Qwyit Universal Authentication and Data Encryption [FPGA (QwyitChip™) and S/W (QwyitSDK™)]

Reference Guide

Version 1.0 August, 2019

Abstract

This paper provides a technical overview of Qwyit™ authentication and data encryption as formulated on an FPGA for hardware application – QwyitChip™, and identically in software as an SDK - QwyitSDK™. Any device, and any application, would be able to incorporate this into their S/W, H/W designs, providing universal, simple and fast authentication and data encryption services. This solves the main issue with current communication networks: security complexity leading to inherent implementation problems, including the complete lack of use and worst of all, the inability to use.

Contents

QwyitChip™/QwyitSDK™ Universal Authentication and Data Encryption

This document outlines the QwyitChip™ (QChip™) and QwyitSDK™ (QSDK™) Universal Authentication and Data Encryption hardware/software solution. The design provides a simple and fast method for any device and/or application running on the device to authenticate and encrypt all communications. One of the main benefits over any existing hardware encryption chips is that simple, fast, mutual authentication is now available, instead of having to rely on complex, slow and intermittent software methods that are only session based, not per transmission. Having a universal, works-the-same-everywhere capability for both software and hardware makes universal security a reality.

Additionally, the world's smallest, fastest and provably secure encryption can be performed entirely within the hardware, instead of current acceleration-only chipsets of weak and not provably secure algorithms, albeit 'standards-based' such as AES. The QCHIP™ design 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. QCHIP™ client demo FPGAs are available from Qwyit LLC; go to www.qwyit.com.

Approach

There are serious deficits in network security:

- Authentication, mutual and per transmission, simply does not exist
- Encryption, on-the-fly and within network communication tolerances, is too complex

These security deficits cause fundamental problems

- The entire network participant spectrum can't enact proper end-to-end security
  - Users don't know what to do
  - Developers don't understand all the complexities of current protocols
  - Operators can't follow all the rules and procedures
  - Owners have no reliable means to upgrade and future protect their investments

These fundamental problems reveal the truth about current network security

- The methods in use today aren't capable, no matter how they are presented
  - Too slow, too complicated, too large, permanently inefficient, factually insecure

In the words of an old ad slogan: "There has to be a better way!" And there is:

1. Create a method with the proper features: Fast, small, efficient, flexible, provably secure
2. Create, build and proliferate the fundamental infrastructure components to deliver the method
3. Detail the simple instructions for use

The Qwyit method has the proper features, it is available in software QwyitSDK™ and now the hardware QwyitChip™, and this document outlines the straightforward use.

In order for QwyitChip™/QwyitSDK™ to be fully utilized across the widest possible spectrum of applications, it is being made available as a simple, fully capable, provably secure, world's smallest

and fastest encryption chip operable with a call and the current key; and identically available as an SDK for those places where software can be relied on to securely deliver. While the entirety of the Qwyit protocol (the initial Verified Setup (VSU) key creation and distribution, the per-session Authentication Handshake (AH), and the per-transmission Normal Trusted Operation (QwyitTalk™) could easily be enacted in hardware (forthcoming as the full protocol QwyitChip2.0™), the initial approach is to provide the QwyitCipher™ in a single operating mode:

- Qsec(PlainText/CipherText, OR, Key) – call the chip or SDK to operate on the included PT/CT using the current Open Return (OR) and session key (SMK in two parts, SQK, SEK for Qwyit-enabled applications. For other applications, any n-bit key where it will be used in two halves (minimum 256-bits recommended).)

This simplicity allows an application to call Qwyit using any of the key-creation/obtainment methods currently available; where the streamed encryption/decryption will provide mutual, per-transmission authentication, and on-the-fly key-to-key provable security. When used in concert w/Qwyit-enabled applications, key-to-key will be end-to-end; when inserted into other, existing methods, the scope of key-to-key is determined by the calling application(s) and their inherent key management.

QwyitChip™/QwyitSDK™ Component

The QwyitChip™ is an FPGA designed to provide Qwyit encryption/decryption – this FPGA Verilog coded design can easily be implemented on even the smallest, low-powered devices.

The QwyitSDK™ is a SDK identically designed as the QChip™, to provide Qwyit encryption/decryption.

As a prerequisite to a client application executing QwyitTalk™ or QwyitKey™, the app will perform and manage the Qwyit keys in accordance with the current QT™/QK™ processes for VSU, AH/AR, QT, and update. For any other applications, QCHIP™/QSDK™ key management is the responsibility of those processes.

QwyitChip™/QwyitSDK™ Encryption/Decryption Process

The following is the core QwyitChip™/QwyitSDK™ encryption/decryption process:

QwyitChip™/QwyitSDK™ Encrypt/Decrypt (Qsec)

- Call Qsec to encrypt a plaintext message, or to decrypt a ciphertext message, using the provided Open Return and Message Key

Details

QwyitChip™/QwyitSDK™ Encrypt/Decrypt (Qsec)

- Client app, whether QwyitTalk™/QwyitKey™-enabled or other, has obtained the current Message Key and latest Open Return (OR)
    - Both values should be either the direct result of a suitable (P)RNG (the OR), or the result of a process based on a suitable (P)RNG (the Message Key)
- Client makes call to the QwyitChip™/QwyitSDK™, sending in either the Plaintext to be encrypted, or Ciphertext to be decrypted, along with the obtained Message Key and OR
    - In the QwyitTalk™/QK™ protocol document, this is the CT creation, or PT decryption, of the QTQS and QTQT Normal Trusted Operation
    - In other applications, this is the CT creation and PT decryption as per the communications process

- The call is identical whether an encryption or decryption, and the chip processing is the same – the calling application handles the result dependent on whether is a to-be-sent CT, or to-be-processed PT

Qsec(CT/PT, OR, 1st-half-MK, 2nd-half-MK) where CT is the ciphertext to be decrypted, PT is the plaintext to be processed, and MK is the current Message Key (SEK,SQK in QwyitTalk™/QK™ enabled apps)

- Client app processes the result

Code Execution

The following are the QwyitCipher™ code steps performed on the QwyitChip™/QwyitSDK™:

1. Generate first R value: Perform a MOD16 on 1st-Half-MK (SEK), and the Open Return (OR)
   R = MOD16(SEK, OR)

2. Combine R and 2nd-Half-MK (SQK), resulting in a n-bit alphabet, A
   A = Combine(R, SQK)

3. Create 1st W: Extract n-bit key W out of A using 2nd-Half-MK (SQK)
   W = Extract(A, SQK)

4. Encrypt/Decrypt the provided message using W

   For every n-bit of message (either PT or CT) that equals the length of W,
   Result = W Xor MessagePart

   IF message's total length is greater than W's length, for every next n-bits that equals the length of W, update W by
   A. **NextR = MOD16(W, R)** where W and R are their last values (e.g., the updated values, not the original. This is W and R for the 1st new W, then they are NewW and NextR successively)
   B. **A = Combine(NextR, SQK)**
   C. **NewW = Extract(A, SQK)**

   End If

   Then use NewW in an XOR on the next n-bits of the message; continuing to update NewW for every W's n-bits in length

   Example, if the message is 512-bits, and the MK is 256-bits, where SEK is 128-bits and SQK is 128-bits, then each W and successive NewW will be 128-bits, and four (4) will be created to encrypt/decrypt the full 512-bit message.