# Building a Low-Cost and State-of-the-Art IoT Security Hands-On Laboratory

Bryan Pearson[1], Lan Luo[1], Cliff Zou[1(✉)], Jacob Crain[2], Yier Jin[2], and Xinwen Fu[1,3]

[1] University of Central Florida, Orlando, FL 32816, USA
{bpearson,lukachan}@Knights.ucf.edu, czou@cs.ucf.edu
[2] University of Florida, Gainesville, FL 32611, USA
jcrain@ufl.edu, yier.jin@ece.ufl.edu
[3] University of Massachusetts Lowell, Lowell, MA 01854, USA
xinwnefu@cs.uml.edu

**Abstract.** The popularity of IoT has raised grave security and privacy concerns. The huge IoT botnets Mirai and Reaper were built on compromised IoT devices. In this paper, we propose to develop a low-cost platform with an industrial grade microcontroller (MCU) ESP32 equipped with a crypto co-processor ATECC608A and create teaching materials including labs and case studies for IoT security education. MCUs have broad applications in IoT. Sensor nodes in various smart systems such as smart home, smart health and smart grid can use MCUs to process commands and perform automatic control. We will develop effective, engaging and novel teaching materials on IoT hardware security, operating system/firmware/software security, network security, and data security with the low-cost IoT kit and IDE. The teaching materials will contribute to the Cybersecurity Workforce Development Initiative led by NICE and help respond to a dynamic and rapidly developing array of cyber threats including those resulting from IoT.

**Keywords:** IoT · Microcontroller · Teaching materials

## 1 Introduction

Internet of Things (IoT) interconnects everything including physical and virtual objects together through communication protocols. IoT has broad applications in digital healthcare, smart cities, transportation, agriculture, logistics and many other domains. The global IoT market is booming. According to Forbes [13], the IoT Market will reach $520B by 2021.

### 1.1 Need for Promoting Education in IoT Security

The popularity of IoT has raised grave concerns about security and privacy. When medical devices are connected to the Internet, compromised medical


Published by Springer Nature Switzerland AG 2020
A. Casaca et al. (Eds.): IFIPIoT 2019, IFIP AICT 574, pp. 289–306, 2020.
https://doi.org/10.1007/978-3-030-43605-6_17

devices may endanger the life of patients. Hacked autonomous cars may crash. Hackers exploited default passwords and user names of webcams and other IoT devices and deployed the Mirai botnet [11]. The huge botnet was then used to conduct the distributed denial-of-service (DDoS) attack against Dyn DNS servers. The IoT reaper botnet was discovered in 2017 and it exploited newly found vulnerable IoT devices [9].

We have also disclosed various vulnerabilities of IoT products on market. We exploited hardware vulnerabilities of the Nest Thermostat, which aims to learn a user's heating and cooling habits to help optimize scheduling and power usage [23]. The hardware architecture of Nest lacked proper protection, allowing attackers to install malicious software into the unit. Through a USB connection, we demonstrated how the firmware verification done by the Nest software stack can be bypassed, providing the means to completely alter the behavior of the unit. Any information stored within the unit was then exposed too. We performed a comprehensive analysis of PurpleAir, a popular low-cost air quality sensor [28]. PurpleAir has one of the largest operational low-cost sensor networks worldwide, being used by individuals and non-profit and governmental agencies for community air quality monitoring. Multiple security vulnerabilities are identified in PurpleAir, including plaintext communication, weak authentication mechanisms, and lack of data integrity measures. By exploiting these vulnerabilities, attackers can impersonate any victim PurpleAir sensor and pollute its data using fabricated data. This is the first security analysis of low-cost and connected air quality monitoring systems. The researchers have been working with PurpleAir to patch their system.

IoT security should be addressed from five aspects [25], including hardware security, operating system (OS)/firmware security, software application security, secure networking protocols and data security. Different IoT systems have different requirements. In this paper, we propose to develop a low-cost platform with the industrial grade microcontroller (MCU) ESP32 [20] and crypto co-processor ATECC608A [30] for IoT security education. While CPUs are more versatile and powerful, MCUs are often used in sensor nodes in various smart systems such as smart home, smart health and smart grid to process commands and provide automatic control.

We will often take an Internet enabled air quality monitoring system as a case study to demonstrate security requirements of an IoT system, and we believe that other IoT systems shall share similar attributes. A secure environmental monitoring system should have hardware security and be able to prevent attackers from reading and changing the data on the device, particularly when the attacker has physical access to the device. Hardware security is a great challenge. For example, advanced attackers may manipulate the flash directly through its I/O interface. The IoT device should have system and firmware security so that it can detect the change of its firmware and protect the overall system. To further protect the firmware and sensitive data stored on the flash, data security through strategies such as flash and file encryption shall be adopted. Secure firmware upgrade of IoT systems is also key to the longevity of an IoT system, since no one can guarantee

that a software has no bugs, and security and functionality patches are always expected.

Network security is required to secure network traffic to and from an IoT device, for example, through SSL/TLS (which will be referred to as TLS for simplicity) to establish mutual authentication, communication encryption and integrity between the device and a server. Mutual authentication is critical for any IoT system. Those systems without mutual authentication often have various vulnerabilities [15,24,25,31,32]. Without device authentication, a fake device may obtain security credentials from the server or a smartphone app. Without server/user authentication, a fake server or user can cheat on the device and collect sensitive information. Certificate based mutual authentication based on public key crypto is often the most feasible and simple implementation of mutual authentication. In TLS certificate-based mutual authentication, a client verifies the server's certificate and identity while the server performs similar operations to authenticate the client.

From the discussion above, we can see that there is an urgent need of promoting education in IoT security and privacy given that IoT will be ubiquitous and its security should be systematically addressed from the perspective of the hardware, system/firmware, software, networking and data security.

### 1.2 Need for Low-Cost and High-Quality Hardware Platforms for IoT Security Education

We propose a low-cost platform with the industrial grade microcontroller ESP32 [20] equipped with a crypto co-processor ATECC608A for IoT security education. A low-cost platform will be easy for dissemination so that schools with low budget can afford it. Pedagogical theories support the use of industry grade hardware platforms for IoT security education. Kaylene Williams and Caroline Williams list various ingredients that can improve student motivation, including student and content [37]. The student ingredient suggests students should be able to connect education results to their future career and even entrepreneurship. The content ingredient requires the content to be accurate, timely, useful, and relevant to students in their life. The hands-on labs and case studies developed in this paper on secure IoT systems are novel, timely and relevant to real life. The platform shall create an environment that students may connect knowledge learned from the school to the industry. With such teaching materials, students will be able to produce practical prototypes such as secure environmental monitoring sensors and generate an impact on the society through innovation. Such an environment will motivate students to learn.

There is also a strong interest among students to learn IoT security and privacy. This is demonstrated by two preliminary surveys performed by us: one tutorial on IoT security at Design Automation Conference (DAC) 2018 and the other from an IoT security and privacy class offered in Fall 2018 at UCF. The survey results from 51 responses in Table 1 reveal that 94.1% of the students do not know how to hack IoT hardware at all or only know somewhat. We cannot expect these students to know how to secure the IoT hardware. 94.1% of the

students do not know how to perform traffic analysis of IoT traffic at all or know only somewhat. These students will not be able to perform risk assessment of IoT communication protocols. 89.2% of the students either strongly agree or agree that it is necessary to learn IoT security through hands-on exercises. Overall, the preliminary surveys demonstrated the need and urgency to develop effective education materials to equip the future workforce with adequate knowledge and expertise in IoT security and privacy.

**Table 1.** Survey on the need for transiting research in IoT security into education

| Question | Answer |
|---|---|
| 1. Do you know how to hack hardware of IoT devices through its debugging ports or other hardware interfaces? | 68.6% No, 25.5% Somewhat, 5.9% A lot |
| 2. Do you know how to analyze network traffic from IoT devices or the app controlling the devices? | 62.7% No, 31.4% Somewhat, 5.9% A lot |
| 3. It is necessary to learn IoT security through hands-on exercises | 61.7% Strongly agree, 27.5% Agree |

We have following major objectives. Please note: the term "laboratory" in this paper refers to the actual physical laboratory, and the term "lab" refers to a student assignment or project that plays with the IoT kit to learn specific knowledge and skills. While we focus on IoT security, privacy is often mentioned since encrypting data and communication is part of privacy.

**Objective 1:** Design an affordable low-cost IoT kit with an Integrated Development Environment (IDE) so that schools with low budget can afford such a laboratory on IoT security and privacy education. Given that the kit is small, a physical laboratory may not be necessary. Students can take home the kit in a box and perform the teaching labs and case studies.

**Objective 2:** Design a suite of teaching labs and case studies using the low-cost IoT kit so that students can learn the state-of-the-art techniques to secure IoT devices. We will design teaching modules covering hardware security, operating system/firmware/software security, network security, and data security. Please note that given the broad areas of IoT security, we focus on those unique and important security features offered by the low-cost IoT kit and do not intend to cover every security topic. For example, in the threat model of an individual defense measure, we will not consider the following physical attacks on hardware platforms: power/electromagnetic side channel attacks [33], laser voltage probing (LVP) based non-destructive reverse engineering [26,27], focused ion beam (FIB) based destructive reverse engineering [36], and other circuit level vulnerabilities. The corresponding solutions to these attacks will lead to the redesign of the microcontrollers and microprocessors. These topics are out of the scope of this

paper. The hardware security labs in this paper will address those caused by illegal access to input/output pins. For software security, secure boot can detect the change of the software. We will not cover dynamic security measures such as intrusion detection. However, the covered secure measures will be able to defeat most serious IoT attacks discovered so far.

**Objective 3:** Integrate the developed IoT teaching labs and case studes into related courses.

## 2 A Capable and Low-Cost IoT Laboratory

We now introduce the low-cost IoT development kit, the laboratory setup and the integrated development environment (IDE) to use the kit. Such an IoT development kit can be less than $10. For hardware security teaching labs, students need the JTAG debugger and bus pirate, each of which costs around $30. Other laboratory supplies and materials including multimeters and soldering kits are of low cost too.

### 2.1 Low-Cost IoT Development Kit and Laboratory

We will design a very low-cost and affordable IoT development kit, as illustrated in Fig. 1. It includes a capable industrial grade microcontroller (MCU) – ESP32 development board ($10.99 at Amazon; $4.27 at AliExpress), a crypto co-processor – ATECC608A ($0.86/unit at DigiKey in pack of 25, $0.55/unit at Microchip in pack of 5000), a temperature and humidity sensor – DHT11 ($2.00 at Amazon; $0.83 at AliExpress), a small breadboard ($1.60 at Amazon; $0.90 at AliExpress), a USB to Micro USB cable ($2.00/unit in pack of 5 at Amazon; less than $1.00 at AliExpress) and a few jump wires (less than $1.00). A computer is required to program the MCU and it is assumed that students have computers such as laptops or the school provided computers. If the right seller is chosen, the low cost IoT development kit costs only $8.55, less than $10.

For the hardware security teaching modules, A JTAG debugger ($29.95 at Amazon) is needed to connect to the JTAG interface of ESP32. A Bus Pirate ($29.95 at Amazon) is needed to play with the UART interface and the flash SPI interface. A pair of JTAG debugger and bus pirate costs $59.90.

Figure 2 shows the laboratory setting. If an institution does not have space, there is no need of a physical laboratory. Since the components are small, a small box can hold all the components. A guided lab can be executed in a classroom and lab work can also be performed at home. The laboratory may need soldering iron kits to solder pin headers onto an ESP32 development board, an ATECC608A onto a breakout board, or pin headers onto breakout boards. Multimeters can be used to assist the hardware labs while they are not necessary. One set of soldering kit and multimeter costs only $15.64 at AliExpress. IoT devices must be connected to the Internet and the data from the IoT devices can be saved on a server. ESP32 supports WPA2 Personal or Enterprise, and can be connected to an isolated private network, a campus WiFi network or the widely

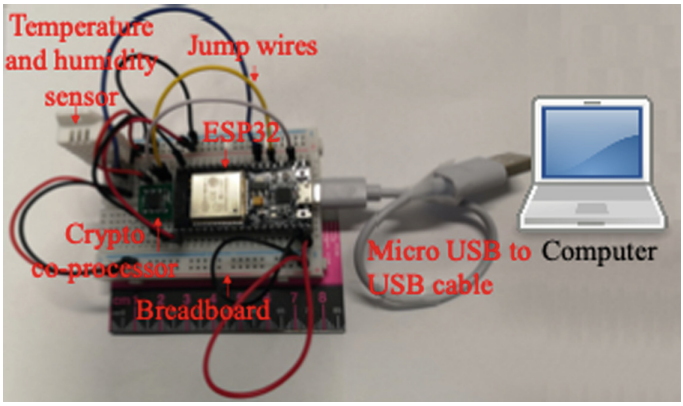


**Fig. 1.** Single IoT platform

deployed eduroam [29]. Amazon AWS IoT [10] provides a free tier of services including IoT core, database and storage. ESP32 devices can be connected to an open source MQTT (Message Queuing Telemetry Transport) server such as Mosquitto [3], which can use a MySql database to save data. A generic computer can run such a MQTT server and MySql database.

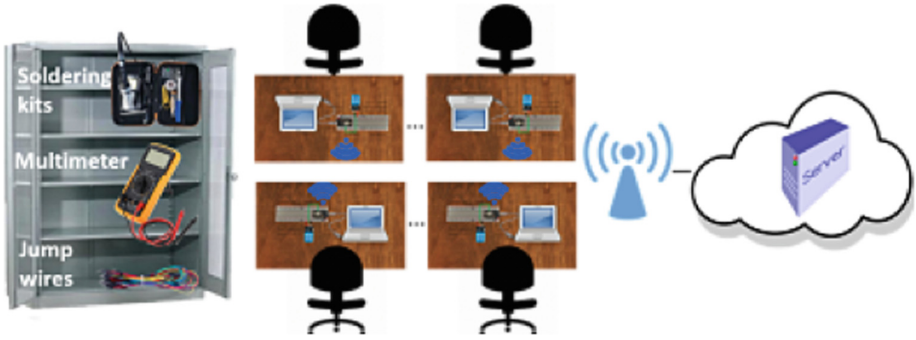


**Fig. 2.** IoT laboratory

### 2.2 Creating an Integrated Development Environment (IDE) for IoT Security Education

Hardware is only part of an IoT system. We plan to create a unified IoT IDE (integrated development environment) for the IoT development kit in Fig. 1, which will be used for teaching modules on IoT hardware security, system/firmware/software security, networking security and data security. The

native programming environment of ESP32 is Espressif IoT Development Framework (esp-idf) [18], which is a command line based environment. The ESP32 community has migrated the esp-idf into the Arduino core [16] for the ESP32 WiFi and Bluetooth chip so that the Arduino IDE [1] can be used to compile an ESP32 application, flash it into the chip, and monitor debugging messages from ESP32 through a serial port. The Arduino community developers have also developed various libraries such as one for the temperature and humidity sensor DHT11. However, the Arduino IDE has no debug capabilities, and lacks support for multiple files on a project. The alternative is the Eclipse IDE [17], which is integrated with esp-idf and supports JTAG, OpenOCD [6] and GDB debugging [35].

Once the IDE is chosen, we will ensure that the IDE supports the developed teaching modules on configuring ESP32's secure features such as secure boot and programming labs with crypto libraries. (i) Configuring ESP32's secure features. Currently, the configuration is performed at the command line. We plan to develop a GUI for such configuration. (ii) Programming with crypto libraries. We must carefully choose the crypto library, which shall support secure features of both ESP32 and ATECC608A. There are two choices, mbedssl [7] and WolfSSL [8]. ESP32 is shipped with mbedssl and provides an API interface for TLS [19]. WolfSSL is ported to ESP32 and is compatible with Microchip's CryptoAuthLib, a crypto library that supports ATECC608A. Whenever needed, we will extend the crypto library and ensure that the chosen library will support the functionalities of ECC, HMAC and AES of ATECC608A, and can connect to Mosquitto, AWS IoT and other MQTT brokers via TLS.

Since an institution may prefer a private network for all the teaching labs and does not want to use AWS IoT, we plan to extend the open source MQTT project Mosquitto with the database capability and mutual authentication. There are two issues with Mosquitto. The default configuration of Mosquitto does not incorporate MySql. We will extend Mosquitto so that IoT devices can send data to the server, which stores data into a MySql database. The second issue with Mosquitto is its current implementation of SSL/TLS mutual authentication is problematic. In the client authentication, the server verifies the client's certificate and checks if the subject fields of the client certificate include the client's IP. This will tie the client to that IP. This strategy is not flexible and practical. We will adopt the certificate based client authentication used by SSH and AWS IoT for Mosquitto. That is, the client certificate will be stored at the Mosquitto server for the purpose of authentication.

## 3 Developing Teaching Labs for IoT Security

We now introduce teaching labs and case studies that will be developed for students to master hardware security, secure key storage, secure boot, data security, network security, secure firmware upgrade and crypto co-processor based on the IoT kit in Fig. 1 and the IDE that we will adopt and extend.

### 3.1 Developing Teaching Modules on Hardware Security

We first introduce hardware security capabilities of ESP32 and then discuss corresponding teaching modules on hardware security based on ESP32.

**Hardware Security on ESP32.** The first step to accomplishing hardware security is to disable I/O ports that may be present on the device. In the case of ESP32, we must disable the JTAG and UART interfaces, since they provide venues for attackers to read and write on the ESP32.

The Joint Test Action Group (JTAG) formalizes a series of standards for boundary scanning and debugging a chip. With JTAG, the programmer can test each component of the chip separately to verify it is connected and functioning correctly. The Open On-Chip Debugger (OpenOCD) is an open-source project and can interact with the JTAG interface in a GNU Debugger (GDB) environment. OpenOCD was extended to add support for the ESP32 JTAG chain. Programmers can use GDB to communicate with OpenOCD, providing complete flash access of the ESP32. It is possible to read and write to any byte of memory, including registers and instruction flow through the JTAG chain. To disable JTAG, the corresponding eFuse value of ESP32 should be set to 1. The ESP32's eFuse is a 1024-bit partition of one-time programmable memory, separated into four 256-bit blocks. Upon setting a value, hardware "fuses" are burned, rendering these values irreversible.

Universal Asynchronous Receiver/Transmitter (UART) is a circuit which allows two devices to communicate over a serial connection. Both devices in a UART connection can either transmit or receive bytes of data. Using a serial register, UART will convert this data either from serial to parallel or vice versa, depending on whether the data is being transmitted or received. Unlike JTAG, which can debug devices, UART is often used for communication. The primary purpose of UART with respect to the ESP32 is to upload an application or firmware to the flash. Other possibilities with UART include monitoring output from the console, and reading or modifying direct memory addresses. The UART bootloader is implemented through an external interface known as esptool. If flash content is encrypted by the encryption key stored in the eFuse, then the UART bootloader will transparently decrypt this content before reaching the serial monitor. Similarly, the UART bootloader will transparently encrypt data when flashing it via esptool. To disable the insecure properties of the UART bootloader, we must set proper eFuse values. Afterwards, the UART bootloader cannot read or write to the encrypted flash.

**Teaching Modules on Hardware Security on ESP32.** We will develop JTAG ethical hacking lab, JTAG defense lab, UART ethical hacking lab, UART defense lab, and flash ethical hacking lab.

JTAG hacking lab. Access to a full JTAG chain allows access to all portions of an integrated circuit, some debugging configurations notwithstanding [21]. In this lab, students will learn the different signals that appear on a JTAG port as

well as their usage. Students will also be introduced to the standard JTAG state machine, as well as how systems are designed to utilize it. Then students will proceed to identify the JTAG pins of the ESP32 and connect a debug probe to it. We will then provide the students with tools to build the OpenOCD debug bridge and the GNU Debugger (GDB) for the target platform. In doing so, students will get experience on finding and constructing their own debug environment for reverse engineering platforms. With OpenOCD and GDB, students will attempt to reverse engineer a binary provided by the lab instructor. Students will learn to utilize the Python facilities provided by GDB to develop their own plug-ins to facilitate the task at hand. As part of this lab, students will learn to view and modify CPU registers, as well as bypassing the local authentication mechanism provided by the binary. Furthermore, through the JTAG interface, students will modify the contents of memory as the binary runs as to modify its behavior and extract runtime secrets such as ephemeral keys used by the binary to establish secure communications with a remote server. Lastly, students will utilize the JTAG interface to permanently modify the binary within the device introducing a backdoor to allow bypassing local authentication as well as leaking credentials.

JTAG defense lab. The JTAG chain has to be disabled in a final commercial product given that attackers may exploit it to interrupt the microcontroller or access sensitive data on the flash. In this lab, students will learn how to disable the JTAG chain by setting JTAG_DISABLE of the ESP32 to 1. Students will use the debug probe again and test if they can still manipulate the chip after JTAG_DISABLE is set as 1. One thing to note is once JTAG is disabled, it cannot be enabled again since the eFuse is physically burned. Therefore, if the ESP32 will be reused, the JTAG defense lab cannot be truly performed and students should only discuss the principle of disabling the JTAG.

UART hacking lab. For this lab, students will be introduced to communications through UART. The lab will center around using a specially crafted binary provided by the instructor. Students will identify the UART lines of the ESP32 microcontroller and capture data generated by the binary using a cheap signal analyzer such as the Bus Pirate [34]. The students will learn to decode UART frames, as well as to compute the BAUD rate being utilized by the device. With the computed BAUD rate, students will capture all incoming frames and examine the received data for any possible hints on device operation. Students will then interact with the device through a serial terminal emulator, allowing them to dynamically send and receive data from the device. Utilizing Python and the pyserial library, students will then develop a framework to automatically send and retrieve data from the device through UART. The binary will be designed to contain a way to locally authenticate a debug mechanism through its serial port. This is done in a semblance of devices that use the UART as a debug method. Students will utilize this opportunity to brute force the debug authentication credentials, or to perform other styles of attack (such as dictionary attacks) against the authentication prompt. Once authenticated, students will proceed to extract secrets from the device, such as wireless network credentials. Lastly, students will utilize the UART to communicate with the ESP32 Boot ROM code and

extract portions of the binary which reside in the on-board flash containing the authentication credentials, modify them, and write them back to the on-board flash utilizing the Boot ROM code.

UART defense lab. The UART has to be either disabled or protected through a strong authentication mechanism in the final commercial product given that attackers may exploit it to access sensitive data on the device. In this lab, students will explore the option to secure the UART. In events where the UART exposes sensitive device information, students will learn ways to securely authenticate with the device to allow access such as timed "knocks", and signed messages. Students will also investigate the option of adding cooldowns after failed authentication attempts and study how it affects the security and usability of the device. The second option is to disable the UART in production devices when it is no longer needed, or to limit its functionality. Students will play with the three eFuse values DISABLE_DL_ENCRYPT, DISABLE_DL_DECRYPT and DISABLE_DL_CACHE, and study the impact. One thing to note is once UART is disabled, it cannot be enabled again since the eFuse is physically burned. Therefore, if the ESP32 will be reused, the UART defense lab cannot be truly performed and students should only discuss the principle of disabling the UART through setting the eFuses.

Flash ethical hacking lab. The ESP32 has multiple SPI channels, with channel 0 dedicated to mapping the external serial flash to memory, and channel 1 for performing writes to this flash. In this lab, students will be provided with a binary by the instructor to be placed in their boards. Students will then probe the SPI lanes and identify read transactions using a cheap signal analyzer such as the Bus Pirate when the device is in operation, identifying the underlying instruction stream being executed by the microcontroller. Students will then interact with the device to allow it to fetch an encrypted secret from the external flash memory. Students will then use the captured instruction stream to decode the secret in order to further their interaction with the device. Students will then use a secret of their own and encrypt it with the binary's algorithm and write it back to flash memory using a cheap programmer such as the aforementioned Bus Pirate. The lab shows the students why flash encryption is needed to protect sensitive data stored in the flash.

### 3.2 Developing Teaching Modules on Secure Key Storage on ESP32

We first introduce the hardware based secure key storage and then discuss teaching modules on secure key storage on ESP32. Secure key storage is part of ESP32 features securing the system and flash firmware (including the application/software) from unauthorized access. ESP32 uses FreeRTOS to manage the hardware components and run a user task/application [4].

**Secure Key Storage on ESP32.** Simply encrypting the data will not guarantee that an IoT system is secure. We must also securely store the encryption keys, so that only trusted systems can access it when needed, and even a software malware that has hacked into the system cannot access the keys. That is,

secure key storage protects secret keys from being externally revealed or modified. The ESP32's eFuse allows for secure key storage. Recall this eFuse contains four 256-bit blocks. Block 0 is reserved for the MAC address, SPI configurations, and related security settings. Blocks 1 and 2 are actually used for key storage — block 1 stores the flash encryption key, while block 2 stores the secure boot key. Both keys are 256 bits and generated using an internal Random Number Generator (RNG) hardware accelerated algorithm. Block 3 is undefined by default, but a programmer may use it to store application-specific encryption keys.

**Teaching Modules on Secure Key Storage.** Secure key storage lab. In this lab, students will learn how to utilize the secure key storage and understand the importance of keeping the secret. ESP32 allows pre-generation of a flash encryption key on a host computer, which can be used to burn this key into its eFuse. This approach is recommended for the development phase. Students will be required to use the given flash encryption key to decrypt encrypted data, which may contain sensitive data such as WiFi credentials. However, when the key is kept secret, secure key storage will not leak the key even if a malware gets inside the device. Students will also learn how to enable the secure key storage for the production phase.

### 3.3 Developing Teaching Modules on System/Firmware Security

We first introduce secure boot, which is part of ESP32 features securing the system and flash firmware from unauthorized access. We then discuss teaching modules on secure boot on ESP32.

**Secure Boot on ESP32.** ESP32 uses hardware-based secure key storage and secures the booting of the firmware. Secure boot requires all components of the firmware be signed and verified before executing [22]. If either the software bootloader or the application firmware is modified, the device will refuse to boot. Once properly configured, two keys are necessary to enable secure boot. The first key is a 256-bit secure bootloader key and allows the ROM bootloader to validate the software bootloader. The second key is the secure boot signing key, generated with ECDSA with the NIST256p curve. The manufacturer will generate the ECDSA keypair. The signing key is used to generate image signatures, so it must be available to the manufacturer. The software bootloader and the application are validated via a "chain of trust" model.

**Teaching Module on Secure Boot.** Secure boot lab. The lab has two parts, attack and defense. As introduced above, if secure boot is not enabled, it is possible that an attacker can change the binary code without being detected. In the attack part of this lab, secure boot is not enabled on ESP32. Students will be required to retrieve the binary code from the flash, reverse engineer the binary code, and change the logic of the code. For example, students will be required

to change the binary of an air quality monitoring sensor and send fake data to a server. Data integrity is critical for air quality monitoring since air pollution may incur public outcry. In the defense part, students are required to enable secure boot. Students will perform the attack again and observe if secure boot can detect the change of the binary of the device. Secure boot is a critical part of building a trustworthy IoT device.

### 3.4 Developing Teaching Modules on Data Security

**Data Security on ESP32.** ESP32 can encrypt the entire flash using a secure AES-256 key. The AES key is stored in block 1 of the eFuse. Once written to the eFuse, the read and write bits for the key are set to prevent anyone from reading or modifying the key. When flash encryption is enabled, application-based flash partitions, i.e., factory and over-the-air (OTA) partitions, are encrypted by default. From there, decryption can only occur at runtime via the flash controller. The flash controller is a hardware component that can perform the following two runtime operations using the AES key: (i) Decryption of memory-mapped read accesses to flash; and (ii) Encryption of memory-mapped write accesses to flash. It is also possible to encrypt other flash partitions by manually setting an "encrypt" flag for a partition. This requires generating a custom partition table rather than using the default table (which only encrypts factory and OTA partitions).

**Teaching Modules on Data Security on ESP32.** Data security lab. From the flash ethical hacking lab or as instructed by the lecturer, students have learned that attackers can read the content of the flash on a device from the flash's interface (such as SPI). In this lab, students will be guided to enable flash encryption of ESP32 to defeat such flash attacks. Students will understand how a system works with an encrypted flash seamlessly.

### 3.5 Developing Teaching Modules on Network Security on ESP32

We will first introduce ESP32's security features for network security and then discuss the teaching labs.

**Network Security on ESP32.** The challenge to implement TLS on an IoT device is often the cost and efficiency of implementing the public key based cryptographic functionalities. As discussed above, the hardware and cost may no longer be the bottleneck. ESP32 has cryptographic hardware acceleration for RSA and Random Number Generator (RNG). Our extensive experiments show that the performance of TLS on ESP32 is satisfactory in various application domains. ESP32 also has cryptographic hardware acceleration for AES and SHA2 so that TLS can be fully implemented. AES encryption can be implemented for communication secrecy and HMAC will achieve communication integrity.

**Teaching Modules on Network Security on ESP32.** We will develop two teaching labs including network attack lab and network defense lab.

Network attack lab. In this lab, students will use mitmproxy [5] to perform traffic analysis of IoT networking traffic. Mitmproxy is a man-in-the-middle proxy between an http(s) client and a server. It can intercept, modify, replay and save http/s traffic. Students will use mitmproxy to decrypt https traffic between a target IoT device and a server to understand its protocol. Students will set up mitmproxy at a computer. Mitmproxy pretends to be the IoT device for the server because the server normally does not check the identity of the IoT device. Mitmproxy can pretend to be the server for the IoT device too. However, the client normally checks the authenticity of the server by verifying the server's certificate. In this case, the mitmproxy will behave as a fake certificate authority (CA), generate a fake server certificate and send it to the client. Students will have to replace the original CA's certificate on the device with the fake CA's certificate so that the fake server certificate can be authenticated, the binary code can run and the mitmproxy can observe the communicating traffic between the server and client to figure out the communication protocol between them.

Network defense lab. Students are required to program and use SSL/TLS with RSA acceleration to secure network connection to a MQTT server, either Mosquitto or AWS IoT. Mutual authentication between the IoT device and the server will be required. This requires the client have a private key stored locally. Students will learn how to generate the public key pair with openssl or Amazon AWS IoT. Students will understand the issue with the private key hardcoded into the IoT application, given ESP32 does not provide secure key storage for RSA private keys. That is, if a malware gets into the device through means such as buffer overflow vulnerabilities, the malware may steal the private key and then a fake device can be created to impersonate the original device. This shows the necessity of both secure key storage and hardware acceleration of RSA.

### 3.6 Developing Teaching Modules on Secure Firmware Upgrade

We will first introduce ESP32's security features for secure firmware upgrade, i.e., secure Over-the-Air update (OTA), and then discuss the teaching labs.

### 3.7 Secure Over-the-Air Updates (OTA) on ESP32

OTA is a process in which the MCU fetches a new image from a remote location, stores this image in the flash, and loads it on successive reboots. OTA updates are seamless and transparent, and many devices can be updated concurrently. The drawbacks are that wireless updates introduce additional attack vectors that must be avoided. The ESP32 offers native library support for https OTA updates. ESP32's partition table includes OTA partitions, which store potential firmware for the ESP32. The otadata partition points to the newer firmware. Upon downloading a new update, the unused partition will be overridden, leaving the current firmware untouched. If the update fails, the device will revert to the

previous application. If it succeeds, the system is updated to point to the correct partition, and the system reboots to the new firmware.

**Teaching Modules on Secure Over-the-Air Updates (OTA) on ESP32.** We will develop OTA attack lab and OTA defense lab.

OTA attack lab. Students are required to attack a device without secure OTA firmware upgrade and change the firmware. There are two cases students will experiment on. In the first case, the device does not employ secure firmware upgrade and allows arbitrary firmware upgrade. Apparently, this allows the attacker to change the whole system arbitrarily. In the second case, the secure OTA is enabled, but it does not check the version number of the upgrade. Therefore, an old firmware with vulnerabilities can be upgraded into the device.

Secure OTA lab. Students will experiment on two alternatives of upgrading. In the first case, https will be used for secure OTA firmware upgrade. This is convenient since the manufacturers may utilize this strategy to push a new firmware into individual devices. In the second case, WiFi or Bluewooth of ESP32 will be used for secure firmware upgrade. This requires users to perform the upgrade from their smart devices such as smartphone or computers. The users first download the new firmware from the manufacturer and then perform the secure firmware upgrade locally.

### 3.8 Developing Teaching Modules on Crypto Co-processor

ESP32 does not have ECC hardware acceleration and does not provide secure key storage for its RSA hardware acceleration. It can cause problems when a malware breaks into the system and steals the private key hardcoded in the firmware. We will first introduce the very low-cost crypto coprocessor, Microchip's ATECC608A, which can address these issues, and then discuss the teaching lab.

**Microchip's ATECC608A.** Microchip's ATECC608A is a cryptographic coprocessor with secure hardware based key storage. It can store 16 keys, and supports ECDSA, ECDH, SHA-256 & HMAC, AES-128 and other features. Communicating with ATECC608A is performed through either a GPIO (general-purpose input/output) pin or a standard Inter-Integrated Circuit (I2C) interface, which is a widely supported serial protocol.

There are two reasons why we want to use a crypto coprocessor with ESP32. First, an old MCU may not have modern support of secure boot, flash/file encryption and hardware crypto acceleration. ATECC608A can be used to secure those MCUs and other processors. ATECC608A can be used with ESP32 to implement those features. Second, ESP32 does not have ECC hardware acceleration while ATECC608A has. Therefore, the use of ATECC608A will boost ESP32 for its SSL/TLS implementation. Software implementation of ECC will hardcode the ECC private key into the software. A malware that gets into the device will be able to read it and use it to impersonate the device. With ATECC608A, the

ECC private key can be burned into ATECC608A's hardware secure storage and will never leave the chip. Even the malware will not be able to get it. It will be also good for students to compare the performance of different implementations of SSL/TLS with RSA and ECC.

**Teaching Modules on Microchip ATECC608A.** Mutual authentication lab with ATECC608A: One weakness of ESP32 is it does not provide secure key storage for the RSA private key used for SSL/TLS. Even if flash encryption is used, malware that gets into the device will be able to read the private key. In this lab, students are required to generate the ECC key pair and burn the ECC private key into the ATECC608A chip. Since ATECC608A performs the crypto operation inside the chip itself and the private key never leaves the chip, a malware inside the device will not be able to read the private key. Students will be required to connect the ATECC608A enabled device to a MQTT server or AWS IoT through SSL/TLS.

### 3.9 Developing Case Studies

Case studies show how a theory or concept is applied to real situations. This method requires critical thinking and analysis, allows students to synthesize course contents, encourages active learning, provides an opportunity for development of key skills such as communication, teamwork and problem solving, and increases the students' enjoyment of the topic and hence their desire to learn.

**Attack Case: Vulnerabilities of Air Quality Monitoring Networks.** We will use the exploit of PurpleAir as a case study [14]. PurpleAir sensors are based on an early version of ESP32 - ESP8266, which does not have security features of ESP32. The measurements of air quality metrics such as PM2.5 are sent to PurpleAir's servers, which show the air quality measurements on Google Map. We explored the system architecture and its communication protocols based on traffic analysis using mitmproxy [14], which is an https proxy tool. We find that the system adopts unencrypted communication and uses MAC addresses to identify sensors in the sensor data sent to the servers. This practice allows us to "pollute" sensor data by conducting a man-in-the-middle (MITM) attack or by sending fabricated data along with a victim sensor's MAC address to the servers. The servers also allow us to check if a specific MAC address exists in the system. This enables us to enumerate all valid MAC addresses of PurpleAir sensors and potentially pollute data from every sensor deployed globally.

We plan to replicate the vulnerabilities of PurpleAir and implement such a vulnerable system on ESP32. Students can play with the system and experiment on man-in-the-middle attack, spoofing attack, device scanning attack, and other ethical hack labs presented in this paper.

**Defense Case: Secure Air Quality Monitoring Networks.** In this case, students will be required to design a secure air quality sensor and monitoring network and evaluate the pros and cons of various secure measures.

Secure boot should be used to prevent the manipulation of the firmware of the sensor. With secure boot, if the firmware is changed, the sensor will not boot. Such a firmware is trustworthy to some extent. Flash encryption should be used to protect sensitive data on the flash, including the WiFi credentials. Certificate based mutual authentication with TLS should be used to defeat the MITM attacks and protect the communication. The mutual authentication renders the MITM attack invalid and the hash of the sensor's public key can be adopted as the device ID if needed.

Secure storage should be used to store the sensor's private key so that the adversary cannot obtain the private key. Students should realize that a per-device private key is needed. Otherwise, if all devices use the same private key and it is compromised, all these devices will be affected. The location of the sensor can be obtained from either a GPS module on the sensor or WiFi localization. A GPS module can be problematic since a dedicated adversary may replace the GPS module with an artificial one. In addition, the GPS may not work inside buildings. The WiFi localization may be more appropriate since the trusted firmware will retrieve the WiFi information for the purpose of localization. However, students should realize that attackers may deploy rogue access points to mislead the WiFi localization strategy. The server may also validate the reported location from the device via the IP location service [12], which finds the geolocation of a sensor from the IP address of the sensor while the accuracy of the IP location service is limited [2]. Secure firmware upgrade is needed in case that vulnerabilities are found in the system.

## 4 Conclusion

In this paper, we propose to develop effective, engaging and novel teaching materials on IoT hardware security, operating system/firmware/software security, network security, and data security with the low-cost IoT kit and IDE. Achieving the proposed objectives will lead to an increased capacity in producing cybersecurity professionals. This will be demonstrated by the outcomes in the following two aspects. *Curricula:* (i) New IoT platforms for cybersecurity education, (ii) transferable modules that will be developed and incorporated into curricula at the participating institutions, and (iii) increased IoT security components in interdisciplinary courses. *Students:* (i) Increased student interest in cybersecurity, (ii) improved knowledge and skills in IoT security, (iii) increased exposure of minority students to IoT and cyber security, (iv) increased employment perspective of students in cyber security, (v) increased number of students in IoT security research, (vi) increased student publications, and (vii) increased collaboration among students in the participating institutes.

**Acknowledgements.** This work was supported in part by NSF grants 1915780, 1931871, 1916175, 1802701 and 1643835. Mr. Jacob Crain is supported through the

REU Supplement of the NSF project (NSF 1802701). Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## References

1. Arduino IDE. https://www.arduino.cc/en/Main/Software. Accessed Nov 2018
2. Center for applied Internet data analysis. Internet protocol address (IP) geolocation bibliography. http://www.caida.org/projects/cybersecurity/geolocation/bib/. Accessed Nov 2018
3. Eclipse Mosquitto, an open source MQTT broker. https://mosquitto.org/. Accessed Nov 2018
4. Hello world with ESP32 explained. https://exploreembedded.com/wiki/Hello_World_with_ESP32_Explained. Accessed Nov 2018
5. mitmproxy is a free and open source interactive https proxy. https://mitmproxy.org/. Accessed Nov 2018
6. Open on-chip debugger. http://openocd.org/. Accessed Nov 2018
7. Readme for mbed tls. https://github.com/ARMmbed/mbedtls/tree/master. Accessed Nov 2018
8. WolfSSL introduction. https://github.com/espressif/esp-wolfssl. Accessed Nov 2018
9. The secretary of commerce and the secretary of homeland security, a report to the president on enhancing the resilience of the Internet and communications ecosystem against botnets and other automated, distributed threats (January 2018). https://www.ntia.doc.gov/files/ntia/publications/eo_13800_botnet_report_for_public_comment.pdf
10. Amazon Web Services Inc.: AWS IoT. https://aws.amazon.com/iot/. Accessed Nov 2018
11. Antonakakis, M., et al.: Understanding the Mirai botnet. In: Proceedings of the 26th USENIX Security Symposium (Security) (2017)
12. Brand Media, Inc.: Where is geolocation of an IP address? https://www.iplocation.net/. Accessed Nov 2018
13. Columbus, L.: IoT market predicted to double by 2021, reaching $520b (August 2018). https://www.forbes.com/sites/louiscolumbus/2018/08/16/iot-market-predicted-to-double-by-2021-reaching-520b/
14. Cortesi, A., Hils, M., Kriechbaumer, T.: mitmproxy: A free and open source interactive https proxy. https://mitmproxy.org/. Accessed Nov 2018
15. Dhanjani, N.: Security evaluation of the philips hue personal wireless lighting system (2013). http://www.dhanjani.com/docs/HackingLighbulbsHueDhanjani202013.pdf
16. Espressif: Arduino core for ESP32 WiFi chip. https://github.com/espressif/arduino-esp32. Accessed Nov 2018
17. Espressif: Build and flash with Eclipse IDE. https://dl.espressif.com/doc/esp-idf/latest/get-started/eclipse-setup.html. Accessed Nov 2018
18. Espressif: ESP-IDF programming guide. https://docs.espressif.com/projects/esp-idf/en/latest/. Accessed Nov 2018
19. Espressif: ESP-TLS. https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/protocols/esp_tls.html. Accessed Nov 2018
20. Espressif: ESP32 overview. https://www.espressif.com/en/products/hardware/esp32/overview. Accessed Nov 2018

21. Espressif: JTAG debugging. https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/jtag-debugging/. Accessed Nov 2018
22. Espressif: Secure boot. https://docs.espressif.com/projects/esp-idf/en/latest/security/secure-boot.html. Accessed Nov 2018
23. Jin, Y., Hernandez, G., Buentello, D.: Smart nest thermostat: a smart spy in your home. In: Proceedings of the Black Hat USA (2014)
24. Ling, Z., Luo, J., Xu, Y., Gao, C., Wu, K., Fu, X.: Security vulnerabilities of Internet of Things: a case study of the smart plug system. IEEE Internet Things J (IoT-J) **4**(6), 1899–1909 (2017)
25. Ling, Z., Liu, K., Xu, Y., Jin, Y., Fu, X.: An end-to-end view of IoT security and privacy. In: Proceedings of the 60th IEEE Global Communications Conference (Globecom) (December 2017)
26. Lohrke, H., Tajik, S., Boit, C., Seifert, J.-P.: No place to hide: contactless probing of secret data on FPGAs. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 147–167. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_8
27. Lohrke, H., Tajik, S., Krachenfels, T., Boit, C., Seifert, J.P.: Key extraction using thermal laser stimulation. Proc. IACR Trans. Cryptogr. Hardw. Embed. Syst. **3**, 573–595 (2018)
28. Luo, L., Zhang, Y., Pearson, B., Ling, Z., Yu, H., Fu, X.: On the security and data integrity of low-cost sensor networks for air quality monitoring. Sensors **18**(12), 4451 (2018)
29. martinius96: ESP32-eduroam. https://github.com/martinius96/ESP32-Eduroam. Accessed Nov 2018
30. Microchip Technology Inc.: ATECC608A. https://www.microchip.com/wwwproducts/en/ATECC608A. Accessed Nov 2018
31. Molina, J.: Learn how to control every room at a luxury hotel remotely. In: Proceedings of DEFCON (2014)
32. Obermaier, J., Hutle, M.: Analyzing the security and privacy of cloud-based video surveillance systems. In: Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS) (2016)
33. Park, J., Xu, X., Jin, Y., Forte, D., Tehranipoor, M.: Power-based side-channel instruction-level disassembler. In: Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (2018)
34. sylvainpelissier: JTAG debugging with bus pirate and OpenOCD (May 2014). https://research.kudelskisecurity.com/2014/05/01/jtag-debugging-made-easy-with-bus-pirate-and-openocd/
35. tedwood: Using eclipse with OpenOCD to build and debug ESP32 (Apr 2017). https://www.esp32.com/viewtopic.php?t=336&start=10
36. Vasile, M.J., Niu, Z., Nassar, R., Zhang, W., Liu, S.: Focused ion beam milling: depth control for three-dimensional microfabrication. J. Vac. Sci. Technol. B Microelectron. Nanometer Struct. Process. Meas. Phenom. **15**(6), 2350–2354 (1997)
37. Williams, K.C., Williams, C.C.: Five key ingredients for improving student motivation. Res. High. Educ. J. **18**(12), 104–122 (2011)