PUF Based Secure Framework for Hardware and Software Security of Drones

Vishal Pal
DESE*
Indian Institute of Science
Bangalore, India
vishalpal@alum.iisc.ac.in

B Srikrishna Acharya
RBCCPS**
Indian Institute of Science
Bangalore, India
bacharya@iisc.ac.in

Somesh Shrivastav
DESE*
Indian Institute of Science
Bangalore, India
shrivastavk@alum.iisc.ac.in

Sourav Saha
DESE*
Indian Institute of Science
Bangalore, India
souravsaha@alum.iisc.ac.in

Ashish Joglekar
RBCCPS***
Indian Institute of Science
Bangalore, India
ashishj@iisc.ac.in

Bharadwaj Amrutur
RBCCPS**, ECE***
Indian Institute of Science
Bangalore, India
amrutur@iisc.ac.in

Abstract—Unmanned Aerial Vehicles (UAVs) augmented with sensors like cameras, GPS and LiDARs, shows tremendous potential in delivering Internet of Things (IoT) services from great heights. Current security challenges in the drone deployment for such applications are: software and hardware breaches, identification of rogue drones and link hijack avoidance. In this work, we extend the secure framework for not only software, but also for hardware breaches and identification of compromised drones using a Trusted Execution Environment (TEE) created by ARM's TrustZone technology and Physically Unclonable Function (PUF). The PUF based unique fingerprints of on-board sensors, flight controller and companion computer of the drone are implemented in a Field Programmable Gate Array (FPGA). The PUF based drone authentication algorithm has been implemented and demonstrated in a proof of concept system and can be incorporated in drone deployments.

Index Terms—Physically Unclonable Function, fingerprint, drones, FPGA, ring oscillator, authentication, IoT

I. INTRODUCTION

Drones are aircraft that can be operated either by remote control or autonomously using on-board computers. These drone systems are vulnerable to attacks that target both the cyber and physical elements, the interface between them, the wireless link, or even a combination of multiple components. Government regulations, makes registration of drones mandatory and holds the operators of drones accountable. The majority of the illegal usage of drones will be achieved via their hostile acquisition, which can be established through exploiting either cyber or physical vulnerabilities. These drones are susceptible to software espionage and data manipulation. The security breaches like tampering with boot loader, kernel, firmware of the on-board electronics like flight controller, companion computer and other sensors, may lead to change of behavior of the drone and compromises the mission. Another potential vulnerability is changing the on-board sensors like GPS, RF Modules, LiDAR, camera and flight controller or the companion computer, which also includes attacks like spoofing of the GPS, LiDAR and other sensors. Any tampering in both on-board hardware and software components may lead to undesired behavior of the drones [1]. Thus, there is a pressing requirement to detect these breaches in both software and hardware and authenticate every component associated with the system.

In this paper, we propose a PUF based security framework for drones that will require each hardware of the drone to have an inbuilt PUF logic. For the proof of concept, the PUF logic is implemented using an FPGA. Unlike conventional PKI based drone authentication protocols, the proposed framework doesn’t need to store cryptographic keys on-board. The proposed framework is scalable and can be extended to secure drone software, hardware and all the on-board sensors.

II. RELATED WORK

In this section, we provide a summary of existing authentication frameworks based on PUF designs. For example, Suh et al. [2] proposed PUF designs for low-cost authentication of individual ICs and generate volatile secret keys for cryptographic operations by exploiting inherent delay characteristics of wires and transistors that differ from chip to chip. A ring oscillator based PUF design was proposed by using 1-out-of-8 masking scheme which effectively chooses the ring oscillator pairs whose base frequencies are far apart. On the other hand, Aysu et al. [3] demonstrated a prototype implementation of SRAM PUF based secure protocol that supports privacy-preserving mutual authentication between a server and a resource constrained device. Recently Mahmud et al. [4] proposed a resource-efficient PUF based security protocol to ensure both software and hardware integrity of IoT devices along with a mutual authentication scheme for the resource...

PUFs are functions that extract a unique signature of an IC, leveraging randomness of the manufacturing process. This signature can be used as device fingerprint. To ensure the trustworthiness of the secure system, authentication mechanisms leverage features of PUFs that allow them to produce specific outputs for integrated devices and that cannot be physically copied. Literature works like of Armin et al. [6] does a thorough analysis of the strengths and weakness of different PUF architectures including Arbiter PUF, Ring Oscillator PUF, SRAM PUF and a few of the latest architectures. Detailed analysis of the strength and weakness of different PUF architectures are summarized in the Table I. We have used Ring Oscillator architecture to create an authentication mechanism since it is well suited for FPGA implementation.

Usage of PUFs in device authentication schemes in IoT domain has a serious drawback in terms of the need to store challenge and response pairs on the verification agent side which can be compromised by an attacker. The protocol proposed by Mahmud et al. [4] doesn’t store a large set of challenge-response pairs at verification agent’s database and hence provides scalability. Our work is based on the security protocol proposed by Mahmud, further scaling it for hardware security of avionics of a drone. We further utilise Arm TrustZone to create a root of trust in an Trusted Execution Environment (TEE).

<table>
<thead>
<tr>
<th>PUF Architecture</th>
<th>Strength</th>
<th>Weakness</th>
<th>FPGA Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbiter PUF</td>
<td>Partially resilient against machine learning attacks</td>
<td>Delay path must be identical</td>
<td>Cumbersome (ASIC suitable)</td>
</tr>
<tr>
<td>Ring Oscillator PUF</td>
<td>Easy to implement and resource efficient</td>
<td>Environmental sensitivity</td>
<td>Yes</td>
</tr>
<tr>
<td>SRAM PUF</td>
<td>Good statistical properties</td>
<td>Low number of CRPs</td>
<td>No</td>
</tr>
</tbody>
</table>

TABLE I: PUF architecture summary [6]

III. PUF IMPLEMENTATION AND DEVICE FINGERPRINT EXTRATION

A. Ring Oscillator PUF FPGA Implementation

Ring Oscillator (RO) PUF was implemented on Digilent Cmod A7, breadboard friendly 48-pin DIP form factor board built around Xilinx Artix-7. For the purpose of demonstration, RaspberryPi (RPi) was used as companion computer and Pixhawk2 was used as flight controller. However, the design procedure can be easily ported to any Single Board Computer (SBC) like NVIDIA Jetson TX2.

RPi sends a 8 bit challenge to the RO PUF logic running on Cmod A7 board. PUF logic executes and a 8 bit response is sent back to RPi. There are several designs for RO PUF logic, in our design to capture more randomness, each RO has 5 inverters and a total of 8 sets of ROs were developed. Each set, contains two subsets of ring oscillators RO-1.1 and RO-1.2 as shown Figure 1. Each subset of ring oscillator i.e RO-1.x further contains 16 ROs. Challenges of 8 bits were given which selects two ROs from each set. Minute differences in the silicon layers of the semiconductor device caused by the manufacturing process results in slight differences between selected RO frequencies. There frequencies were compared by counters and the response of the PUF is ‘1’ if the first RO is faster else the response is ’0’. As shown in the Figure 1 Master Controller (MCNT) triggers the counters CNT-1 and CNT-2. On arrival of enable signal, both the counters starts counting the rising edges generated by their respective ring oscillators. Master controller stops these two counters after a fixed time interval. The two counters are subsequently compared and the response is ‘1’ if CNT-1 is greater than CNT-2 else ’0’.

Fig. 1: Ring Oscillator design implemented on FPGA

Eight bit response is obtained with each set of RO contributing one bit to the response. The PUF logic uses I2C slave module which is also implemented in VHDL to interface with the RPi. We have used Vivado tool for the design. Timing report and resource utilization are as shown in Table II and III. Power reported by Vivado for the implemented design was 0.075 W (0.072W static and 0.003W dynamic).

<table>
<thead>
<tr>
<th>Clock Speed</th>
<th>Worst Negative Slack (WNS)</th>
<th>Worst Hold Slack (WHS)</th>
<th>Worst Pulse Width Slack (WPWS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 MHz</td>
<td>8.721 ns</td>
<td>0.172 ns</td>
<td>9.500 ns</td>
</tr>
</tbody>
</table>

TABLE II: Timing report generated by Vivado tool for RO implementation

<table>
<thead>
<tr>
<th>Resource</th>
<th>Utilization</th>
<th>Available</th>
<th>Utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT</td>
<td>954</td>
<td>20800</td>
<td>4.59</td>
</tr>
<tr>
<td>FF</td>
<td>324</td>
<td>41600</td>
<td>0.78</td>
</tr>
<tr>
<td>IO</td>
<td>4</td>
<td>106</td>
<td>3.77</td>
</tr>
<tr>
<td>BUFG</td>
<td>1</td>
<td>32</td>
<td>3.13</td>
</tr>
</tbody>
</table>

TABLE III: Utilization report generated by Vivado tool for RO implementation

B. Device Fingerprint Extraction

In this section we discuss the methodology used to extract fingerprint of all the peripherals of the drone. This involves interfacing of Cmod A7 FPGA board to the device in context, where the FPGA board provides both the device fingerprint
and relays the data to flight controller without altering existing functionality. Using open source PX4 firmware, we have implemented a driver to enable communication between FPGA board and Pixhawk2 flight controller. As a proof of concept, we have also modified the existing LiDAR device driver to interface with FPGA board for authentication process. This can be extended to any drone peripheral.

1) Fingerprint of Companion Computer: To establish communication between companion computer (RPi) and Cmod A7 board, an I2C interface was implemented. I2C slave driver was written in VHDL and runs on Cmod A7 FPGA board. RPi runs I2C Master.

![Setup to obtain fingerprint of RPi](image)

Figure 2 shows the interface between FPGA board and RPi. For a given 8-bit challenge, the PUF logic is executed and FPGA board responds with a 8-bit response. This setup was further used to perform PUF profiling to obtain the “Unique” responses for device enrollment and authentication. PUF profiling involved identifying the stable challenge-response pairs with the variation in ambient temperature and voltage.

A total of 256 different challenges were given to two Cmod A7 FPGA boards. Responses of each board were logged for 72 hours. Responses to the challenges which didn’t change during this period were earmarked as “Stable Responses”. To obtain “Unique Responses”, “Stable Responses” of both the boards were compared. A total of 74 “Unique Responses” were obtained as shown in Table IV. Further, only these “Unique Responses” were used during device enrollment and authentication phases.

<table>
<thead>
<tr>
<th>Total Challenges</th>
<th>Stable Responses</th>
<th>Weak Responses</th>
<th>Unique Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board A</td>
<td>256</td>
<td>117</td>
<td>139</td>
</tr>
<tr>
<td>Board B</td>
<td>256</td>
<td>94</td>
<td>162</td>
</tr>
</tbody>
</table>

TABLE IV: Results of PUF profiling

2) Fingerprint of Flight Controller: To authenticate Pixhawk2’s integrity, interfacing of Cmod A7 board to it is required. Since, Pixhawk2 integrates with most of the sensors over I2C, it was decided to interface Cmod A7 over I2C as well. However, this required modifying existing I2C driver to seamlessly interface the sensors as discussed before. The interfacing architecture is as given in Figure 3.

The modified firmware can send challenges to Pixhawk2 from GCS via RPi over MAVLink protocol [7]. Which is then published over uORB topic [8] and sent to FPGA board over I2C interface. FPGA board (connected to Pixhawk2) runs PUF logic, and publishes response over uORB topic and then subsequently sent back to GCS over MAVLink. Received responses are then verified against the existing database at GCS for authentication process.

3) Fingerprint of Sensors: In the proposed security framework, it is assumed that each sensor on-board the drone has PUF logic implemented. For the purpose of demonstration, LiDAR (LiDAR Lite V3 by Garmin) was chosen as the sensor to which Cmod A7 FPGA board will be interfaced. However, this demanded modifications to the existing device driver of LiDAR in the PX4 firmware. The interfacing architecture is as given in Figure 4.

![Setup to obtain fingerprint of LiDAR](image)

The response received at GCS doesn’t match the responses in database, then the sensor, in this case LiDAR, is considered as compromised or not functioning. A “LiDAR Stop” message is sent to Pixhawk2, subsequently, appropriate action may be taken (continue or abort mission) depending on the mission requirements. In this case, we have configured Pixhawk2 to stop using compromised LiDAR as sensor and keep the drone flying with barometer as altitude source.

IV. PROPOSED SECURITY FRAMEWORK

In this framework, every sensor module/ peripheral of the drone has a UUID realised by embedding a PUF in the appropriate chip associated with the module. A layout of the peripherals of the drone is as given in Figure 5.

![Layout of different modules on-board the drone](image)
All the red marked peripherals need to have their UUID implemented using PUF. Secure boot enables root of trust by detecting tampering to the bootloader, kernel, firmware or the mission applications. For example, secure boot of companion computer can be implemented using ARM’s TrustZone technology. Before discussing the proposed framework, we will introduce a few terminologies that will be referring to throughout the paper:

A. Key Terminologies:
- **Base Station (BS):** Ground based controller, to control a network of drones in the area of deployment.
- **Device:** An IoT device, like flight controller and companion computer.
- **Drone Identity Provider (DIP):** DIP stores challenge response pairs as well hash values of software in a database. Its role is to query the drone for authentication.
- **Domain Security Manager (DSM):** DSM provides a drone access to the network and it interacts with DIP to verify a drone’s identity.
- **Cipher-based Message Authentication Code (CMAC):** CMAC is a block cipher-based message authentication code algorithm. It will be used to provide assurance of the authenticity and, hence, the integrity of binary data. We will write CMAC (plain data, key) to refer to signed data.

Authentication process is divided into two phases: enrollment phase and device authentication phase.

B. Enrollment Phase
1) Enrollment phase is performed by the manufacturer before deployment.
2) A device is given a challenge (c) and it generates a response (r). Every device is given the same challenge (c). However, each device produces a unique response (r).
3) Very first response (r) is used as a device identifier.
4) Response (r) is again fed to the device’s PUF as a new challenge (c(i)) and device generates new response (r(i)).
5) Device stores (c(i)) to its non volatile memory as its unique device ID.
6) The challenge response pair (c(i), r(i)) is securely transmitted to the DIP over a TLS/SSL link. The DIP stores the (c(i), r(i)) in its challenge-response pair (CRP) database. DIP also stores the hash of the software (Hsw) installed on the device. Figure 6 shows above steps in detail.

C. Authentication Phase
1) DIP sends Authentication request to the drone.
2) Companion Computer (CC) fetches challenge from memory, executes its PUF logic and calculates software’s hash value (Hsw). It also generates a new challenge using a Public Random Number generator function (F). Response to new challenge is generated by the PUF logic. A Nonce (N(i)) is also added in the intermediate response (u(i)) to avoid any possibilities of replay attack. Signed information (achieved using CMAC with r(i) as key) is sent to DSM which forwards it to DIP.
3) DIP, verifies the received information for hardware and software authenticity by comparing it with the information already available in the database.
4) Subsequently, a signed (CMAC based) value of new challenge and hash value of response (H(r(i))) is sent to DSM. Post integrity checks are initiated to ensure authenticity of DIP and DSM.
5) Companion computer initiates authenticity checks of flight controller and sensors in similar fashion.
6) Compromised hardware and software are flagged immediately at the DIP. Figure 7 and Figure 8 shows above steps in detail.

D. Threat Analysis
- **Eavesdropping:** Attacker can not attack communication channel between DSM and DIP, as it’s a secured link. Channel between Drone and DSM is untrustworthy. Attacker has access to c(i), Hsw, r(i) XOR r(i+1), CMAC(u(i), r(i)) and CMAC(c(i+1), r(i+1)). However, attacker can’t learn r(i) and r(i+1) as drone sends XORed values of r(i) and r(i+1). Attacker doesn’t have knowledge of r(i) since it can’t clone drone’s PUF circuitry. Therefore, can not learn r(i+1). Also, it is not possible to know r(i+1) from CMAC(c(i+1), r(i+1)). An attacker can only recover earlier response (r(i)) when he has physical access to PUF and he taps the I2C interface between PUF chip and device/sensor. Since, we are proposing PUF as ASIC and integrated part of device/sensor, it requires special skills and tools to tap the I2C interface. Though it can’t be ruled out but it is cumbersome to achieve. Having an encrypted I2C interface between PUF ASIC and device may avoid such an attack.
- **Replay Attack:** Since the authentication request contains a random nonce N(i) from the drone, attacker can’t fool DIP by simply replaying a message.
• **Data Integrity**: There is a possibility that an attacker can modify in-transit message from drone to DSM to imitate a real drone. However, CMAC(u(i), r(i)) is used to verify message integrity which is a signed message and hence can’t be decrypted.

• **Memory Mapping**: Only the challenge c(i), is kept in the non-volatile memory. Attacker can tamper with the memory to extract c(i). However, it is not possible to generate response bit using the compromised c(i) since PUF circuit of the drone is unclonable. Hence, attacker can not decrypt r(i) XOR r(i+1).

• **Cloning Attack**: Attacker can capture drone and copy embedded software and challenge to another malicious device. However, proposed design is based on PUF which is physically unclonable.

• **Software Tampering**: Any tampering with the software/codes will produce different hash value and hence will be flagged at the DIP. In case of Pixhawk2, hash value of all the parameters is calculated and hence any tampering of the parameter values will also be flagged at the DIP making it more secure.

V. DEMONSTRATION SETUP AND RESULTS

Drone enrollment and authentication algorithms were implemented in Python language. Figure 9 shows the setup wherein Drone, DIP and DSM communicates with each other to achieve drone authentication.

A. Hardware Setup

Drone authentication scheme is demonstrated using the lab setup as shown in Figure 10. Every drone peripheral is augmented with Cmod A7 FPGA board.

B. Software Setup

The software setup consists of three terminal interfaces: DIP interface, DSM interface. A server is started both at DSM and drone ends. DIP runs a console for the operator to send authentication request to the drone. As shown in Figure 11, DIP runs drone authentication console. The operator can select the drone serial no. to be authenticated from the drop
Fig. 10: Drone authentication framework lab demonstration setup

down menu and then send "Authentication Request" to verify the hardware and software integrity of drone. The operator can also authenticate all deployed drones at one go. Drone authentication status window shows the status of the various peripherals including companion computer, flight controller and sensors like LiDAR, GPS, camera and RF module. In case any peripheral is compromised, it is flagged in red.

Fig. 11: Drone authentication control center console

C. Test Cases

1) Sending authentication request: DIP sends an authentication request to the drone. After receiving authentication request, drone executes its PUF logic of companion computer and flight controller and hash value of the software running on companion computer and flight controller. This information is sent back to the DIP. DIP verifies the received information with its database. If found correct, drone is authenticated. During this process, integrity of DIP and DSM is also checked by drone to ensure it is communicating with the authenticated DIP and DSM.

2) Hardware attack on companion computer and flight controller: We swap the FPGA boards of companion computer and flight controller to simulate the case when an adversary attacks the hardware of both these peripherals. In this case, the PUF logic responses received from the drone doesn’t match with the database of DIP and hence hardware integrity is flagged as compromised at console. Since software were not tampered, their authenticity is still ensured.

3) Software attack on flight controller: We restore the FPGA boards but we modify one of the parameter values of PX4 firmware in QGroundControl to simulate the case when an adversary tries to change the parameter to modify desired behaviour of the drone. In this case, PUF logic responses received from the drone are found as expected. However, the hash value of the software running on Pixhawk2 doesn’t match with the database at DIP and hence is flagged as compromised on console.

VI. Conclusion

Drones as a platform requires integration of several Cyber-Physical Systems (CPS) aspects like sensing, computation, control, networking, and physical processes. Drones deployed for surveillance operations work as a network of drones continuously relaying information to each other and GCS. At the time of field deployment of these drones, hardware and software integrity is paramount. Any breaches or manipulation of the sensor readings may results in catastrophic events. An adversary may also tamper the flight parameters and autonomous mission algorithms leading to undesired behaviour of the drone. Hence an authentication protocol for hardware and software entity of the drone is the need of the hour. Our work has focused on proposing a Physically Unclonable Function (PUF) based drone authentication protocol. We have demonstrated the proposed authentication protocol to authenticate companion computer, flight controller and LiDAR. We recommend that the RO PUF architecture used in this work for proof of concept of the proposed authentication scheme may be replaced with more stable PUF architectures like Arbiter PUF. The proposed framework is scalable and can be extended to authenticate other sensors as well.

ACKNOWLEDGMENT

We thank the Indian National Academy of Engineers for funding support via the Abdul Kalam Technology Innovation Fellowship (2018-2021) to Bharadwaj Amrutur. We also thank Robert Bosch Center for Cyber Physical Systems at IISc, for funding support as well as access to its laboratory facilities for carrying out this work. We thank Prof. Vinod Ganapathy of CSA dept., Prof. Kuruvilla Varghese and Prof. T V Prabhakar of DESE dept for their advise and inputs.

REFERENCES