Quick Start Guide

SecureRF Security Tools for the Arrow Chameleon96 Community Board

SECURE RF

Securing the Internet of Things®

Introduction

SecureRF® Corporation provides fast, quantum-resistant security tools powering the Internet of Things (IoT). The company's asymmetric (public-key) solutions enable companies to deploy small, fast and ultra-low-energy solutions providing integrity, authentication and non-repudiation for IoT devices.

SecureRF provides a Chameleon96 FPGA board image that includes the WalnutDSA™ Digital Signature Verification Algorithm and Ironwood™ Key Agreement Protocol. Ironwood enables two endpoints to generate a shared secret over an open channel, while WalnutDSA allows one device to generate a document that is verified by another. Both are implemented partially in software on the Intel® Cyclone V's ARM Cortex-A9, and partially in FPGA fabric. The FPGA image is configured to run WalnutDSA together with Ironwood as a demo.

A typical application for the powerful Cyclone V SoC is an IoT base station or gateway that is required to authenticate hundreds or thousands of remote endpoints (e.g. sensors). In this application, the Cyclone V would be programmed to authenticate each endpoint. To emulate such a scenario, the Chameleon96 image is configured to contain a WalnutDSA-signed certificate with a public key for a mock remote IoT endpoint. The certificate is included in a C header file of the demo app, and may be swapped out for other sample certificates.

For demo purposes, the Chameleon96 board interfaces with a PC via a USB virtual serial port. A terminal program running on the PC commands the board to validate the certificate, then conduct an Ironwood key exchange operation. The number of cycles to perform the WalnutDSA signature verification plus Ironwood shared secret calculation is displayed on the terminal. A more generalized version of the C code is provided for customer applications.

Package Contents

1. C source code that runs on the Cyclone V's ARM Cortex-A9 hard processor system. The Ironwood Key Agreement Protocol is implemented partially in software on the ARM Cortex-A9 and partially in the Cyclone V's FPGA fabric. All of the compute-intensive routines are executed in hardware (in the FPGA fabric). There are two C source code deliverables:
    a. The code that runs the combined WalnutDSA/Ironwood demo; and
    b. A more generalized version you can use for your own production applications along with a required SecureRF-specified DMA IP block.
2. Sample WalnutDSA signature plus Ironwood keys included in the C header file of the demo app
3. Verilog RTL for Ironwood core implementation, in Qsys-compatible Verilog RTL
4. Layout and timing constraint files

5. Test bench simulation system for Modelsim-Altera Edition with test vectors
6. Chameleon96 FPGA board image with SecureRF's Ironwood/Walnut demo incorporated

Requirements

- Chameleon96 development board
- Quartus Prime
- Chameleon96 SecureRF demo SD card image
- A serial terminal (e.g. PuTTY or similar)

Running the Demo

1. Load the Chameleon96 card image onto your SD card (if not done already) and ensure the bootsel switches are set to the proper configuration to boot from the SD card
2. Power up the board and open your serial terminal (115200 baud)
    a. The default login for Linux on the demo image is *root* with no password
3. Once the device has been programmed, launch the demo from the serial terminal
    b. If successful, the device will display the number of clock cycles it took to perform the WalnutDSA verification and Ironwood shared secret calculation.

Ironwood Key Agreement Protocol for FPGA Developers – Theory of Operation

This section introduces SecureRF's "Ironwood" Meta Key Agreement and Authentication Protocol.

Ironwood is considered a "meta" key agreement protocol because it has some characteristics of a public key solution, and some of a shared-key solution. Specifically, it does not require a secure key database in order to authenticate endpoints.

In a real-world application, two parties are involved in running Ironwood. One side is called the "Home Device" and the other is called "Other Device." Ironwood enables these two endpoints to generate a shared secret over an open channel without any prior communication.

The Home Device is provisioned with a public key, a private key and "T-values" (a collection of non-zero field elements). The Other Device is provisioned with an Ironwood public authentication token and a private authentication token. In general, the Home Device should be the device that is best able to safeguard the private key and T-values. For example, if the two devices consist of a mobile device and a hardened RFID tag, you will likely want to make the tag the Home Device. If run-time is of paramount

concern, it may be helpful to know that the Other Device requires three times less computations than the Home Device.

Either party can be configured to authenticate the other side, or they can be configured to authenticate each other. In most cases, you'll want the public key for the side being challenged to be included in a

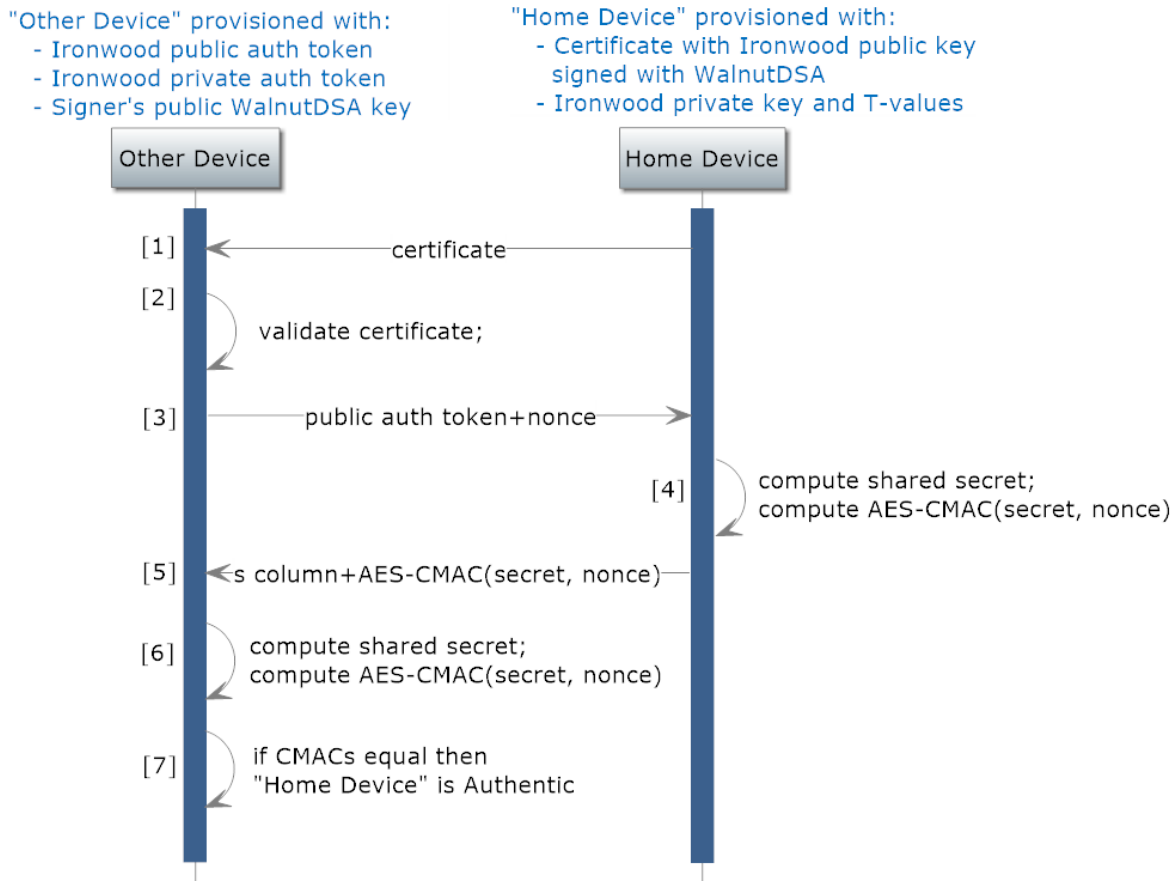
digital certificate that is signed by a trusted party. The challenger will validate the certificate before moving on to the shared secret calculation portion of the protocol. The Home Device and the Other Device may authenticate each other offline, i.e. without any connection to a supporting server.

Example Protocol

Below is example of how Ironwood can be used to authenticate one of the two parties, in this case, the Home Device. This is just one example—feel free to modify the protocol for your own requirements. The protocol given provides for replay protection by its use of a nonce (a random number) that is generated anew for each authentication session.

The example protocol also employs the use of SecureRF's Walnut Digital Signature Algorithm, which as previously mentioned is also available for the Chameleon96 board.

Sequence diagram for "Home Device" – "Other Device" authentication session



Protocol sequence for authenticating the "Home Device"

Step	Description
[1]	"Other Device" reads certificate from "Home Device"
[2]	"Other Device" validates certificate using a public WalnutDSA signing key from a trusted party
[3]	"Other Device" sends the public portion of its authentication token to the "Home Device" together with a 16-byte randomly-generated nonce
[4]	Upon receiving the auth token and nonce, the "Home Device" computes a shared secret using its own private key and the "Other Device" public auth token. The "Home Device" uses the shared secret to key an AES CMAC of the nonce.
[5]	"Other Device" reads the second portion of the "Home Device" public key known as the "s column." The "Other Device" also reads the "Home Device" CMAC of the nonce.
[6]	"Other Device" computes its own shared secret using the private portion of its authentication token, the "Host Device" public key and its s column.
[7]	"Other Device" uses its computed shared secret to key an AES CMAC of the original nonce. If the "Home Device" CMAC matches the "Other Device" CMAC, the "Other Device" considers the "Home Device" to be authentic.

Walnut Key Digital Signature Algorithm for FPGA Developers – Theory of Operation

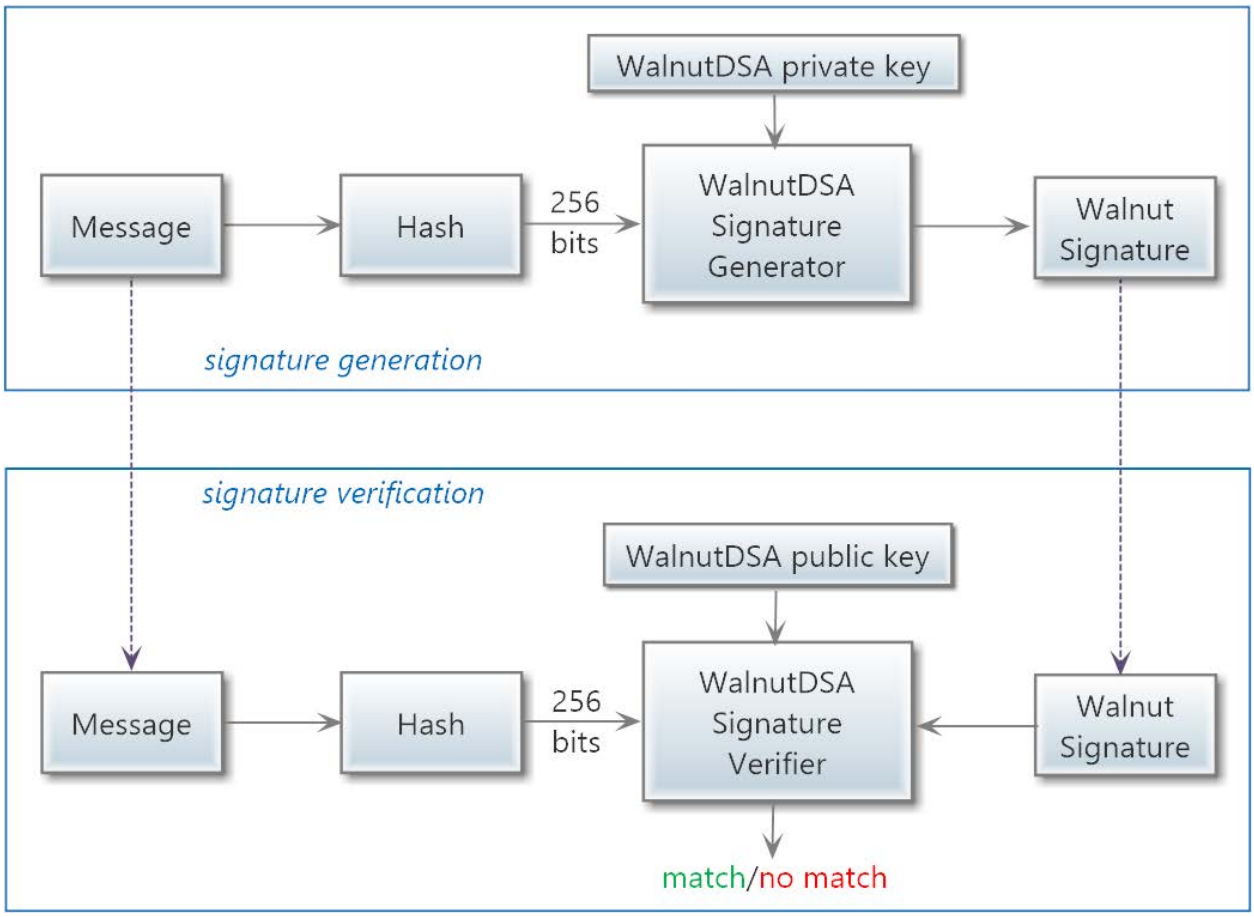
This section introduces SecureRF's Walnut Digital Signature Algorithm (WalnutDSA).

WalnutDSA allows one device to generate a message that may be verified by another device. More specifically, the algorithm allows a signer with a fixed private-/public-key pair to create a digital signature associated to a given message. This message can be validated by anyone who knows the public key of the signer and the WalnutDSA verification protocol.

One common application of WalnutDSA for the security solutions that SecureRF provides involves the exchange of public keys for SecureRF's Ironwood™ Key Agreement Protocol. In that protocol, two parties exchange their Ironwood public keys, and each compute a shared secret.

But how does the challenger party know that the public key it receives is really from the party it claims to be? The answer is to require the party who is challenged provide an Ironwood public key that is signed by a third-party that the challenger trusts, and to ensure that the signer's WalnutDSA public key is available to the challenger. If the signature verifies correctly using the signer's WalnutDSA public key, the challenger knows that the message could only have come from the signer since only the signer possesses the associated WalnutDSA *private* key. This process is depicted in Fig. 1. below where for our example, "Message" is an Ironwood public key.

Block diagram for WalnutDSA "signing" and "verification" process



The Chameleon96 board contains a WalnutDSA signature verification IP core. The WalnutDSA verifier requires a 256-bit input, so the message is hashed to a constant length prior to being processed. A separate Security Development Kit will be available from SecureRF to create WalnutDSA signatures.

Assistance & More Information

If you require assistance implementing our solutions, or would like more information, call our Technical Support Hotline at +1.203.277.3151 or email info@securerf.com.