Qwyit Data Storage and Access Control Service (QwyitStore™)
Storage As A Service

Reference Guide

Version 1.0 April, 2018

Abstract

This paper provides a technical overview of a Qwyit™ application: operating a QwyitStore™ *Storage as a Service* – an unbreakable data storage and identifiable access and use control service. Entities would be able to provide complete protection and access control management for their customer data. This solves the current constant gaping security hole between customer data provision, and commercial data storage.

Contents

QwyitStore™ Data Storage and Access Control Service

This document outlines the QwyitStore™ Data Storage and Access Control *Service* (QwyitStore™, QS™), a complete privacy and security solution for electronic data at rest. The service provides a heretofore nonexistent capability for private communication participants to offer personal data and maintain unbreakable control over it while stored by participating parties. Complete control over the stored data's access is also provided. Participation in the QwyitStore™ service 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. QS™ client demo APIs are available from Qwyit LLC; go to www.qwyit.com.

Approach

QwyitStore™ easily allows any entity to provide complete protection and access control management for their customer data. The demonstration includes a QS™ Directory Server application that can be run on, or added to, any current communications server (web, file, comms, app, DB, etc.), as well as example client code for easy insertion into the data record creation, storage and access of a connected device/app.

QwyitStore™ Components

The following are required for QwyitStore™:

1. QwyitTalk™ Server (operated by Qwyit LLC at www.qwyit.com/Qwyit)
2. QwyitStore™ Server (operated by Qwyit LLC at www.qwyit.com/Qwyit)
3. QT™/QS™-compatible client application (client w/QwyitTalk™ and QwyitStore™ SDK embedded capability)

As a prerequisite to any client application executing the QS™ processes to encrypt, store and control their data records, they must have joined the QwyitTalk™ community by registering and performing a VSU. The entity operating a QS™ Store for their customers will have joined as well.

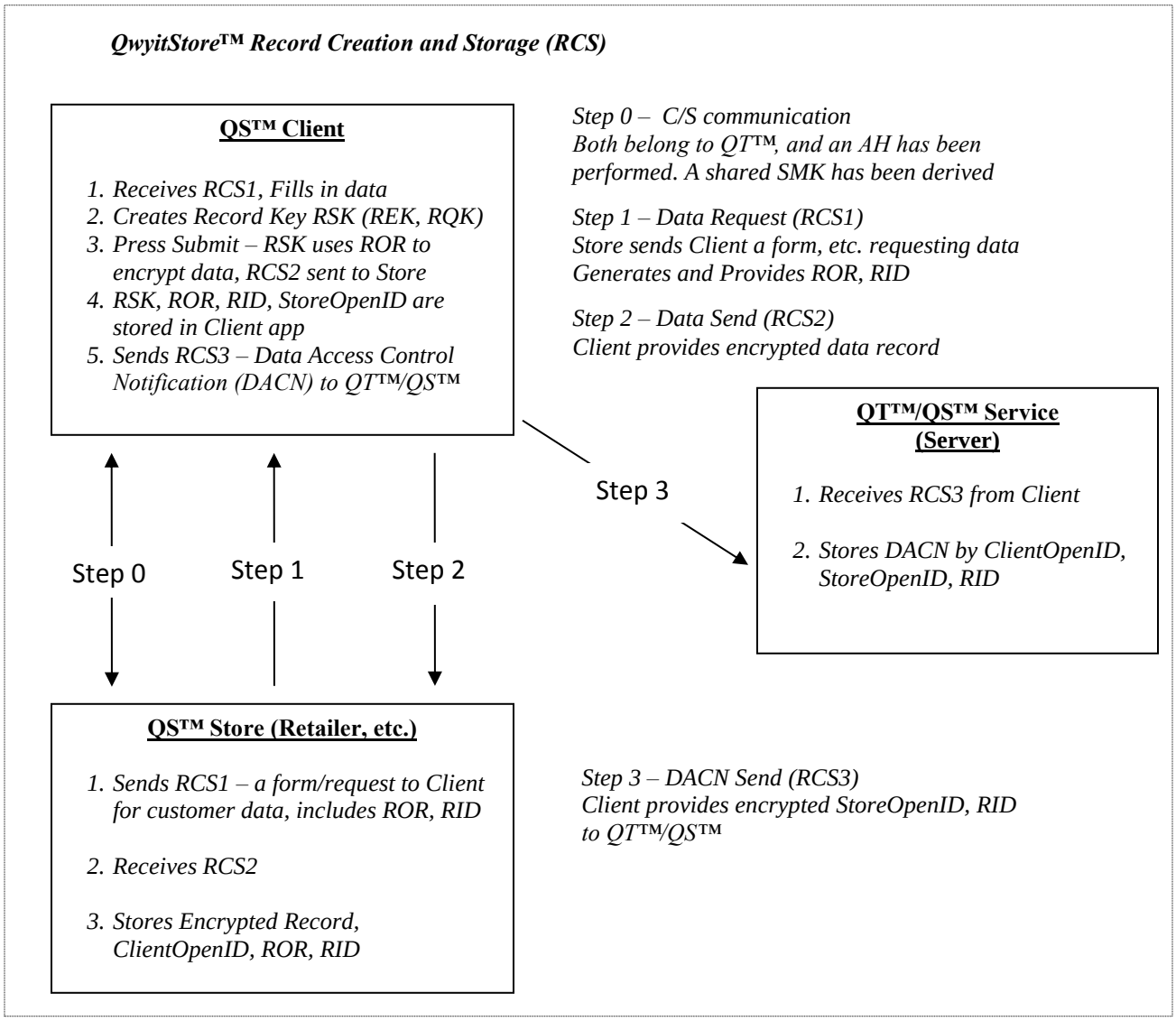
QwyitStore™ Core Required Processes

The following are the core required QwyitStore™ processes:

Record Creation and Storage (RCS)

- Initial QwyitStore™ data record creation and encrypted storage process. A QS™ Store will present their data collection (app, form, some method for the presentation of the required and/or optional customer data to be collected) to the Customer (Client), who will provide the data, encrypt it and return it. The Client will then notify the public QS™ Service/Server of the transaction through a Data Access Control Notification (DACN).

Architecture



Details

Record Creation and Storage (RCS)

- Client and Store have both become members of the QwyitTalk™ secure community (either with/through another app or this QS™ application after discovery that no QT™ keys are in storage (DSKs)
- Client and Store are in secure QT™ communication after having performed an AH, and a shared SMK has been derived for their session (*Step 0, RCS0*)
    - All communication is unbreakably communicated using the current SMK in QTQT messages – the below indicates the encrypted message contents

- Store initiates the RCS process, sending a data collection mechanism (form, request, etc. – some means within this QS™ app), along with an ROR, RID for this data record (*Step 1, RCS1*)
    - Generate ROR (64 hex digits, 256-bits)
    - Generate RID (64 hex digits, 256-bits)
    - Send RCS1 to Client

  [RCS1: StoreOpenID, OR, CT where CT is the ROR, RID, data form] – may take several QTQT messages

- Client receives RCS1
  - Fills in the requested data
  - Creates Record Key, RSK (REK, RQK)
    - Generates RSK, in two parts
      - REK (64 hex digits, 256-bits)
      - RQK (64 hex digits, 256-bits)
  - Press Submit/Send – App will perform a Qwyit™ encrypt on the form data using the RSK and ROR
  - Sends the RCS2 back to the Store (*Step 2, RCS2*)
    - This data creation is stored in the Client app (NOT the actual data, but a record of its creation and encryption) by RSK (REK/RQK), ROR, RID, StoreOpenID

  [RCS2: ClientOpenID, OR, CT where CT is the encrypted data record] – may take several QTQT messages]

- Client sends RCS3 – Data Access Control Notification (DACN) to QT™/QS™ (*Step 3, RCS3*)
  - In order to maintain decryption control, Client sends a DACN to the public QT™/QS™
  - As in a QT™ Authentication Handshake, Client generates an OR, and using their QT™ DSK, sends RCS3
    - Generate OR (64 hex digits, 256-bits)

  [RCS3: ClientOpenID, OR, CT where CT is the ClientOpenID, StoreOpenID, RID of the RCS2 message]

- Store receives RCS2
  - Decrypts the QTQT message, revealing the Encrypted Record as the PT
  - Stores the Encrypted Record, along with its ClientOpenID, ROR, RID

- QT™/QS™ receives DACN RCS3
  - Decrypts the DACN using the Client's OpenID and their QT™ DSK, revealing StoreOpenID and RID
  - Stores these in the QS™ DACN record database

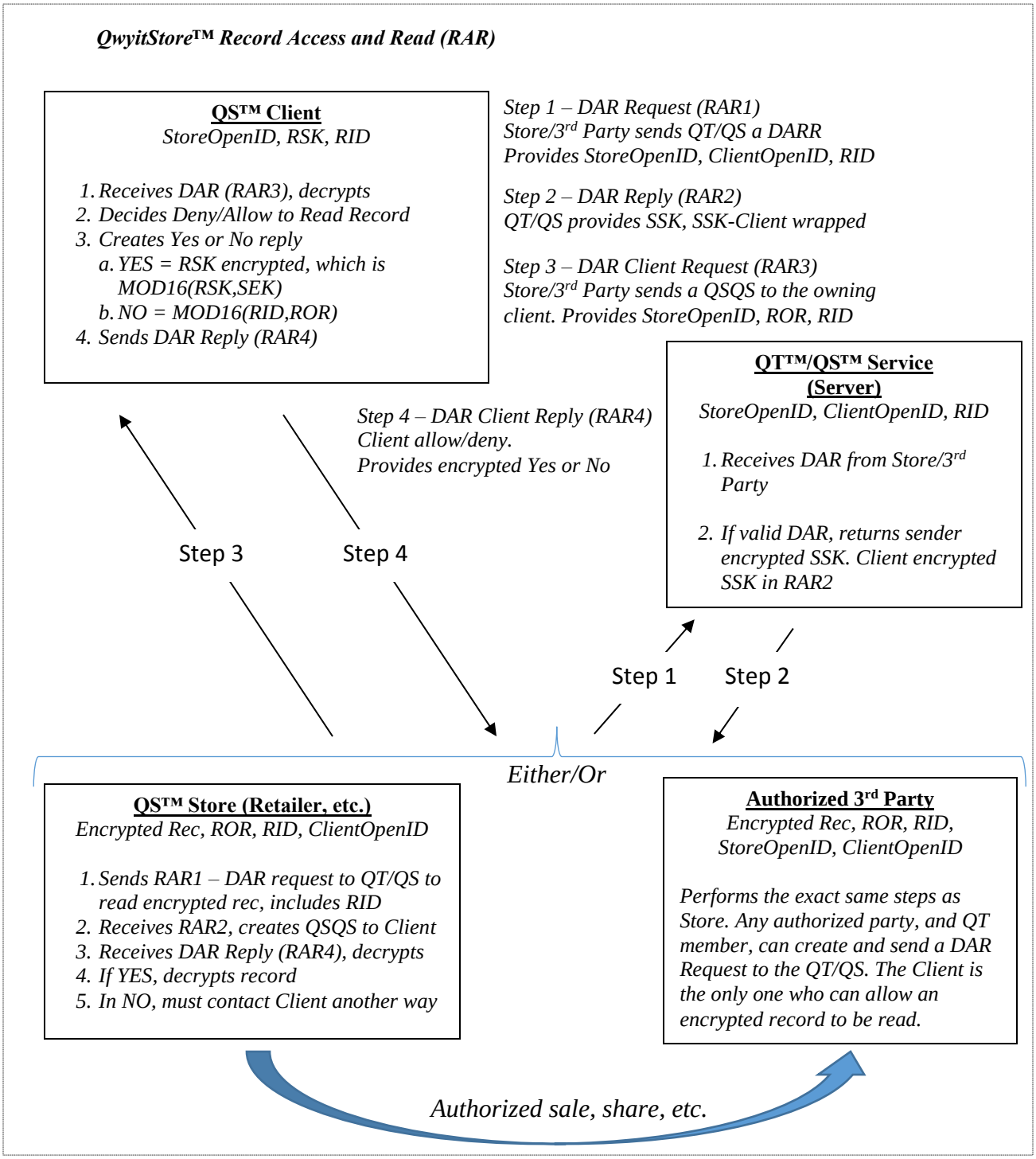
- QT™/QS™ and Client perform a PDAF PFS NIL communication DSK key update
    - Perform a PDAF(MQK 256-bits, MEK 256-bits) for two rounds, creating 512-bit result w/MQK, MEK halves
      - Optionally, extend the above PDAF for 2 rounds (doubling length) and perform an OWC

NOTE: In order to ascertain and assimilate any QS™ data record, one must steal the encrypted record from the Store, along with its ROR (which resides in both the Store and the Client) and its RSK (which resides only with the Client). The QT™/QS™ public DACN records do *not* contain any information, other than the owners of the info and keys – which is of little value since Stores are already known to have encrypted QS™ records. One must steal information from the Store, and then each individual Client. This meets the goal of defeating massive data collection losses with a single attack at a single location.

Record Access and Read (RAR)

- Recursive QwyitStore™ data access and control process. A QS™ Store, or authorized 3rd Party, will present a Data Access Read (DAR) Request to the QwyitStore™. This DAR will be validated at the QT™/QS™ by RID, and a session key created for the Store and the associated record owning Client. QT™/QS™ will send a DAR Reply. The Store/3rd Party will perform a QTQS/QSQT message sequence with the Client, who will allow/deny the DAR directly with the requestor.

Architecture



Details

Record Access and Read (RAR)

- Store may have any number of Authorized 3rd Parties who have received copies (or have the only and original version) of the Encrypted Client Record; additionally, they have the StoreOpenID, ClientOpenID, RID and the ROR
- The Steps 1 and 3 may originate from either the Store or any 3rd Party; Steps 2 and 4 replies will be sent to originator
    - All participants *must be* QwyitTalk™ members, and have DSKs
    - All communication is unbreakably communicated using the current SMK in QTQT messages – the below indicates the encrypted message contents

- Store initiates the RAR process, sending a Data Access Read (DAR) request to the QT™/QS™ including the StoreOpenID, ClientOpenID and the RID for the to-be-read encrypted data record (*Step 1, RAR1*)
  [RAR1: StoreOpenID (or 3rdPartyOpenID), OR, CT where CT is the StoreOpenID, ClientOpenID, RID]

- QT™/QS™ decrypts the RAR1, and performs a lookup of the data elements. Upon confirmation, replies with a DAR reply in RAR2. (*Step 2, RAR2*)
    - Generate a SSK for the Store/3rd Party and Client to request/deliver record reads (64 hex digits, 256-bits)
        - Send SSK (RAR2) to Sender, with wrapped Record-owning Clients' msg
            - The RAR2 includes 2 different CTs
                - The SSK is encrypted using the Sender's DSK and the OR
                - The SSK is twice-encrypted using the Record Client's DSK and the same OR

  [RAR2: StoreOpenID (or 3rdPartyOpenID), OR, CT1, CT2 where CT1 is the SSK-encrypted ciphertext using the Sender's DSK and CT2 is the twice-encrypted SSK ciphertext using the Client's DSK

- Store/3rd Party decrypts the RAR2, prepares a DAR Client Request (RAR3) to send to the record-owning Client. This is a QTQS normal messaging sequence. The Store/3rd Party creates a QR, and sends a two CT Qwyit Start message to the Client. (*Step 3, RAR3*)
  [RAR3 (QTQS): OpenID, QR, OR1, OR2, CT2, Ciphertext of IMSG where OR2 is the RAR2 OR, and CT2 is the twice encrypted SSK and IMSG ciphertext is the StoreOpenID, ROR, RID]

- Record-owning Client receives and decrypts RAR3, then prepares a DAR Client Reply (RAR4) to send to the requesting party (Store or 3rd Party). The Client QS application will handle how to reply, allowing Client to pre-authorize to the original Store, be alerted when it is an unknown 3rd Party (whom they then can 'save' as authorized), etc. Upon the decision, either a Yes or No DAR Client Reply will be sent (*Step 4, RAR4*)
    - If YES, reply w/RSK encrypted
        - RSK encrypted = MOD16(RSK, SEK)
    - If NO, reply w/RID encrypted
        - RID encrypted = MOD16(RID, SEK)

  [RAR4 (QSQT): OpenID, OR, CT where CT is either the YES or NO contents encrypted]

- Store/3rd Party decrypts the RAR4, and determines whether a YES or NO (perform MOD16Ds, and if the RID is NO, if not, it is the RSK)
    - If YES, use the decrypted RSK and ROR to decrypt the Client record
    - If NO, must contact the Client in some other fashion to get authorization to read the record

NOTE: Client QS™ apps may add functionality to 'auto' reply to Step 3 DAR Store requests; and the ability to configure such replies (either Yes and/or No). Additionally, local storage (Client and/or Store/3rd Party) may well be augmented with PIN and other types of offset/dictionary attack security, as noted in QwyitTalk™ and other Qwyit® documents.

NOTE: Please see the *QwyitTalk™ Reference Guide* for further information on messaging security (hence record storage security), the Qwyit cipher formation and use, group messaging (hence, group record storage and shared SSKs/RSKs), etc.