QwyitKey Service (QwyitKey™)
A Qwyit Authentication Service

Reference Guide

Version 1.0 August 6, 2019

Abstract

This paper provides a technical overview of a Qwyit™ application: operating a QwyitKey™ *Authentication Service* – an invulnerable secret key authentication and encryption token management service. Applications would be able to simply place a small code segment within their communications protocol, allowing network participants the full benefit of perfectly secure Qwyit™ authenticated and encrypted message traffic without concern for 3rd party key management. This is the world's first participant-managed, independent-trust secure messaging capability.

Contents

QwyitKey™ Authentication Service

This document outlines the QwyitKey™ *Authentication Service* (QwyitKey™, QK™), the world's first participant-managed, independent-trust secure messaging capability. QwyitKey™ is an implementation capability based on the full Qwyit protocol as outlined in the current version of the *Qwyit Protocol Reference* document, available from Qwyit LLC. QK™ client demo APIs are available from Qwyit LLC; go to www.qwyit.com.

Introduction

The Qwyit™ protocol provides authentication (embedded) and data security (stream cipher) for communications traffic based on a secret-key system. Any Qwyit™ implementation can operate when the application/device provides a meaningful system key. The key is either a system level authentication token or simply a per-message encryption key. The key's origins and management can be of any type; including, but not limited to:

- Current PKI protocols (e.g., TLS) where the authentication is handled by PKI and the encryption is our QwyitCipher™ instead of AES
- Our own QwyitTalk™ 3rd party key management system which replaces PKI and TLS, providing authentication and encryption
- Self-managed systems, uniquely designed and created using any aspect of our Qwyit™ protocol
- This new QK™ Authentication Service, for applications needing a simple, fast, self-managed, independent-trust key distribution service for authentication and encryption – specifically, our QwyitChip™ hardware and accompanying identical SDK

Approach

The QK™ Authentication Service provides a method to allow any network application to authenticate and encrypt all messages between its participants. Please see the *QwyitKey™ Overview* for a complete explanation of the rationale for this simple system. QK™ requires a single round-trip anonymous registration, then minimally initial-only authentication requests for any participant's communication Receiver list. After that, participant-to-participant authentic, provably secure messaging can be performed for the lifetime of participant network membership.

For any network that wants to use the QK™ Authentication Service, their applications will:

- Ask the user to anonymously join QK™ by performing a simple, 1-click join
  - QK™ Authentication Service membership keys (called Authentication Service Keys (ASK)) are stored and managed by the network application
    - The ASK can be participant PIN-protected by the network application, as outlined in the following CKS section and/or other Qwyit™ docs
- For any network message destination (receiving participant) to whom/where the user hasn't initially communicated, a 1-click single round-trip Authentication Request (AR) is performed
  - This is a simple, well-known invulnerable way for network participants to remain anonymous for channel authentication during key creation and exchange, while they're known to each other (and uniquely authenticated) by the controlling application

- The trusted 3rd party QwyitKey™ has no network membership nor participant knowledge, and is only used to assure key distribution between the correct two messaging parties.
  - The trust is *only* that the messaging pair are provably authenticated to have established their key exchange with each other's device, setting up their own Qwyit™ VPN tunnel. Within the content of this secure context, and the controlling application performing this exchange, they will authenticate themselves as the expected participant end users
- The summarized process: the sending participant self-generates a Qwyit™ Session Master Key (SMK in two parts, SQK/SEK), specifically for communication with this intended Receiver. The Sender then submits a secure Authentication Request to the QK™ using their ASK that includes the Receiver OpenID and only ½ of the key, the SEK. Whereupon the QK™ replies to the sender in their ASK by including a receiver-only readable Authentication Bundle (AB), containing the SEK, in the Receiver's ASK. The sending participant then communicates provably securely with their intended Receiver participant with the first message including the AB; which when confirmed by the Receiver, creates their lifetime unique SMK (unless updated through another QK™ Authentication Request)

The QK™ Authentication Service:

1. Eliminates 3rd party key management complexity by actually mirroring how network traffic works – independent user and messaging authentication and security
2. Eliminates network software application complexity, reducing security development and programming to 2 simple, universal steps (Anonymous VSU, Authentication Request)
3. Takes full advantage of 'Anywhere/Everywhere Security' using Qwyit™, specifically offering a simple proliferation strategy for implementations of the QwyitChip™ hardware product, and the identical software QwyitSDK™
4. Solves the 'end-to-end security problem' by offering a simple, universal authentication and encryption method

QwyitKey™ Components

The following are required for QwyitKey™:

1. QK™ Authentication Server (private hosting, and/or publicly operated by Qwyit LLC at www.qwyit.com/Qwyit)
2. QK™-compatible client application (client w/QwyitKey™ embedded capability)

QwyitKey™ Core Required Processes

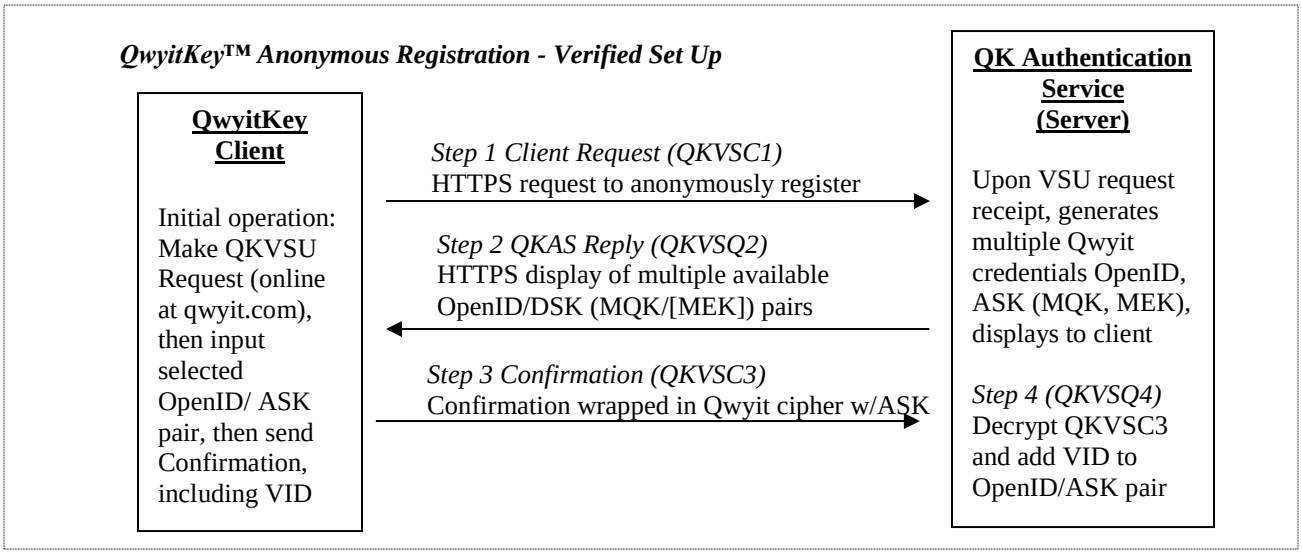
The following are the core required QwyitKey™ processes:

One-Time Anonymous Registration (QK™ Verified Setup – QKVSU)

- Initial QwyitKey™ client anonymous authentication token distribution is accomplished through a QK™ Verified Setup (QKVSU) with the QwyitKey™ Authentication Server (QAS). Token (ASK) is a 512-bit 2-part key, with a public identifier: OpenID [up to 64-bits], MQK [Master QwyitKey™ Key, 256-bits], MEK [Master Exchange Key, 256-bits])
  - This is to setup participation in any network wanting to establish simple participant messaging authentication and security

- This is *not* user/client registration, as in QwyitTalk™ or a PKI Certificate Authority
  - The QK™ service authenticates *the message*; the participant is authenticated within the network context and *content* (See QK™ Overview for distinction)

Architecture



NOTE: Steps 1 and 2 are online, HTTPS 'messaging', reliant on that security. Should it be deemed insufficient, there are other VSU methods, including our own QwyitTalk™ initial key distribution. The process should be risk/reward analyzed.

Details

NOTE: QK™ may use the TCP/UDP IANA reserved port for HTTPX, port number 4180 in a web architecture

NOTE: Following each step description, if there is a message sent, the format is shown in brackets: [MessageTitleID: parameter1, parameter2, … ended with a QK™ termination character (^)]

NOTE: As noted in previous Qwyit docs, each participant pair (QK™/Client, Client/Client) may perform a PDAF to create new keys at session end (either MQK, MEK after a QKVSU, or updating SQK, SEK after a TR), at the designated interval, etc.; which can be static, or known-random (i.e., from a selection of the uniquely embedded key content – this would be client-app configured.)

QK™ Verified Setup (QKVSU)

- Client, initiate QKVSU within the application (and as required/desired) with the Authentication Service/Server (*Step 1, QKVSC1*)
  - Go to www.Qwyit.com/Qwyit where an HTTPS session will begin. Select the 'QK Registration' link and/or button

- QK Authentication Service/Server reply to client (*Step 2 – QKVSQ2*)
  - QKAS server generates multiple available (unique) OpenID/Authentication Service Key (ASK) pairs and displays them to the registering anonymous client (QKVSQ2)
    - The MQK, MEK keys are each currently recommended to be 256-bits (64 hex digits)

- Client selects on OpenID/ASK pair, and copy/pastes it into their QK application; and sends an encrypted Confirmation message back to the QK notifying it of the selected pair (*Step 3, QKVSC3*)
  - Client will copy & paste the selected OpenID, ASK (possibly displayed in two sections, MQK, MEK) into their application, which stores the values (possibly, PIN-protected as noted in the CKS following section)

- Insert into use (store ASK, OpenID in cookie, file, db – method?)

  - Reply (QKVSC3) to DS with Confirmation message
    - Perform QwyitCipher™ encryption using selected ASK (MQK and MEK), generating message key (W)
    - Use message key to encrypt Confirmation message, which is a 64-bit, 16 hex digit Verified Identity Number (VID) created by using the last 16 digits (64-bits) of the MQK in a PDAF with the first 16 digits (64-bits) of the MEK
      - Perform a PDAF(MQK last 64-bits, MEK first 64-bits); result is Confirmation VID
      - Perform QwyitTalk™ encrypt using W and VID, result is VID encrypted (VIDe)
    - Send (QKVSC3) output (OpenID, OR, VIDe) to AS

[QKVSC3: OpenID, OR, VIDe]

- QKAS decrypt of confirmation message (*Step 4, QKVSQ4*)
  - Perform QwyitCipher™ decrypt using the ASK (found from the listed pairs by sent OpenID) and reveal the message key (W) – the QKAS will hold displayed pairs for a specified period of time for QK client apps to reply
    - Use message to decrypt Confirmation (by creating the correct same VID and comparing)
      - Perform a PDAF(MQK last 64-bits, MEK first 64-bits); result is Confirmation VID generated
      - Perform decrypt using W and VIDe, result is VID received decrypted (VIDd)
      - Compare VIDd received decrypted with the VID generated
        - If doesn't match, error sent
        - If match confirmed, store OpenID, ASK and VID into QKAS Authentication DB – method?

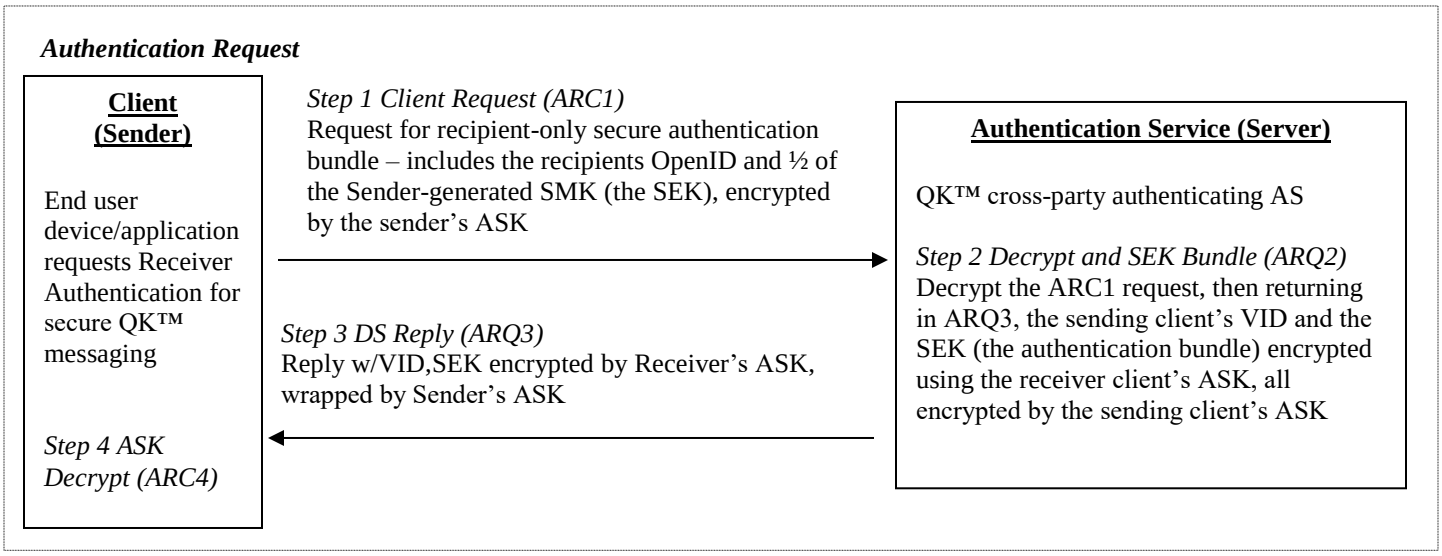
NOTE: Whether or not the VID remains 'unpublished', it doesn't leak any information about any of the keys; although the key space for creating it is small (only 64-bits). Should it be desired that the VID is a public, openly verifiable value, simply create and display it by using the entire PDAF key result (128-bits). In the QK™ system, the VID is the authentication token and is never publicly available (encrypted by the Sender, and the QK™). In the public case, it will still be the authentication token and sent openly; but it will allow additional nuisance attacks (accurate and simple substitution, etc.)

One-time, Initial Receiver Authentication Request (AR)

NOTE: The client will need to have the ability to generate PRNG/RNG digital data

NOTE: The following is for a P2P architecture, with 1-1 communication. Group key scenarios can easily be defined in the same way they were in Appendix C of the current *QwyitTalk™ Reference Guide*. See Qwyit.com for details.

- For each intended messaging Receiver, the sender will initiate a private real-time Authentication Request (AR) with a shared QK™ Authentication Server in order to create a QK™ VPN tunnel with the intended receiver (based on their OpenID). The AR will exchange a token that is a 256-bit Session Exchange Key (SEK). This is one half of the Sender-generated Session Master Key (SMK) pair in 2-parts: SQK [Session QT™ Key, 64 hex digits, 256-bits], SEK [Session Exchange Key, 64 hex digits, 256-bits])
  1. This is a simple, well-known invulnerable way for network participants to remain anonymous for channel authentication during key creation and exchange, while they're known to each other (and uniquely authenticated) by the controlling communication application
  2. The trusted 3rd party has no network membership nor participant knowledge, and is only used to assure key distribution between the correct two messaging parties.
     i. The trust is *only* that the messaging pair are provably authenticated to have established their key exchange with each other's device, setting up their own Qwyit™ VPN tunnel. Within the content of this secure context, and the controlling application performing this exchange, they will authenticate themselves as the expected participant end users.



Details

NOTE: The AR SEK is unique to each messaging pair of participants. The QKAS simply decrypts AR requests and sends AR replies, never storing any SEKs. Should there be a requirement to retain SEKs, the QKAS will need to be set to do so. Since the participants are anonymous, if the requirement arises to understand private conversations, after requiring the QKAS to store SEKs, *the entire message stream from the very beginning* to/from the anonymous participants would need to be captured in order to be read (requirement to outside agency);

Authentication Request (AR)

- Client Sender sends Authorization Request to the QK™ Authentication Service/Server (*Step 1, ARC1*)
    - Send Authorization Request (ARC1) to AS in order to receive the intended Receiver's authentication bundle
        - Randomly generate a Session Start Key (SSK) for the intended Receiver (128 hex digits, 512-bits) in two parts, QK, EK
            - Create the Session Master Key (SMK) that will be used for Qwyit™ provably secure messaging with this intended Receiver by performing SMK = PDAF(QK, EK) in two rounds such that SMK is 512-bits (with new SQK, SEK halves)
        - Send ARC1:SenderOpenID, OR1, CT1 where CT1 is encrypted in QwyitCipher™ using ASK to AS

  [ARC1: SenderOpenID, OR1, CT1] where CT1 is the encrypted ReceiverOpenID, EK

- Authentication Service/Server decrypts ARC1 (*Step 2, ARQ2*)
    - Decrypt ARC1 from Client using the received SenderOpenID and OR1 and their ASK in QwyitCipher™ revealing the ReceiverOpenID and EK
        - Check that Receiver client exists

- Authentication Service/Server reply to Client (*Step 3, ARQ3*)
    - Return sending client's VID and the EK back to the Sender, encrypted in a new QwyitCipher™ message using the Receiver client's ASK, all wrapped in a QC™ message using the Sender Clients' ASK
        - The ARQ3 is returned to the sending Client
            - If Receiver Client does not exist in AS:
                - If this AS is part of a federated group, check w/other AS members as per the QK™ communication key exchange framework of the group
                - Notify Sending Client of situation (if desired/required – no Receiver client)
        - The ARQ3 includes the Receiver client's authentication bundle in a QC™ message (including a new OR) that is encrypted using the Sender's OpenID in a QC™ message (with a separate OR)

- The Sender's VID and the EK are encrypted using the Receiver's ASK and a new OR3 [this is the Authentication Bundle (AB)]
- The ARQ3 is then a new QC™ encrypted message returned back to the Sender, using their ASK and a new OR2, where the CT is OR3, AB, creating eOR3, eVID1, eEK

[ARQ3: OpenID, OR2, CT2] where OpenID is the Sender Client's OpenID, OR2 is newly generated, and CT2 is [eOR3, eVID1, eEK] encrypted using the Sender's ASK

- Sender Client decrypts ARQ3 (*Step 4, AHC4*)
  - Sender Client decrypts ARQ3 from AS using their ASK in QC™
    - Reveals the Authentication Bundle, still encrypted by the Receiver's ASK, with the open OR3 (The VID, EK encrypted using the Receiver's ASK is now the remaining CT2)
    - Sender is now prepared to authenticate and share their self-generated key with their Receiver
      - Sender and Receiver Clients commence QK™ Messaging with Sender initiating contact with QwyitKey™ Start message (including OR3, CT2 for Receiver to authenticate the sender by the VID, and to derive and share the SMK)

- Authentication Service/Server and Sender Client perform a PDAF PFS NIL communication ASK key update
  - 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

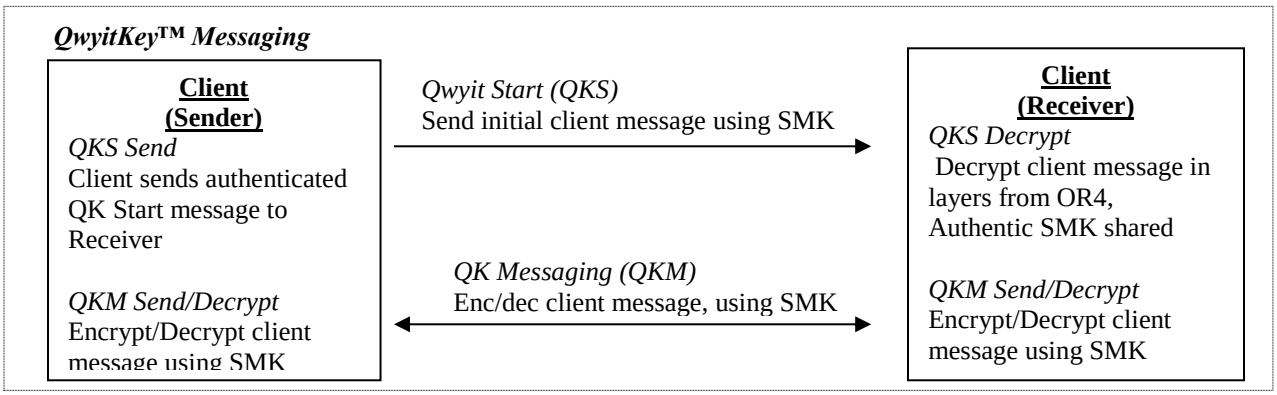
NOTE: Keeping the keys in synch between the ADS and the Clients needs careful attention. Each different QwyitKey™ network may require unique handling.

QK™ Messaging

- QK™ Messaging, each pair participant using their own generated SMKs in secure QwyitCipher™ exchanges

(The SMK (SQK, SEK) must be handled and stored securely.)

Architecture



Details

QK™ Messaging

- Sending Client sends QwyitKey™ Start to Client Receiver (*QwyitKey™ Start, QKQS*)
  - Using the AR self-generated SMK created for this Receiver:
    - Perform MOD16(QK,EK); result is OR4
    - Create new message content (IMSG, the Initial Message opening communication)

- Perform a QC™ on Sender's VID, IMSG using OR4 and the self-generated, QK/EK calculated SMK (SQK,SEK), the result is CT5 (eVID2, eISMG)

- Send QwyitKey™ Start (QKS) message
  - Send QKS: OR4, CT4
    - Where CT4 is the Authentication Bundle (OR3, CT2), and CT5

[QKS: OR4, CT4] where CT4 is OR3, eVID1, eEK, eVID2, eISMG

- Receiving Client decrypts QKS (*QK™ Messaging, QKM*)
  - Receiver Client, decrypts in layers:
    - 1st decryption: Use ASK and OR3 to decrypt the Authentication Bundle, revealing VID1 (sent from the QKAS) and EK.
    - 2nd decryption: Perform a MOD16D(OR4, EK) to reveal QK
    - Perform PDAF[QK,EK] to reveal SMK (SQK,SEK)
    - 3rd decryption: Use OR4, SMK to decrypt VID2 (sent from the Sending Client) and ISMG
    - Perform the Authentication Check, comparing VID1 to VID2
      - If they match, then both clients have a shared SMK – as long-lived as deemed appropriate for the application network
        - Operate on the IMSG content and continue QK™ Messaging (sending QKMs) using the shared SMK in QwyitCipher™ provably secured messages
      - If they don't match, send error

- At known intervals, or 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

Error Messaging (DSE)

- Accompany all of the above w/appropriate error trapping and notifications

[ASE: MessageID, appropriate contents]

QwyitKey™ Optional Processes

The following are optional system-defined QwyitKey™ processes:

Client Application Key Storage (CKS)

- Turn on Pin-protected Client Key Storage (CKS)

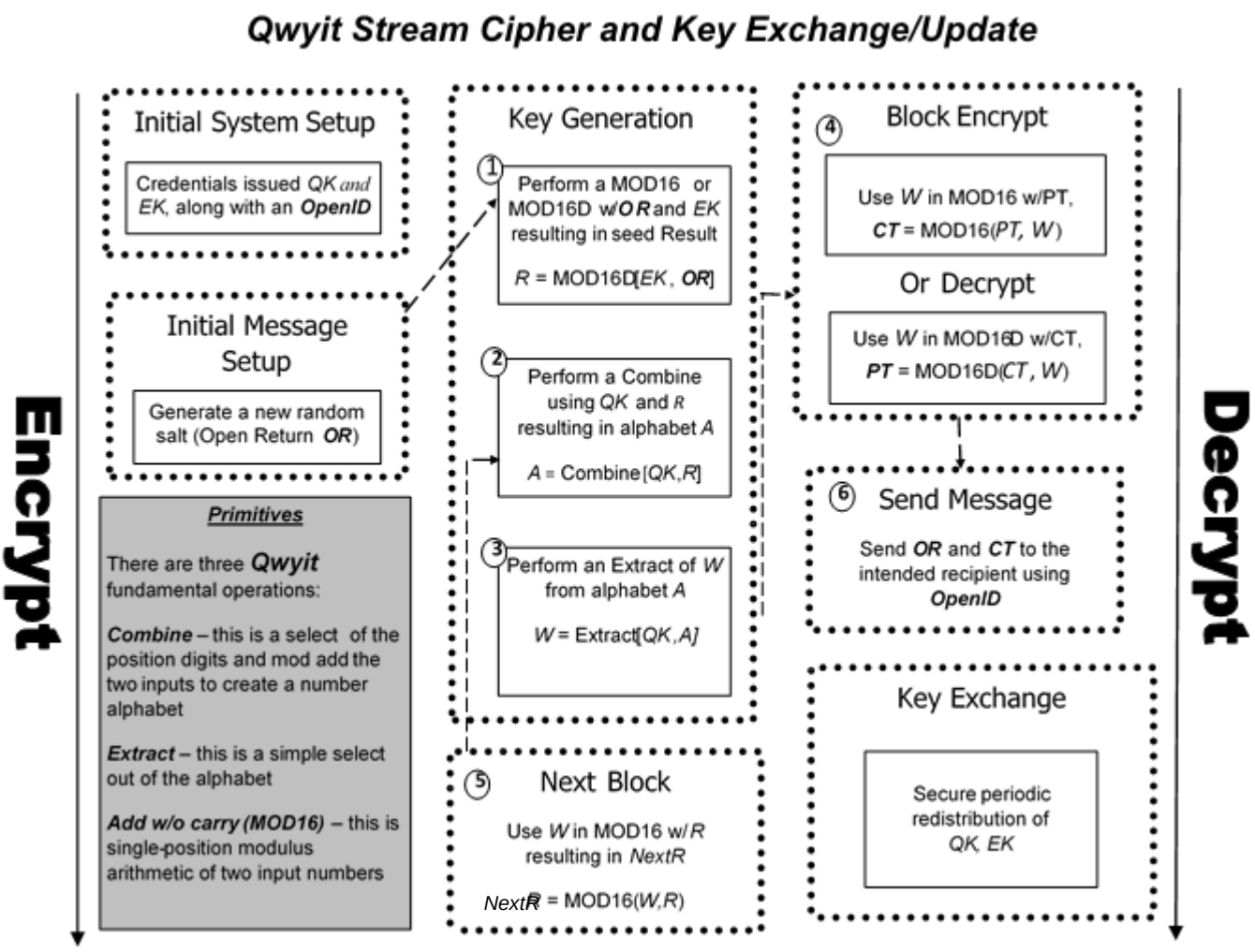
  Should there be a requirement/desire to tie the local version of the QKVSU ASK to the individual user, as opposed to just the device (or if there's a desire for a simple, effective encrypted storage technique in addition to the many already available to the client application through the OS, etc.), it is recommended that an n-digit hex PIN be used to encrypt ASK local storage with simple MOD16 addition. The desired security indicates the number of digits, and the PIN must never be stored anywhere on the device. This local protection maybe also be accompanied by limiting the number of wrong use attempts locally, after notification from the AS of incorrect key use (online, in HTTPS).

Anonymous Registration Options (QKVSU)

- Pre-Shared
  a. Chip manufacturers provide QK™ lists of OpenID/QK Keys (in two 256-bit parts, MEK/MQK). In this case, the User's device would begin network participation by sending the *QKVSC3* Confirmation message, including the self-generated VID.
  b. Security is based on secure submission from/to the manufacturer and QK™
     i. **NOTE:** As of this document's date, this is the only Pre-shared supported method

c. Other Pre-Shared methods are possible, such as the Device self-generates their own OpenID/ASK and submits them to the QK™ using some channel
    i. Channel examples such as phone, email, snail mail could include an smaller-than-QKK sized offset, since copying/submitting 512-bit QKKs could be error-prone
    ii. TLS-based web submission can include cutting/pasting from the generating App into a QK™ submission web page
    iii. Security is based on the submission type (TLS, snail mail, phone, email, etc.)
    iv. Requires an additional round-trip Accepted/Denied QK™ Reply message to the Confirmation message that first, the OpenID is unique and second, to test that the keys are correctly understood (since this is an anonymous submission). The QK™ would perform a PDAF(MEK/MQK) and send the 256-bit result, encrypted by the shared ASK, to the Participant to check. If the Participant's result of their PDAF doesn't match, a VSU Cancel message would be sent to the QK™ to remove the OpenID/ASK pairing, and the Participant would try again.
        1. **NOTE:** As of this document's date, these additional Pre-shared methods are *unsupported*.

Appendix A – QwyitCipher™ Stream Cipher



Send

- Generate random salt Open Return [OR 256-bits])
  (This step does not require an 'unbreakable' or 'secure' (P)RNG process since the value is public; but it should meet the base requirements for uniqueness and randomness since it is the QwyitTalk™ basis)

1. Perform MOD16 using EK and OR, resulting in R [256-bits]

2. Perform COMBINE using QK and R, resulting in alphabet A [256-bits]

3. Perform EXRACT using QK and A, resulting in message key W [256-bits]

4. Perform MOD16 with W and up to 256-bits of the Plaintext (PT), resulting in Ciphertext (CT)
   a. For performance, with key and PT identical lengths, this may be an XOR of W and PT resulting in CT

5. Perform MOD16 using W and R, resulting in the NextR [256-bits]

   1. Perform 1 (using NextR as OR), 2, 3, 4 and 5 iteratively until the end of Plaintext (256-bits at a time), concatenating total C (There is no need to pad the plaintext)

      - Optionally (continual/random/regular), update EK (and/or QK) in MOD16's with last R or W
        Any 'update series' (with flag values, error checking and confirmations) is system-specific.

6. Send OpenID, OR and C to Receiver

Receive

- Perform MOD16 using EK and OR, resulting in R [256-bits]
- Perform COMBINE using QK and R, resulting in alphabet A [256-bits]
- Perform EXRACT using QK and A, resulting in message key W [256-bits]
- Perform MOD16D using 256-bits of Ciphertext (C) and W, resulting in Plaintext (P) [or CT XOR W = PT]
- Perform MOD16 using W and R, resulting in the NextR [256-bits]
    - Perform 1 (using NextR as OR), 2, 3, 4 and 5 iteratively until the end of Ciphertext (256-bits at a time), concatenating total P
    - Optionally (continual/random/regular), update EK (and/or QK) in MOD16's with last R or W