George Mason University

Implementing OTA Updates for an IoT Security Device

Authors:
Gerson Dalton Cardozo
M. Sohail Iqbal
Aneesh Malhotra
Mohamed Nur
Ryan Thomas

Supervisor:
Dr. Jens-Peter Kaps

April 4, 2018

Contents

1 Executive Summary

This project will build upon a previous ECE 492 project *FPGA Enhanced Wireless Sensor Node for IoT Applications*. The project sought to use an FPGA to enhance the security of a wireless sensor node network while maintaining low power consumption. The network that this was implemented on was an in-home security system, which detects motion using an IR sensor on a node, and sends a picture of the intruder to the user via the gateway. Security can be an issue in wireless sensor networks and it may be necessary to update the private key of compromised node or update the firmware of a node to enhance its performance and functionality. The most efficient and inexpensive way to distribute such an update to many users without having to dismantle the system is over-the-air (OTA). In this process a manufacturer will distribute a secure update to its users and the system will update its own firmware. Our goal in this project is to provide OTA update capability to the in-home security system developed by the previous group.

2 Approach

2.1 Motivation

Many IoT devices now use wireless-sensor networks in which a user is able to control several devices called "nodes" through a single device called a "gateway". These kinds of networks are used in many popular smart home devices such as Google's Nest and Phillips Hue, most of which already provide OTA update capability. Additionally, security for these networks is becoming a concern. Wireless sensor networks are often deployed to monitor and respond to events occurring in the environment and are meant to be left unattended, making them susceptible to a variety of attacks [1]. With the versatile capabilities of FPGA's, we see many researchers using FPGA's to enhance the capabilities of wireless sensor networks, including security [2]. As FPGA's make their way into more wireless sensor networks, we will need to be able to be able to equip them with OTA update capability. Our goal is to provide this capability to our network that uses an FPGA for enhanced security.

2.2 Problem Analysis

The problem we face is to be able to provide OTA updates to the node of the previous system (Figure 1). Since the node is not directly connected to a computer, we must be able to reprogram the node using only the existing hardware on the node. Additionally, the node consists of an MSP432 microcontroller and an Actel IGLOO FPGA. Updating the security features such as the private key and the algorithm on the FPGA will require us to be able to reprogram the FPGA remotely. Likewise, updating the firmware and capabilities of the microcontroller will require us to be able to reprogram the MSP432 remotely.

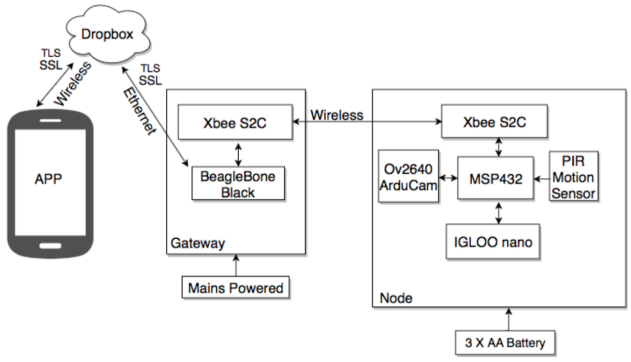


Figure 1: Top Level Diagram of Camera Security System

2.3 Solution

Our solution to this problem is to redesign the system to be able to request updates from an external sever, download the update to the Gateway, push the update to the MSP432 via the XBee, and allow the MSP432 to reprogram both itself and the FPGA through a bootloader. The update will simply consist of a .zip file that consists of a YAML file, the updated code, and some header information. Our goal is to be able to parse the update on the Gateway, and send bootloader commands to the node. The MSP432 on the node will then use these commands to reprogram its own program memory, as well as interface to and reprogram the FGPA.

2.4 Components

1. External Server
    - Contains secure signature and contains update files.
    - Stores user data collected from the node such as images.
2. Update
    - Contains a .zip file with a YAML file as well as the source code.
    - The YAML file will have have an MSP432 and FPGA section, and will be converted to bootloader commands.
3. Phone
    - Provides interface to user
    - Checks for and initiates updates
    - Allows for rekeying and initiating capture from the node camera.
4. Gateway
    - Main system component
    - Contains a BeagleBone Black running Linux and an XBee to communicate with the node
    - Parses YAML file, and maps it to bootloader commands for the MSP432.
    - Takes in data from the node and makes it accessible to the user.
5. Node
    - Secondary system component
    - Contains an MSP432, Actel IGLOO FPGA, an XBee, an ArduCam v5 5MP camera, and an IR sensor.

- The MSP432 controls all other components and will have the capability to reprogram itself and the FPGA.
- The node will capture an image and send it back to the Gateway.

2.5 Top-Level Design

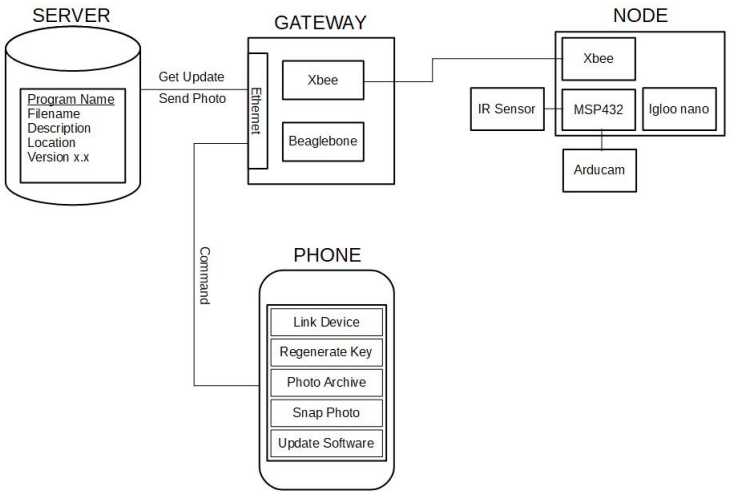


Figure 2: The camera security system no longer uses Dropbox, but rather a server to host the updates.

1. The server contains a file with the version number and location of an update.
2. The user compares the version number of the update to that of the system.
3. If the version numbers do not agree, the user will have the option to begin an update.
4. The Gateway will request the complete update, process it, push the the update and appropriate MSP432 commands to the node.
5. MSP432 will receive the update from the Gateway as well as instructions for reprogramming itself and the FPGA. This will be done via Bootloader commands.

The following diagram depicts this workflow.

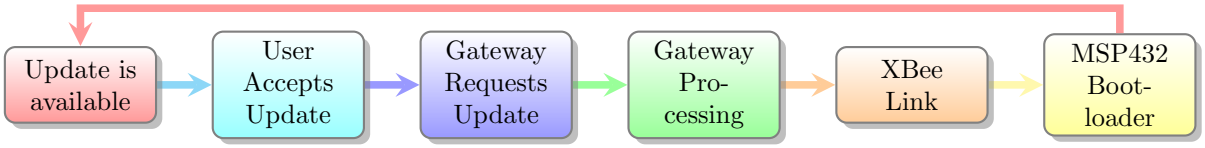


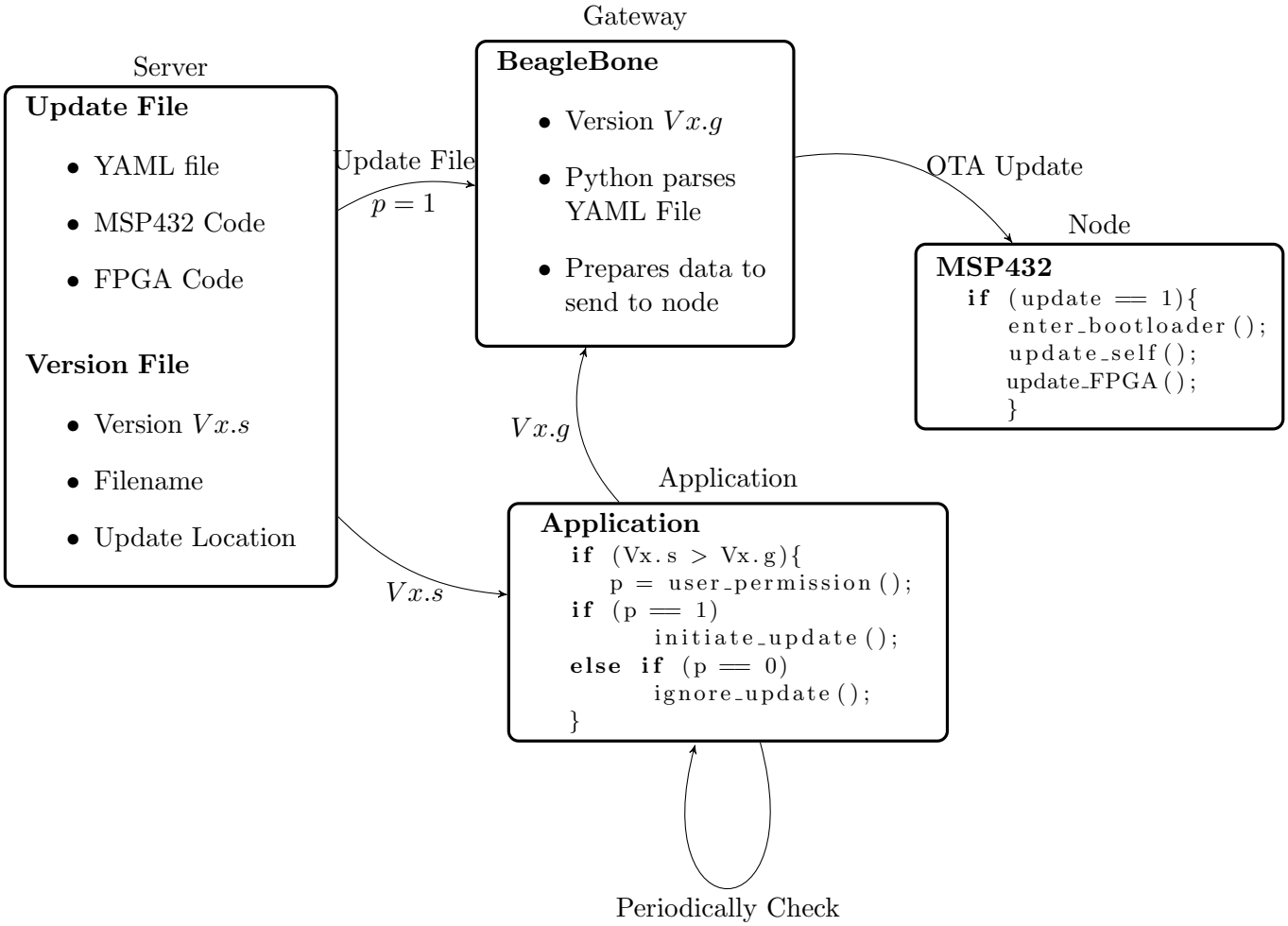
Figure 3: Update Workflow

2.6 Alternative Designs

2.6.1 Dropbox

We had initially decided to use Dropbox as a cloud storage system. Since the gateway is not a member of the local area network, this would have made communication between the Android device and the gateway to be much simpler. This communication, however, is much slower. Additionally a company pushing a firmware update to a user will not typically have access to a user's Dropbox account. A user can also accidentally delete important configuration files such as the ones used for rekeying, making it less reliable and less realistic than a dedicated server. For these reasons, we decided to use a dedicated web server for distributing updates and storing data.

3 System Architecture



3.1 Method for MSP432

The MSP432 can be given a *bootloader*, which is a program that allows a user to access memory. By using a bootloader, we can program particular blocks of MSP432 memory corresponding to the update. These bootloaders are often available online, and have the ability to access external devices via I^2C, and therefore obtain incoming update information from the XBee device.

3.2 Method for FPGA

Our system design architecture (shown above) shows each component involved in performing an update. Specifically, the process by which we update the FPGA using the MSP432 will be through a Jam Standard Test and Programming Language (Jam STAPL) player. This software will make it easier to send a binary bitstream to program the FGPA via the JTAG interface. We will, however, need to create an API that interfaces to the STAPL player and provides high level functions to use. The programming using this can be done both via the main function of the MSP432 and the bootloader, and we will need to decide which method to use.

3.3 YAML File

YAML is a data serialization language, and is essentially a tool to convert update data, which will consist of memory locations of the processor that need to be modified, and translate it into commands, specifically bootloader commands, which can be transmitted to the MSP432 through the XBee. This file will be processed on the BeagleBone Black on the Gateway using Python.

4 Preliminary Experimental Plan

1. Our first experiment will be to see if we can successfully use the JTAG interface on the MSP432 to program the FGPA. This is a preliminary experiment to test our conceptual design and determine any further complications.

2. Our second experiment will be to simply map some bootloader commands into a YAML file, and then send them to the node. If these two tests are successful, it will verify our conceptual design and we will be able to successfully implement an update.

3. Once the system is running completely, one goal we have is to be able to update the private key of the FPGA. We will create some code that will update the private key of the node and push that as an update to the user.

5 Preliminary Project Plan

5.1 Tasks

1. Create an "update" to send to the system

2. Download MSP432 bootloader and learn how to interface to it

3. Download STAPL player and develop an API

4. Construct a YAML file and write code to parse the file.
5. Update the application to allow for an update feature.

5.2 Allocation of Responsibilities

5.2.1 Aneesh

- Project manager responsibilities such as keeping everyone on schedule.
- Create update file, and update verification
- Work with YAML

5.2.2 Ryan

- Mainly responsible for writing MSP432 code to be able to reprogram itself

5.2.3 Gerson

- Create PCB design of final system, for both the gateway and the Node.
- Handle any issues interfacing via ZigBee to any of the devices

5.2.4 Mohamed

- Working on the application, which will interface to the server, to communicate with the gateway.
- Helping write code to update the BeagleBone on the Gateway

5.2.5 Sohail

- Working on downloading and implementing STAPL player
- Writing API to be able to interface to the FPGA via JTAG.

References

[1] J. Sen, "Security in Wireless Sensor Networks," *arXiv:1301.5065 [cs]*, Jan. 2013, arXiv: 1301.5065. [Online]. Available: http://arxiv.org/abs/1301.5065

[2] L. G, S. K, and V. K. R, "Elliptic Curve Cryptography implementation on FPGA using Montgomery multiplication for equal key and data size over GF(2m) for Wireless Sensor Networks," in *2016 IEEE Region 10 Conference (TENCON)*, Nov. 2016, pp. 468–471.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987. [Online]. Available: http://www.ams.org/home/page/