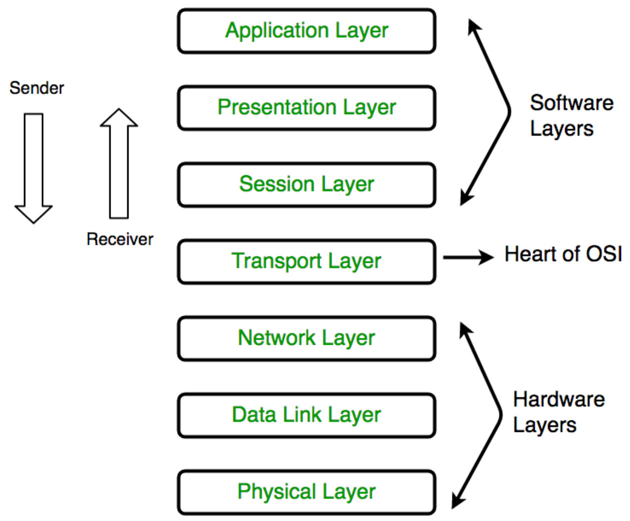
QwyitChip: Prototype Overview

Introduction

There are two basic processes that occur in any ‘security paradigm’ regardless of whether this involves transmission or storage of important data: authentication and encryption. The first is mostly ignored, or ‘summarized’ into the opening of any data session; the second is ‘standardized’ into the use of an ‘approved’ algorithm.

Currently (5/2019), this is done almost exclusively in the software layers of the OSI (mostly the application layer):



The majority of the problems and insecurity of this approach, is that the methods used are overly complex. Aside from their inherent process issues, performing the basic tasks at this level opens a Pandora’s Box of available attack vectors. There are so many, that the real results are that the systems have very little value other than perceived security: attacks happen every day, at unprecedented levels.

By 2030, each person will own at least 15 IoT devices…where there are currently over 30+ different ‘security protocols’ incompetently running on them. This is an *inherent lose-lose* situation: it’ll never be the answer.

A particularly alarming outcome of this overall problem, is that the methods don’t encourage – or even allow – one of the simplest solutions to pervasive insecurity: embed it in the hardware so that *everything* transported/stored is already *secure*. ***If there was a simple method embedded into the hardware, that provided mutual authentication and perfect encryption, any and every software application would be secured with the same type of simple call to ‘Transport (Store)’:***

Secure[Data, Key] = Ciphertext
Transport[Ciphertext]

The *only* required ‘security paradigm’ function for the Software Layers is to manage where/when/how to get the Key (Key management). This would be directly related to the Authentication Model of the network/device:

Fundamental Key provisioning methods/Models – *based on the Authentication (Trust) Model*

a. Central, federated key depot (such as QwyitTalk)
b. Network Owner – Iot, stand-alone device (car, Google Home/Amazon Alexa/etc (Smart Network Control devices), etc.)
c. Participant – individual connectivity, Asymmetric Public Key generated
d. Manual – loaded as per Network membership (by owner, operator, manufacturer, etc.) – This can be called Static also – where it's already a set, non-changing key, looked up by whatever devices will communicate with it

Such a new model of Authenticated Secure Data easily delivered inherently within the hardware of the communication/storage devices will drastically, dramatically eliminate the majority of the current network security problems.

Issues

- The current methods can't be used. Since they have been around for over 50 years – Authentication is mostly a Public Key exchange (Diffie-Hellman, ECC, etc.), and encryption is the 'standard' AES algorithm – this is apparent by the complete lack of standardized hardware encryption chips using these methods. It has to be assumed that Hardware manufacturers simply can't make these methods work inherently, since the majority of what they offer are AES-accelerators; not inherent 'encryption chips'. These accelerators 'help' the application software 'do encryption faster'. This is the only advancement in hardware security for the last 50 years.
    - If this assumption is wrong, and it *is* possible to cheaply, efficiently offer 'a complete security solution' in HW using the current methods, then at a minimum, these manufacturers need to become aware of order of magnitude improvement in 'cheap and efficient' such that they'll take up the gauntlet of improving security by offering embedded solutions.

- Until new methods are 'approved', HW manufacturers simply won't investigate, create and offer embedded security

- Regardless of 'approval' (by whom, for what platforms, etc.?) New methods must be demonstrable, not just theorized [This problem is 'solved' by providing a demonstrable solution]

- What comes first, the product or the customer? Currently, since there aren't any HW solutions, there isn't any 'demand' nor capability. [This problem is 'solved' by providing that demonstrable solution to a large-scale customer with immediate benefit, and partnering to deliver the product]

The Market Objective

The QwyitChip (QC) method provides the exact same authentication and encryption used in SW today. The method is available, 3rd-party verified, and superior to current techniques. The *main benefits* of the QC method are *speed and efficiency (size, processing, etc.)*. ***These are the main (sometimes only) benefits to any HW improvement touted over the last 50 years in computing.***

A demonstrable QC in hardware aligns the search for an answer to today's security paradigm failures with manufacturers' main next-generation chip designs benefits.

It remains to be seen whether this perfect alignment of Solution to Product goals will surmount manufacturers reticence over the mystical 'approved', and non-existent customer demand. If the current state of affairs in the Financial payment processing industry is any indication, where they are trying just about any and every new way to move money around the marketplace regardless of ‘standards’, now is the time to introduce a Perfect Encryption Chip.

The Technology Objective

The goal is to provide a framework QC 1st version prototype that is:

- Demonstrable
    - Performs the QC code as outlined in the QC Tech Ref doc in an HDL (Verilog or VHDL)
- Testable
    - Can be verified and tested by manufacturers that it does what it says and shows the potential
        - First iteration does not need to meet/benchmark any specifics, but make them apparent
- Presentable
    - It is difficult to ‘show security’ – but we can show how it works. However this is built, it must be portable and easily shown (and delivered for independent verification)

Future versions will formalize our approach, and include actual benchmarks, comparisons, etc.

First version prototype should be able to accept a plaintext ‘message’ (up to 10MB at least) and a two-halves key (512-bits in total, each half 256-bits) as inputs and return the corresponding exact length ciphertext.

The ciphertext and key should be able to be input, and return the corresponding correct original plaintext.

The operating code will perform the QC simple XOR of the PT and Key, update the key every 256-bits as per the MOD updates in the ref manual, and continue XORing until complete. The best, optimized, expected design architecture that accomplishes our Qwyit mission of speed and efficiency should be used (gate size, etc.) E.g., whatever any HW manufacturer would expect to see as a demonstrable/testable presentation (a simulated version, an actual FPGA design and built, Verilog code, etc.) should be used.

The Presentation Objective

At any QC introductory meeting, we will present a simple, quick presentation deck:

- Overview of the Qwyit protocol – The above: Current problem w/security, what’s needed (simple, perfect), our features and benefits
- Why a Qwyit SW solution is too horizontal (20+ years of “it works for you, and you, and you…”, but a HW introduction is focused and a total solution
- Overview of the QwyitChip , features and benefits (speed and size!)
- How QC meets HW manufacturers New Chip goals: speed and efficiency…WHILE solving The Security Problem! [And…if possible, an identified partner is waiting in the wings…]

Then the QC demonstration

Discussion