B05A0DCE88B6651E6220FFF

First, I'll calc my CS to associate with this AH
Click Continue in Client1...

The CS associated with this AH is 27989956E84BC7EB
Now, I'll send the AHC1 to the QDS with the OpenID of Client2
I need to get an OpenReturn in order to encrypt my AHC1
OpenReturn = 55E51600B8EF22340D7C0FB277D1927D738796688747DD965421A9D5A255B858
The Client2 OpenID is A67241A3186F6CCD
Click Continue in Client1...

The AHC1 message key is A4395EB99CDEF81A81D07DD2CBBAD98C45C36998B98F8F66040FAB4B04EFACE5
The AHC1 ciphertext is 73030A010C7C770F7C7B7007050F7403
[AHC1: 0123456789ABCDEF, A67241A3186F6CCD,55E51600B8EF22340D7C0FB277D1927D738796688747DD965421A9D5A255B858, 73030A010C7C770F7C7B7007050F7403] sent using Port 4180 to the QDS
AHC1 processing now switches to the QDS
Click Continue in Client1...

QDS

AHC1 processing now continues here in QDS
First step is to decrypt the received AHC1 using QwyitTalk cipher and the keys associated with the sending Client, in this case, Client1

The Client1 AHC1 received message key is: A4395EB99CDEF81A81D07DD2CBBAD98C45C36998B98F8F66040FAB4B04EFACE5

The Client1 AHC1 received plaintext, which is the CS, is: 27989956E84BC7EB
Click Continue in Client1...

Check that the Calc CS equals the sent, decrypted CS (not done here, but would be)
Check that the Client2 exists as a Qwyit member
Keys retrieved and shown in Client 2...
When this is true, create a Session Start Key (SSK) for this pair of Qwyit members to communicate
The Client1-Client2 SSK is
934FFCFFE84B2A99068A16F5BF251F94349AC6A2887647A120CE0DEC5196DDE1F29A1EFB8C54309716F6641C9D91F3283BEEC9114E94EB7144C21FBEBEB73471
Click Continue in Client1...

Now, I'll send out the AHQ3 to just Client1 with Client2's wrapped SSK
I need to get an OpenReturn in order to encrypt the AHQ3 for Client1
OpenReturn = 4DA1A27A59137E39562917FFE4A677EADED42D0305FDDF86A8E2D58ACA74B03B
Click Continue in Client1...

Only AHQ3 is to Client1...
The AHQ3 Client1 message key is 4D94ADE3BB2B35D47B28F3ECB8DED39FD67373B964A3BF6F683FEEDB68B3ED1A
The AHQ3 first ciphertext is
0D770D7207070375077A060001747D0D07740A7977050376007E76707575007277020E727405030B0E0C760576717777040870037501010103097B050100747072760075700103717A010
77600057D030674740E700774007B7C7D7402000B7E77747276740A73080271780707040177020C707474030607747D000476700670
The AHQ3 Client2 message key is AE088ECDB215E3727E058172A7096BC6DEE633999D141BEA7E437BD7869AA9DC
The AHQ3 second ciphertext is
7876047E7E060502070A057777720E0B07730874090771070371020C07047A0277717C777005780B017C06020575047005757776070601740D070077057D017207770979090005067A710
40176030E05067376030E050671787309087071710E77070073700A08080D0108007400727003717701060406727A737B76720D7372
[AHQ3: 0123456789ABCDEF, 4DA1A27A59137E39562917FFE4A677EADED42D0305FDDF86A8E2D58ACA74B03B,
0D770D7207070375077A060001747D0D07740A7977050376007E76707575007277020E727405030B0E0C760576717777040870037501010103097B050100747072760075700103717A010
77600057D030674740E700774007B7C7D7402000B7E77747276740A73080271780707040177020C707474030607747D000476700670,7876047E7E060502070A057777720E0B077308740
90771070371020C07047A0277717C777005780B017C06020575047005757776070601740D070077057D017207770979090005067A71040176030E05067376030E05067178730908707171
0E77070073700A08080D0108007400727003717701060406727A737B76720D7372] sent using Port 4180 to Client1
AHQ3 processing now switches to Client1 and Client2
Click Continue in Client1...

Client1

Client1 now decrypts their section of the AHQ3 using QwyitTalk to get the SSK for their session
The Client1 AHQ3 received message key is: 4D94ADE3BB2B35D47B28F3ECB8DED39FD67373B964A3BF6F683FEEDB68B3ED1A
The Client1 AHQ3 received plaintext, which is the SSK, is:
934FFCFFE84B2A99068A16F5BF251F94349AC6A2887647A120CE0DEC5196DDE1F29A1EFB8C54309716F6641C9D91F3283BEEC9114E94EB7144C21FBE
BEB73471
Click Continue in Client1...

After sending (the QDS), and receiving (Client1), both perform a PDAF PFS NIL communication DSK key update
This is accomplished using the MQK and MEK in a PDAF. Optionally, for a 1,024-bit length, and then performing an OWC to pare to 512-bits

Here is the PDAF result for Client1 new MQK/MEK keys:
2054844D0EC32E83F881D65485F8BCE96DAA2A4A10299030D33BDCD9C57B8C06175F0FF98ABBF97B471290DF0C78CE1D7E7B014A4190BD302CB05CD
C652C3551DA2FFFF2733C99FCBFD988E9299F745E5189D26CB1A40EEB24B05785D4314A01627CBCC73843160740C6195883F0176E28B6E2B1F38A31F
C146F089D0E81950EB1E14112C3B02B710750042FBB2E0B653C2714E45851C841BF791DED838095D36D789882138AD240F4488EC7FC6B7CD6626A897D
CB624C4100CE6DE87E7A822A6631411FC0227BE78EC99D49077CFCE4D9C1CCF5D4924230B5C96CD93E1F27933B2D4DD4C5A3676F76D4045F017F13C
11A3525F5DC04F9DAB5B9567EC16A7E2F
Here is OWC final result for Client1 new MQK key: 29C1EF0B7939D77734CE1B930E96124684F82682B39CCFAE521E5983EB19BE86
Here is OWC final result for Client1 new MEK key: C450C425656D73B26D841B630526109CEF111DD5D144164DB874948704B5D051
The AH is Complete! Now Client1 and 2 can message securely...
Lastly, the QDS performs the NIL communication key update...
Click Continue in Client1...

QDS

I perform the exact same PFS key updates for Client1

Here is the PDAF result for Client1 new MQK/MEK keys:
2054844D0EC32E83F881D65485F8BCE96DAA2A4A10299030D33BDCD9C57B8C06175F0FF98ABBF97B471290DF0C78CE1D7E7B014A4190BD302CB05CD
C652C3551DA2FFFF2733C99FCBFD988E9299F745E5189D26CB1A40EEB24B05785D4314A01627CBCC73843160740C6195883F0176E28B6E2B1F38A31F
C146F089D0E81950EB1E14112C3B02B710750042FBB2E0B653C2714E45851C841BF791DED838095D36D789882138AD240F4488EC7FC6B7CD6626A897D
CB624C4100CE6DE87E7A822A6631411FC0227BE78EC99D49077CFCE4D9C1CCF5D4924230B5C96CD93E1F27933B2D4DD4C5A3676F76D4045F017F13C
11A3525F5DC04F9DAB5B9567EC16A7E2F
Here is OWC final result for Client1 new MQK key: 29C1EF0B7939D77734CE1B930E96124684F82682B39CCFAE521E5983EB19BE86
Here is OWC final result for Client1 new MEK key: C450C425656D73B26D841B630526109CEF111DD5D144164DB874948704B5D051

The AH is Complete! Now Client1 and 2 can message securely and I am Finished!...
Click Continue in Client1...

Client2

These are for informational purposes – Client2 is not a participant in the routine AH
Here are the retrieved Client2 MQK : 46594234B75B9DAA00E0DD6A3563870ABCA7A02BEE5D767150EEB97CBF8363EF
Here are the retrieved Client2 MEK : 41B1D16C88CAE1010168B2CB74875AD1F79C5166560AA7C208BEBF243BAF12F2

The AH is Complete! Now Client1 and 2 can message securely...
Next, Client 1 performs the NIL communicaiton key update...
Click Continue in Client1...

QT™ Messaging Example (continuing from the AH..)

Client1

Now I am going to talk to Client2 in our application...

First, I'll create a Qwyit Return (QR) value
QR is
21FBFCFE73E926F4FF0E7F193C49EFA7410D2DDF28BEFAE92B47E81576A272C9B561F30467E4C36A6A34E80981567263EBCA53C59D2D759F1974641D
B8D3D6DD
Click Continue in Client1...

Next, I'll use the SSK from the QDS and the just created QR to create a Session Master Key (SMK,in 2 halves, SQK and SEK)
New Client1-Client2 SQK is B43AE8ED5B24408DF588850EEB6EFE3B7597E371A024318A4B05E5F1C7384FAA
New Client1-Client2 SEK is A7FB01FFE338F3F1702A4C151EE7658B16A81CD6DBB150005D3673CB668A0A4E
Click Continue in Client1...

Now ready the message, which will be: Now is the time for all good men to come to the aid of the party
Then, encrypt it using QwyitTalk...
OpenReturn for this message is 7969F5F110C59D7521FDC1C59B10BD8F97D96A0973DECE0DCD5DB0C1E596FD0D
The QTQS message key is 7C4BC934713CEDA7495E45AC98DABD1E81CC99808DAAE2CFDC0BEC7B3EBAAB73
QwyitTalk Qwyit Start (QTQS) is 0123456789ABCDEF,
21FBFCFE73E926F4FF0E7F193C49EFA7410D2DDF28BEFAE92B47E81576A272C9B561F30467E4C36A6A34E80981567263EBCA53C59D2D759F1974641D
B8D3D6DD, 7969F5F110C59D7521FDC1C59B10BD8F97D96A0973DECE0DCD5DB0C1E596FD0D,
4DA1A27A59137E39562917FFE4A677EADED42D0305FDDF86A8E2D58ACA74B03B,7876047E7E060502070A057777720E0B07730874090771070371020C0
7047A0277717C777005780B017C060205750470057577760706017 40D070077057D01720777097909000 5067A71040176030E05067376030E05067178730908
7071710E77070073700A08080D010800740072700371770106040 6727A737B76720D7372,792C43622A4A13405F5413372C29241752564765555 92D635E572
B256229542B18452C635A56555518302E61315A2666252A54622A2517365B2062312030434A
Now, QT processing continues in Client 2...
Click Continue in Client1...

Client2

I've just received a QTQS message from Client1...I'll process it!
First, I need to unwrap the SSK...
The Client2 AHQ3 received message key is: AE088ECDB215E3727E058172A7096BC6DEE633999D141BEA7E437BD7869AA9DC

The Client2 AHQ3 received plaintext, which is the SSK, is:
934FFCFFE84B2A99068A16F5BF251F94349AC6A2887647A120CE0DEC5196DDE1F29A1EFB8C54309716F6641C9D91F3283BEEC9114E94EB7144C21FBEBEB73471

Next, I need to unwrap the SMK...
Now, take the just received QR and create the SMK for this message
New Client1-Client2 SQK is B43AE8ED5B24408DF588850EEB6EFE3B7597E371A024318A4B05E5F1C7384FAA
New Client1-Client2 SEK is A7FB01FFE338F3F1702A4C151EE7658B16A81CD6DBB150005D3673CB668A0A4E
Click Continue in Client1...

The QTQS message key is 7C4BC934713CEDA7495E45AC98DABD1E81CC99808DAAE2CFDC0BEC7B3EBAAB73
QTQS plaintext message is Now is the time for all good men to come to the aid of the party
Now I'll reply back to Client1...message is: Why won't anyone throw me a party with cup cake for my birthday?
Click Continue in Client1...

OpenReturn for this message is 9D1A5E78202F95449BBE8EFF5C8925A370FDE77CA6318C852E176AE50C8FE4E8
The QTQT message key is 5E6727967FD25485F855B7B8814BAFF011B88E1B165BAB81F2B7DB7DD075BA4D
QwyitTalk Qwyit Talk (QTQT) is A67241A3186F6CCD, 9D1A5E78202F95449BBE8EFF5C8925A370FDE77CA6318C852E176AE50C8FE4E8,
622D4F17455857114366255C4C5B5650664C5D472D4062555D115562312734444811355 14C2D11214446152120295D11205D3017293B17262D42435D26204D7B

Now, QT processing continues in Client 1...
Click Continue in Client1...

Client1

I've just received a QTQT message from Client2...I'll process it using our shared SMK keys and the sent Open Return!

The QTQT message key is 5E6727967FD25485F855B7B8814BAFF011B88E1B165BAB81F2B7DB7DD075BA4D
QTQT plaintext message from Client 2 is Why won't anyone throw me a party with cup cake for my birthday?

Click Continue in Client1...

After sending and receiving and the SMK key life is reached, both Clients perform a PDAF PFS NIL communicaiton DSK key update
This is accomplished using the SQK and SEK in a PDAF. Optionally, for a 1,024-bit length, and then performing an OWC to pare to 512-bits
First, here in Client 1...
Click Continue in Client1...

Here is the PDAF result for Client1 new SQK/SEK keys:
FFB765EB3BAC35B2DDDB3EEC4565654F0FAA53599515392E35F82C9B62D63442F6B9CDCB33F92835AD83ECB1CB8CDCDEE518F8CBBF3C8BC5561DD

```
8DF7B15EA87B8BFB3C8081478F55D6F6369DDA1213C573F816561EE153A0FC98D790ABDE12938823A93BA72F8D5DA6D78E5CF956553A9D1E88C1C98
8C8FBE7489DFF113F38F84329C4B30A2B512E5316CF31E7F0F75E84B0231D75F21D8753F94CE650A8372AC120CCC43AF90FDD3EFB6E79CE8986F8692
24E623A4466FC98F2F46AF8C7FF2011F28DB33AADE6B23663F35530EFE461DE71FAFE869FFD983401FAC45ECE75233EA20C08372BEBBA07BF5BC35A
010BB7F7CFAF69412A67B39F98013A4582CCF
Here is OWC final result for Client1 new SQK key: E2B9E68DA81A9BB3F48EE6C087E48376549768A87BAC749B3977AF31BE5C268F
Here is OWC final result for Client1 new SEK key: 10A864B15C288EDAE5096FE6B4069A57682CB996A2478A166395D302C884EDEB
Now, update the SMK in Client 2...
Click Continue in Client1...
```

Client2

```
Here is the PDAF result for Client2 new SQK/SEK keys:
FFB765EB3BAC35B2DDDB3EEC4565654F0FAA53599515392E35F82C9B62D63442F6B9CDCB33F92835AD83ECB1CB8CDCDEE518F8CBBF3C8BC5561DD
8DF7B15EA87B8BFB3C8081478F55D6F6369DDA1213C573F816561EE153A0FC98D790ABDE12938823A93BA72F8D5DA6D78E5CF956553A9D1E88C1C98
8C8FBE7489DFF113F38F84329C4B30A2B512E5316CF31E7F0F75E84B0231D75F21D8753F94CE650A8372AC120CCC43AF90FDD3EFB6E79CE8986F8692
24E623A4466FC98F2F46AF8C7FF2011F28DB33AADE6B23663F35530EFE461DE71FAFE869FFD983401FAC45ECE75233EA20C08372BEBBA07BF5BC35A
010BB7F7CFAF69412A67B39F98013A4582CCF
Here is OWC final result for Client2 new SQK key: E2B9E68DA81A9BB3F48EE6C087E48376549768A87BAC749B3977AF31BE5C268F
Here is OWC final result for Client2 new SEK key: 10A864B15C288EDAE5096FE6B4069A57682CB996A2478A166395D302C884EDEB
Click Continue in Client1...
```

Client1

```
This concludes the AH and QwyitTalk demonstration. Thank you!
```

Client2

```
This concludes the AH and QwyitTalk demonstration. Thank you!
```

Appendix 2 – QwyitTalk™ VSU, AH and Messaging Code Calls

The following are the code calls for the example of the process flow for the complete QT™ system. These calls create the output, and those required to build the QT™ Security as a Service, of the *QDS-Full-Example.exe* app available from Qwyit LLC at www.qwyit.com.

QT™ VSU Example

Client1

Client, initiate VSU within the application (and as required/desired) with the Directory Service/Server (Step 1, VSC1)
Goto www.Qwyit.com/Qwyit where an HTTPS session will begin. Enter the required information.
This generates a VSU start on the MyQTApp client application and the QDS (VSC1)
Client app now awaits receipt and entry of the key

Minimal Parameters (captured in web form): Name, Email Address, Phone number [optional: IP address, physical address, etc.]

QDS

Directory Service/Server reply to client (Steps 2,3, and 4 - VSQ2, VSQ3, VSQ4)
The client has submitted a VSU on the webpage, and is waiting for the reply

Generate OpenID (recommend 16 digit, 64-bit unique IDs), DSK for client
Generate a random, unique 16-hex digit, 64-bit OpenID for this client (stored in the DB per email/SMS/IP (and whatever other parameters in MyQTApp))
```
sOpenID_Client1 = GetRandom (16)
```

Generate random 352-bits that includes three (3) parts
Master Qwyit Key (MQK, 64 hex digits, 256-bits)
```
sClient1_MQK = GetRandom (64)
```

Email Offset Key (EOK, 16 hex digits, 64-bits)
```
sClient1_EOK = GetRandom (16)
```

SMS Offset Key (SOK, 8 hex digits, 32-bits)
```
sClient1_SOK = GetRandom (8)
```

Perform a PDAF(EOK, SOK) generating 8 rounds (cycling through each round of SOK moving the start 1 digit to the right)
Result is the Master Offset Key (MOK, 64 hex digits, 256-bits)
```
sClient1_MOK = PubDS.PDAF(sClient1_EOK, 32, 0, sClient1_SOK, 1, 0, 0)
```

Perform a PDAF(MQK, MOK) generating 1 round
Result is a the Master Exchange Key (MEK, second 64 hex digits, 256-bits) - store with MQK as this new client's Qwyit keys
    sClient1_MEK = PubDS.PDAF(sClient1_MQK, 32, 0, sClient1_MOK, 1, 0, 0)

Encrypt EOK and SOK using MQK
Concatenate EOK and SOK
    sClient1_TempEOK_SOK = sClient1_EOK & sClient1_SOK

MOD16 add the first 96-bits (24 hex digits) of MQK with the concatenated EOK and SOK
    sClient1_EOK_SOK_encrypted = PubDS.MOD16(Left(sClient1_MQK, 24), sClient1_TempEOK_SOK)

Separate the encrypted EOKe (LEFT 16 or 64-bits) and SOKe (RIGHT 16 or 32-bits) and ready to send in email and SMS respectively

Reply (VSQ2) with OpenID, (MQK) during the HTTPS session on the webpage
Reply Step 3 to email address, sending OpenID and EOKe
Reply Step 4 to SMS number, sending OpenID and SOKe

Client1

Client decrypt of reply and confirmation (Step 5, VSC5)
    Client will cut & paste the session-shown OpenID, EOKe and SOKe into the MyQTApp, and OpenID and MQK will appear in the app/window, as the HTTPS session has received those
    Open email message to reveal OpenID and EOKe; cut & paste (or type) into application (both values)
    Open SMS text message to reveal OpenID and SOKe; cut and paste (or type) into application (both values)
    Click button to "Store Key" (or some relevant, pertinent UI text)
    App will check that all 3 OpenIDs match
    App will decrypt EOKe and SOKe, and create MEK

Concatenate EOKe and SOKe
    sClient1_TempEOK_SOK_encrypted = Left(sClient1_EOK_SOK_encrypted, 16) & Right(sClient1_EOK_SOK_encrypted, 8)

MOD16D the first 96-bits (24 hex digits) of MQK with the concatenated EOKe and SOKe
    sClient1_EOK_SOK_decrypted = PubDS.MOD16D(sClient1_TempEOK_SOK_encrypted, sClient1_MQK)

Separate the decrypted EOK and SOK and ready for use to create MEK
Perform a PDAF(EOK, SOK) generating 8 rounds (cycling through each round of SOK moving the start 1 digit to the right)
Result is the Master Offset Key (MOK, 64 hex digits, 256-bits)
    sClient1_MOK_decrypted = PubDS.PDAF(Left(sClient1_EOK_SOK_decrypted, 16), 32, 0, Right(sClient1_EOK_SOK_decrypted, 8), 1, 0, 0)

Perform a PDAF(MQK, MOK) generating 1 round
Result is a the Master Exchange Key (MEK, second 64 hex digits, 256-bits) - store with MQK as this new client's Qwyit keys

```
    sClient1_MEK_decrypted = PubDS.PDAF(sClient1_MQK, 32, 0, sClient1_MOK_decrypted, 1, 0, 0)
```

Concatenate MQK and MEK making complete 512-bit DSK – Insert into MyQTAPP storage (store DSK, OpenID in cookie, file, db – method per MyQTApp requirements)

Reply (VSC5) to QDS with Confirmation message
Perform Qwyit cipher using DSK (MQK and MEK), generating message key (W)
Use message key to encrypt confirmation message, which is a 128-bit, 32 hex digit ID salt created by using the last 32 digits (128-bits) of the MQK in a PDAF with the last 32 digits (128-bits) of the MEK

Perform a PDAF(MQK last 128-bits, MEK last 128-bits); result is Confirmation Salt

```
    sClient1_ConfirmationSalt = PubDS.PDAF(Right(sClient1_MQK, 32), 16, 0, Right(sClient1_MEK_decrypted, 32), 1, 0, 0)
```

Perform Qwyit encrypt using W and CS, result is CS encrypted (CSe)

```
        sOpenReturn = GetRandom (64) 'return a 64-digit 256-bit random number
        sClient1_ConfirmationSalt_encrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK_decrypted, sOpenReturn, sClient1_ConfirmationSalt,"","")
```

'*****************************
[**THIS QwyitSCX Cipher function does not exist in the C/Java/C++ SDKs – the function needs to be built, added to the library – it is a combination of the existing SDK functions, along with a simple XOR – here are the call steps in pseudo code]

```
Public Function QwyitSCX(sQK As String, sEK As String, sOR As String, Optional sPT As String, Optional sCT As String, Optional sNewR As String)

  IF sNewR is included (not blank), then use it
       sR = sNewR
  Else Create R
       sR = MOD16(sEK, sOR)
  End If

  'Combine R and QK, resulting in a n-bit 'alphabet', A
  sA = Combine(sR, sQK)

  'Extract n-bit key W out of A using QK
  sW = Extract(sA, sQK)


  IF Encrypt the plaintext message using W
      sTtemp = sW Xor sPT
  Else Decrypt the ciphertext message using W
      sTtemp = sW Xor sCT
```

```
    End If

    'Get Next R for both Enc/Dec
    sNextR = MOD16(sW, sR)

RETURN sNextR, sW (if needed), sOUTPUT (where sOUTPUT is either the CT if encrypt, or the PT if decrypt)

End Function
'**********************************
```

Send (VSC5) output (OpenID, OR, CSe) sent using Port 4180 to the QDS
    [VSC5: OpenID, OR, CSe]

CONTENT and FLAG values will determine message (This is TBD by development team: it will be standardized in Universal QT™ Connection Doc)

QDS

Message received on Port 4180
CONTENT and FLAG values will determine message (This is TBD by development team – standardized in Universal QT™ Connection Doc)

QDS decrypt of confirmation message (Step 5, VSQ5)

[**THIS is the above QwyitSCX to-be-built C/Java/C++ function]
Perform Qwyit decrypt using W and CSe, result is CS received decrypted (CS)

```
        sClient1_ConfirmationSalt_decrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK, sOpenReturn,"", sCiphertext,"")
```

Perform a PDAF(MQK 128-bits, MEK last 128-bits); result is Confirmation Salt generated

```
        sClient1_ConfirmationSalt_qds = PubDS.PDAF(Right(sClient1_MQK, 32), 16, 0, Right(sClient1_MEK_decrypted, 32), 1, 0, 0)
```

Compare CS received decrypted with CS generated

```
        If sClient1_ConfirmationSalt_qds = sPlaintext Then
                Match confirmed, store IP Address, OpenID, DSK, email address, SMS (and whatever other required session ID was collected) into QDS
                MyQTApp section - method?'
        Else
                Doesn't match, error sent - "Uh oh...they aren't the same... Something is wrong - enact error routines...Client1 is NOT a QwyitTalk member!"
        End If
```

Client1

If no error returned, VSU PROCESSING COMPLETE!

QT™ AH Example

Client1

Retrieve MQK and MEK from myQTApp (the client application that has performed a VSU w/QDS)

First, calc my CS to associate with this AH (this is myQTApp system defined, as recommended by Qwyit – either as per above definition in the AH section, or as here: PDAF the MQK and MEK to full length, then perform an OWC on the result – switch the Value and Offset keys from the VSU CS above)

```
sClient1_CS_PDAF = PubDS.PDAF(sClient1_MEK, 32, 0, sClient1_MQK, 1, 0, 0)
sClient1_CS = PubDS.OWC(sClient1_CS_PDAF,1)
```

Next, generate an OpenReturn (an IV) for this AHC1 message:

```
sOpenReturn = GetRandom (64) 'return a 64-digit 256-bit random number
```

Then retrieve OpenID of my intended recipient from myQTApp (the client app has stored, and/or a method to retrieve other QTApp participant OpenIDs – where the recipient can be chosen from)

Now Encrypt my CS
[**THIS is the above QwyitSCX to-be-built C/Java/C++ function]

```
sClient1_AHC1_encrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK, sOpenReturn, sClient1_CS,””,””)
```

CONTENT and FLAG values will determine message (This is TBD by development team – standardized in Universal QT™ Connection Doc):
Send [AHC1: SenderOpenID, ReceiverOpenID, OR, CT] where CT is sCiphertext, using Port 4180 to the QDS

QDS

Decrypt AHC1 from Client using respective OpenID and DSK in Qwyit Cipher
The sOpenReturn is used from the received AHC1 message....and the MQK and MEK are retrieved using the received sOpenID in the AHC1
This is done by retrieving the MEK/MQK from the QDS database of the OpenID from the sending Client 1 (SenderOpenID)

After retrieval, decrypt the received ACH1
[**THIS is the above QwyitSCX to-be-built C/Java/C++ function]

```
sClient1_AHC1_decrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK, sOpenReturn, ””, sCiphertext,””)
```

Check that Client who is requesting is the same as encrypted request and Receiver client exists
This is done by separating the Client2 OpenID and the CS in the received Plaintext

```
First, check that the Calc CS equals the sent, decrypted CS
sClient1_CS_PDAF = PubDS.PDAF(sClient1_MEK, 32, 0, sClient1_MQK, 1, 0, 0)
sClient1_CS = PubDS.OWC(sClient1_CS_PDAF,1)
```

```
    If Client1_CS = sPlaintext
        'Continue!
    Else
        'Proceed w/any system error handling, including Sending error to Client 1, stating 'Incorrect AHC1 received!', and mark in DB, etc.
    End If
```

Proceed by retrieving the MEK/MQK keys for Client2 from the QDS database from the ReceiverOpenID in the received AHC1

Create Session Start Key (SSK, 128 hex digits, 512-bits)
[IF to be stored, by MID and SenderOpenID, ReceiverOpenID – requirement by 3-letter agency ONLY]

```
    sClient1_Client2_SSK = GetRandom (128)
```

QDS reply to Clients (Step 3, AHQ3)
Send SSK Accept (AHQ3) to Sender Client encrypted in Qwyit Cipher using Client1 DSK (If Receiver Client does not exist in QDS – Handle as per myQTApp instructions.) The AHQ3 includes the wrapped SSK encrypted by that Receiver client's DSK

First, generate an OpenReturn

```
    sOpenReturn = GetRandom (64) 'return a 64-digit 256-bit random number
```

Next, encrypt the SSK for the Sender Client using their DSK
[**THIS is the above QwyitSCX to-be-built C/Java/C++ function]

```
    sClient1_AHQ3_encrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK, sOpenReturn, sClient1_Client2_SSK, ””, “”)
```

Now, Encrypt the Receiver Client SSK bundle using their DSK
[**THIS is the above QwyitSCX to-be-built C/Java/C++ function]

```
    This requires TWO encryptions of the SSK
    sClient2_AHQ3_encrypted_v1 = PubDS.QwyitSCX(sClient2_MQK, sClient2_MEK, sOpenReturn, sClient1_Client2_SSK, ””, “”)

    The second encryption uses the returned sNextR
    sClient2_AHQ3_encrypted_v2 = PubDS.QwyitSCX(sClient2_MQK, sClient2_MEK, sOpenReturn, sCiphertext2_v1, sNextR)
```

CONTENT and FLAG values will determine message (This is TBD by development team – standardized in Universal QT™ Connection Doc):
Send [AHQ3: sOpenID_Client1, sOpenReturn, sCipherTextOut, sCipherTextOut2] using Port 4180 to Client1

Client1

Client1 decrypts SSK (Step 4, AHC4)
Sender Client1 decrypts AHQ3 from QDS using DSK in Qwyit Cipher

Perform Qwyit decrypt using W and CSe, result is CS received decrypted (CS)

```
            sClient1_SSK_decrypted = PubDS.QwyitSCX(sClient1_MQK, sClient1_MEK, sOpenReturn, , sCiphertext)
```

Client1 only reply to QDS if unable to decrypt key (for whatever reason), sending an AHE for error – IF NO ERROR, the AH is Complete! Proceed

Client1 will store the sOpenReturn and sCipherTextOut2, or hold in memory after receipt, for including in the following QTQS send to Client 2

Now, Client1 performs a PDAF PFS NIL communciation DSK key update (QDS will in the next section)

```
            Perform a PDAF(MQK 256-bits, MEK 256-bits) for two rounds, creating 512-bit result w/MQK, MEK halves
            Optionally, extend the above PDAF for 2 rounds (doubling length) and perform an OWC
            sClient1_Key_PFS_Update = PubDS.PDAF(sClient1_MQK, 256, 0, sClient1_MEK, 1, 0, 0)
            sClient1_MQK_MEK_New = PubDS.OWC(sClient1_Key_PFS_Update, 1)
            sClient1_MQK_New = Left(sClient1_MQK_MEK_New, 64)
            sClient1_MEK_New = Right(sClient1_MQK_MEK_New, 64)
```

QDS

Performs the exact same PFS key updates for Client1 – this can be done after performing the AHQ3 send…and no error returned…so 'wait time' is important, AND/OR performed immediately and stored differently, then handled by DB to update…

```
            sClient1_Key_PFS_Update = PubDS.PDAF(sClient1_MQK, 256, 0, sClient1_MEK, 1, 0, 0)
            sClient1_MQK_MEK_New = PubDS.OWC(sClient1_Key_PFS_Update, 1)
            sClient1_MQK_New = Left(sClient1_MQK_MEK_New, 64)
            sClient1_MEK_New = Right(sClient1_MQK_MEK_New, 64)
```

The AH is Complete! Now Client1 and 2 can message securely and QDS is Finished!...

QT™ Messaging Example (continuing from the AH..)

Client1

Now I am going to talk to Client2 in our MyQTApp...

Client 1 sends Qwyit Start to Client Receiver (Qwyit Start, QTQS)

```
            Sending Client1 generates a Qwyit Return (QR, 128 hex digits, 512-bits)
            sClient1_2_QR = GetRandom (128)
```

Using the AH SSK, from the received QDS AHQ3 message, for this communication:
Perform MOD16(SSK,QR); result is the Session Master Key (SMK, in 2 halves, SQK and SEK)

```
            sSMK_Client1 = PubDS.MOD16(sClient1_Client2_SSK, sClient1_2_QR)
            sSQK_Client1 = Left(sSMK_Client1, 64)
```

```
    sSEK_Client1 = Right(sSMK_Client1, 64)
```

Create new message content (IMSG, the Initial Message opening communication) in the MyQTApp

Perform Normal Trusted Operation using Qwyit Cipher on IMSG with SMK (SQK and SEK)

```
    sIMSG = "Now is the time for all good men to come to the aid of the party" (EXAMPLE MyQTApp message)

    sOpenReturn = GetRandom (64)
    sClient1_IMSG_encrypted = PubDS.QwyitSCX(sSQK_Client1, sSEK_Client1, sOpenReturn, sIMSG, ,"")
```

Send Qwyit Start (QTQS) message from within MyQTApp to Client 2

QTQS:OpenID, QR, OR, Ciphertext of IMSG, which is [QTQS: OpenID, QR, OR, OR2, CT2, CT] where OR2 and CT2 are from the AHQ3 QDS message, as stored and/or in memory to include in this send

Client2

Receiving Client decrypts QTQS (Qwyit Talk, QTQT)

```
    Determines SMK from the QS message by performing the same MOD16(SSK,QR) – THIS takes TWO decryptions
    sClient2_SSK_decrypted_v1 = PubDS.QwyitSCX(sClient2_MQK, sClient2_MEK, sOR2, , sCT2,"")
        Where sCT2_v1 and sNextR are returned from the first SSK decryption
    sClient2_SSK_decrypted_v2 = PubDS.QwyitSCX(sClient2_MQK, sClient2_MEK, sOR2, , sCT2_v1,sNextR)
    Now Client 2 has the fully decrypted SSK that is being used by Client 1 for this message, called sPlaintext2
```

Next, create the SQK and SEK used by Client1 in the QTQS

```
    sSMK_Client2 = PubDS.MOD16(sPlaintext2, sClient1_2_QR)
    sSQK_Client2 = Left(sSMK_Client2, 64)
    sSEK_Client2 = Right(sSMK_Client2, 64)
```

Now, having the SQK and SEK, commences Normal Trusted Operation message, by decrypting the Client 1 Ciphertext (of IMSG)

```
    sClient2_IMSG_decrypted = PubDS.QwyitSCX(sSQK_Client2, sSEK_Client2, sOpenReturn, "", sCiphertext, "")
```

MyQTApp portrays sPlaintext as the received Client 1 message

Clients continue communication by performing Normal Trusted Operation messaging w/Qywit Cipher using SMK (QwyitTalk, QTQT Encrypt/Decrypt)

```
    sIMSG_2 = "Why won't anyone throw me a party with cup cake for my birthday?" (EXAMPLE MyQTApp reply message)
    sOpenReturn = GetRandom (64)
    sClient2_IMSG2_encrypted = PubDS.QwyitSCX(sSQK_Client2, sSEK_Client2, sOpenReturn, sIMSG_2,"", "")
```

Send Qwyit Talk (QTQT) from within MyQTApp…

Send [QTQT: OpenID, OR, CT] where CT is the sCiphertext of just encrypted message contents

Client1

Messaging continues with QTQTs sent back and forth in MyQTApp…
QTQT Decrypt w/SMK (SQK, SEK) in Qwyit Cipher to reveal secure message contents

```
    sClient1_IMSG_decrypted = PubDS.QwyitSCX(sSQK_Client1, sSEK_Client1, sOpenReturn, “”, sCiphertext,”” )
```

MyQTApp portrays sPlaintext as the received Client 2 message

At known, system defined end of SMK key life reached (# of messages, time, etc.), both Clients perform a PDAF PFS Nil communication SMK key update

```
    Perform a PDAF(SQK 256-bits, SEK 256-bits) for two rounds, creating 512-bit SMK result w/SQK, SEK halves
    Optionally, extend the above PDAF for 2 rounds (doubling length) and perform an OWC
    sClient1_Key_PFS_Update = PubDS.PDAF(sSQK_Client1, 256, 0, sSEK_Client1, 1, 0, 0)
    sClient1_SQK_SEK_New = PubDS.OWC(sClient1_Key_PFS_Update, 1)
    sClient1_SQK_New = Left(sClient1_SQK_SEK_New, 64)
    sClient1_SEK_New = Right(sClient1_SQK_SEK_New, 64)
```

Client2

At known, system defined end of SMK key life reached (# of messages, time, etc.), both Clients perform a PDAF PFS NIL communicaiton SMK key update

```
    Perform a PDAF(SQK 256-bits, SEK 256-bits) for two rounds, creating 512-bit SMK result w/SQK, SEK halves
    Optionally, extend the above PDAF for 2 rounds (doubling length) and perform an OWC
    sClient2_Key_PFS_Update = PubDS.PDAF(sSQK_Client2, 256, 0, sSEK_Client2, 1, 0, 0)
    sClient2_SQK_SEK_New = PubDS.OWC(sClient2_Key_PFS_Update, 1)
    sClient2_SQK_New = Left(sClient2_SQK_SEK_New, 64)
    sClient2_SEK_New = Right(sClient2_SQK_SEK_New, 64)
```

Client1

Continue as QTQT above, including system defined PFS SMK updates

Client2

Continue as QTQT above, including system defined PFS SMK updates