Accelerating PUF-based Authentication Protocols Using Programmable Switch

Divya Pathak
IIT Hyderabad
Email : cs21mtech12009@iith.ac.in

Ranjitha K
IIT Hyderabad
Email : cs21resch01002@iith.ac.in

Krishna Sai Modali
BITS, Pilani
Email : h20210110@pilani.bits-pilani.ac.in

Praveen Tammana
IIT Hyderabad
Email : praveent@cse.iith.ac.in

Antony Franklin A
IIT Hyderabad
Email : antony.franklin@cse.iith.ac.in

Tejasvi Alladi
BITS, Pilani
Email : tejasvi.alladi@pilani.bits-pilani.ac.in

***Abstract*—Many IoT use cases have ultra-low latency and strong security requirements. But achieving both simultaneously is challenging. In this paper, as a use case, we consider the authentication of IoT devices for every transaction and develop a fast and secure authentication protocol. Our key idea is to leverage highly secure Physically Unclonable Functions (PUFs) and high-speed programmable switch and offload PUF-based authentication protocol to the switch. By doing so, it enables authentication of every transaction at network speed. In this paper, we demonstrate the feasibility of our idea by offloading the authentication protocol to a programmable switch with Tofino chip. Our preliminary experiments show that protocol offloading reduces authentication latency by 2-4 times and scales to a few hundred thousand IoT devices.**

I. Introduction

IoT applications have been deployed around the world at a rapid pace with use cases in the domains of industrial automation, intelligent transportation, telemedicine, virtual reality, and smart cities. A category of IoT devices is vulnerable to various attacks [1] that lead to security and privacy issues (*e.g.*, leakage of sensitive data, tampering, spoofing). To defend against such attacks, it is essential to deploy an authentication system that ensures only legitimate devices gain access to the network.

For session authentication in today's applications, classical cryptography-based techniques [2]–[4] are widely used. These techniques rely on a secret key, assumed to be stored permanently on the device or battery-backed storage and is unknown to an adversary. But in practice, the mechanisms used to store the key on the device memory (*e.g.*, non-volatile memory) are subjected to physical attacks [5], [6]. An attacker can use techniques to clone memory (*e.g.*, micro probing) or may use side-channel information to retrieve any information about the key leading to a security breach.

Instead of using a secret key stored in permanent storage, an alternative approach is to *generate* a key securely whenever an IoT device tries to establish a data session to a remote entity followed by using the key to secure the session. To realize this, Physically Unclonable Function (PUF) hardware-based [7], [8] device authentication and key generation schemes are promising. More specifically, PUFs operate on a challenge-response mechanism, that is, given an input stimulus called a challenge, it generates an output called a response. This response depends on physical factors like supply threshold voltage, temperature, gate level delay, power-on state, and many other physical characteristics. PUFs are unclonable mainly because even though the manufacturing process is the same among different Integrated Circuits (IC), each IC differs from the other because of variations present during the manufacturing process.

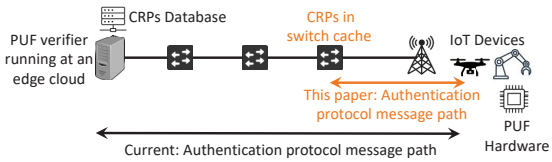


Fig. 1. **PUF-based authentication**

In short, to secure data sessions using PUF-generated keys, the IoT device and the destination it wants to speak to have to agree on a key. This is followed by using the key (perhaps in combination with other secret information) to secure the session. However, the key-sharing process adds an extra step: how to ensure that the key is indeed shared with the intended IoT device? This requires authentication of the IoT device initiating the session.

For authentication, many PUF-based authentication protocols [9]–[15] have been proposed. Fig. 1 summarizes the key idea behind these protocols. To be specific, consider the authentication functionality (*i.e.*, PUF verifier logic) running on a general-purpose CPU server at the edge. Before an IoT can access resources (*i.e.*, data, compute, network), (a) it sends an authentication request to the server, (b) then the edge server issues a challenge to the IoT, (c) followed by a response message from the IoT to the server, and (d) finally the verifier validates the response and sends an acknowledgement to the IoT device. We define end-to-end authentication request completion time as the time to complete these four steps. It includes (1) multiple RTTs and (2) packet copies and I/O interrupts at the server, especially while forwarding packets from the network interface to the hypervisor layer [16], [17]

to the application layer to PUF verifier.

Enabling PUF-based authentication in Ultra-reliable low latency communications (URLLC) applications is challenging because of additional delays (transmission, propagation, and processing) introduced by the protocol. Especially, in URLLC applications where latency is crucial as the time budget to finish a transaction (*i.e.*, authentication followed by data transmission) is expected to be less than 1 millisecond [18]–[20]. Some example communication scenarios are (1) machine-to-network-to-machine in industrial automation (*e.g.*, sensor-controller-actuator [21]), and (2) vehicle-to-network-to-vehicle in autonomous navigation (*e.g.*, drone-based delivery [18]).

The existing work [9]–[15] mainly focus on the security analysis (*e.g.*, security proofs) of proposed PUF-based authentication protocols, there are no insights into other important performance parameters such as latency, throughput, and their feasibility in 5G and edge computing environments.

In this paper , we take a step towards building a secure and fast PUF-based authentication system by leveraging features offered by the P4-based high-speed programmable switch data plane (*e.g.*, switch with Tofino chip [22]). Many recent works show performance benefits by offloading latency-sensitive tasks [21], [23]–[26] from CPUs to programmable switches. For instance, [21], [25], [26] improve sensor-actuator communication delays by offloading control actions from the industrial controller to a programmable switch. Inspired by these works, we offload an existing PUF-based authentication protocol to a high-speed programmable switch and reduce authentication completion time.

In this paper, we consider a typical private edge setup where IoT devices communicate with URLLC applications deployed at the nearest mobile-access edge compute (MEC). We assume that both IoT devices and the network connecting the devices to the edge cloud are under a single administrative domain (*e.g.*, industrial automation) and every transaction initiated by an IoT device should be authenticated before the data transmission starts.

We focus on three objectives: (1) reduce authentication completion time; (2) scale the offloaded authentication protocol to support a reasonably large number of IoT devices (hundreds of thousands of devices); and (3) secure the offloaded protocol from various attacks. However, programmable switches have limited memory (100's of MBs [22]) and allow a small set of per-packet operations (*e.g.*, hash, arithmetic, logical) with a small number of per-packet memory accesses (1-2 per pipeline stage). These constraints make it challenging to achieve these objectives. We address this challenge by carefully placing per-packet operations essential for implementing and securing the PUF-based authentication protocol. Also, to support the authentication for many IoT devices, we carefully split challenge-response pairs across multiple stages in the switch pipeline. Together, this approach significantly improves the number of IoT devices supported at a given time.

The main contributions of this paper are as follows:

- We carefully address the resource constraints imposed by P4-based programmable switches and implement a PUF-based authentication protocol prototype that supports authentication for hundreds of thousands of IoT devices. Code is available at https://github.com/networked-systems-iith/PUFAuth
- We also conduct a security analysis of the authentication protocol offloaded to the switch.
- We evaluate the prototype running on a real switch. We observe that authentication time is improved by 100-472% compared to the time taken by the protocol running on a general-purpose CPU server.

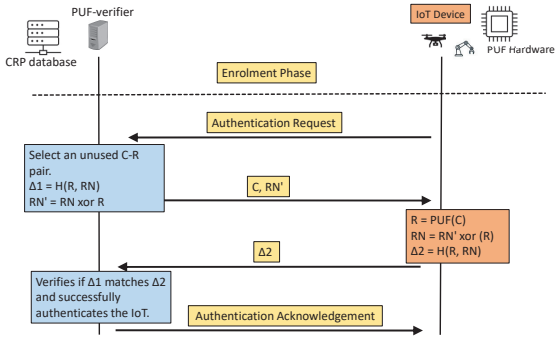


Fig. 2. **PUF-based authentication protocol workflow**

II. Motivation

PUF-based authentication protocol. Fig. 2 shows an existing PUF-based IoT device authentication protocol. The PUF-verifier logic is running on a server and IoT devices are equipped with PUF hardware. During the enrollment phase, numerous Challenge-Response Pairs (CRPs) of each IoT device are produced in advance and securely stored in the PUF-verifier database. This phase is executed once for each device. During the authentication phase, the verifier issues a challenge to an IoT device, followed by a response generated by the IoT device's PUF. The authentication is successful only if the IoT device's response matches the response at the verifier.

Transaction-level authentication. Consider a delay-sensitive transaction where sensor data from a PUF-equipped IoT device [27] is sent to a centralized industrial controller which analyzes the data and sends a control action to an actuator (the same or another IoT device) [21]. Assuming the controller is handling transactions of a few tens of thousands of IoT devices with one transaction per second from each device, this translates to a few tens of thousands of transactions per second. If such transactions have to be authenticated using PUF-based authentication protocol, then the authentication of each transaction by a central controller before accepting the data introduces additional delays (*i.e.*, at least two RTTs, multiple packet copies, and I/O operations at the edge server from NIC to hypervisor layer to VM). This approach not only increases the authentication time because of many hops but also needs more computing to scale to such a huge number of transactions.

Thus, reducing authentication latency enables us to meet transaction time budgets (*e.g.*, 1ms). But satisfying either low latency or security requirements may not be hard, but

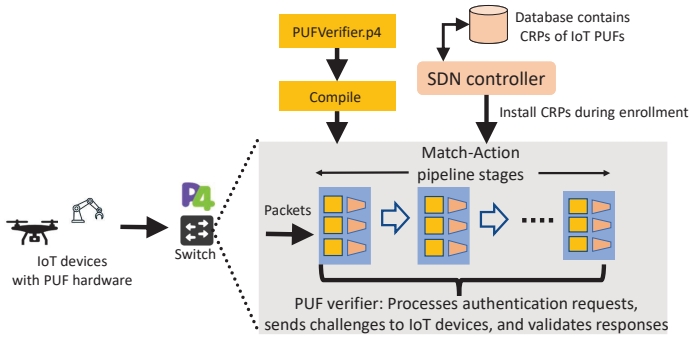


Fig. 3. **PUF-verifier offloaded to P4 switch**

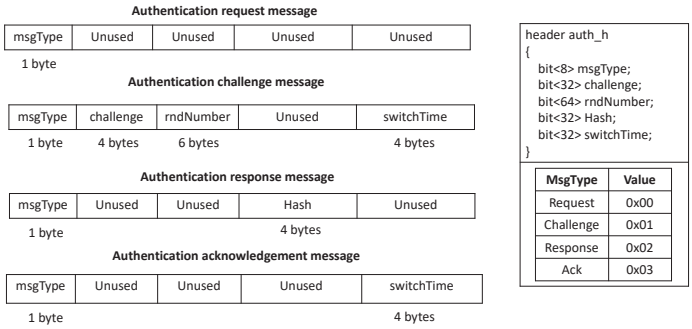


Fig. 4. **Authentication protocol header and message formats**

achieving both simultaneously is challenging. This motivates the need for accelerating PUF-based authentication protocols such that the transaction can be completed fast.

III. Acceleration of Authentication Protocol

We propose to offload PUF-verifier logic running on a general-purpose CPU to a programmable switch. We aim to meet the following objectives while offloading the authentication functionality to the programmable switch.

Reduce authentication latency. We aim to reduce the time to authenticate IoT device before initiating data transfer to a remote server.

Scalability. We aim to support the authentication for tens of thousands of IoT devices while satisfying the memory and processing constraints imposed by the programmable switch.

Security. We aim to ensure that the communication between IoT devices and the PUF-verifier running on the switch is secure and robust against various security attacks. This is challenging because of limited hash-based primitives and per-packet operations allowed by the programmable switch.

A. Protocol workflow

In the literature, there are many variants of PUF-based authentication protocols [9], [12], [15]. We have chosen an authentication protocol from the work done by Chattergee et. al. [11] to demonstrate its implementation feasibility on a programmable switch. Fig. 2 shows the workflow of the authentication protocol that we offload to the programmable switch. Fig. 3 summarizes our approach. PUF-based authentication primarily comprises two phases: the enrollment phase and the authentication phase.

1) Enrollment phase: We assume that IoT devices are equipped with PUF hardware (either in-built or pluggable [27]). Before deploying an IoT device, Challenge-Response Pairs (CRPs) of associated PUF hardware are either generated or provided by the IoT device vendor. We consider strong PUFs [28], [29] capable of generating a very large set of CRPs per IoT device. The retrieved CRPs are stored in a central database (CRP-DB) maintained at an edge cloud and then loaded to switch memory (CRP-Cache) by the SDN controller. The IoT device enrollment phase is assumed to happen in a secure offline manner.

2) Authentication phase: In this phase, IoT device gets authenticated according to the PUF-based authentication protocol shown in Fig. 2. To implement this phase, as shown in Fig. 4, we define protocol header ($auth\_h$) to support four types of protocol messages: Request, Challenge, Response, and Acknowledgement. An authentication message type is identified using *MsgType* field.

Request processing at the switch. On packet arrival, the switch ingress parser extracts the authentication header, checks whether the packet is a request message or a response message. If the message is an authentication request, the switch retrieves a challenge-response pair (challenge C and response R) associated with that IoT from the switch CRP-Cache. This is followed by computing a random number ($RN^{'}$) by xoring response (R) with another random number (RN). Also, $\triangle 1$, the hash of R and RN is stored in switch stateful memory. Finally, a challenge packet with C and $RN^{'}$ is sent to the IoT.

Challenge processing at IoT. From the challenge sent by the switch, the IoT device gets the corresponding response R from its PUF hardware. It also retrieves RN from $RN^{'}$ and R followed by computing hash ($\triangle 2$) of RN and R. Finally, the IoT device prepares a response packet with $\triangle 2$ and sends it to the switch.

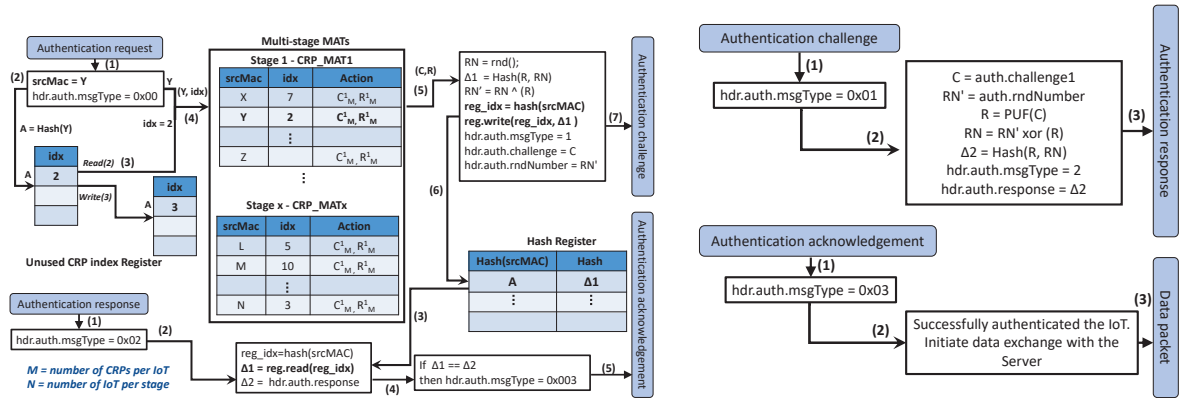
Response processing at the switch. On receiving the response packet, the switch retrieves the hash value ($\triangle 1$) stored previously in the stateful memory and compares $\triangle 1$ with the hash value ($\triangle 2$) in the response packet. If both are the same, then the switch will send an acknowledgment message to the IoT to indicate that the IoT is successfully authenticated.

Acknowledgement processing at IoT. On receiving an authentication acknowledgment message, the IoT device continues with the next steps in data transmission.

B. Technical challenges

*1) **Scaling to a large number of IoT devices**:* To meet our objectives mentioned earlier, we need to perform the operations involved in the authentication protocol under the memory and computation constraints imposed by the switch. In this section, we present our approach that can scale to support tens of thousands of IoT devices.

As shown in Fig. 5(a), on receiving authentication request from an IoT device, say X, corresponding authentication



(a) **Request packet and response packet flow at the switch**

(b) **Challenge packet and Acknowledgement packet flow at IoT**

Fig. 5. **PUF-based authentication protocol message flow at switch and IoT**

challenge packet is prepared and send to the IoT device. To do so, we need to take care of three major tasks which have to be executed sequentially due to data dependencies. The tasks are: (T1) retrieve currently unused CRP from the list of IoT device's CRPs; (T2) generate a random number and do operations like xor, bitwise shift, and concatenation; and (T3) compute a hash value ($\triangle 1$) and store it in a stateful register which will be retrieved and compared with ($\triangle 2$) in authentication response from the IoT device.

For T1, we program MATs such that each table entry matches on two header fields: (1) IoT device MAC address (srcMAC) to uniquely identify the IoT device, and (2) An index (idx) ranging from 1 to M pointing to the currently unused CRP from the list of IoT device's CRPs. Also, idx is incremented by one and put it back into the unused CRP Index Register, so that subsequent authentication requests know which are unused CRPs. To scale this approach to more IoT devices, we place per-IoT CRPs across multiple MATs in different stages in the switch pipeline. This allows using per-stage SRAM memory. We reserve the first pipeline stage for idx calculation and the remaining stages till the third last stage contain exact match-action tables with M CRP entries for each of N IoT devices. The second last stage is used to perform T2 and the last stage is used to perform T3.

This way, we implement the authentication protocol by leveraging SRAM available across multiple stages in the switch pipeline. Using tofino1 [22] switch, this approach scales to roughly 468K IoT devices per pipeline when M=100 CRPs per IoT and it is 884K IoT devices per pipeline using tofino2 [30]. The details on the resource consumption and how we arrived at this number are discussed in detail in the evaluation section V-E.

*a) **Discussion and future work**:* **Use switch memory efficiently.** Due to resource constraints, it is practically impossible to maintain all CRPs at the central database (CRP-DB) in switch memory (CRP-cache). When the CRPs of a particular IoT device in the CRP-cache get used up, the controller can replenish CRP-cache with a new set of CRPs. To achieve this, we can use a per-IoT counter to keep track of the number of used CRPs in CRP-cache. When this counter crosses a user-defined threshold, the switch can be programmed to send a notification to the controller which updates the CRP-cache with a new set of CRPs. However, if CRPs are exhausted before the update then the switch can be programmed to send authentication requests to a backup PUF-verifier running on a server. We expect the performance implication (*i.e.*, latency increase) would be minimal if the controller reacts fast to the switch notification. As part of our future work, we plan to extend the current offloaded protocol with this approach and evaluate the impact on performance for different workloads.

Scaling to more IoT devices. To support an even larger number of IoT devices, one can incorporate fast path and slow path architecture. In this architecture, a copy of the first request packet from an IoT device is sent to a centralized SDN controller and the original request is sent to a PUF-verifier running on a server (slow path). Next, the controller installs the IoT's CRPs on the switch such that subsequent requests from the same IoT device are processed entirely in the data plane (*i.e.*, fast path). This approach also requires an efficient eviction policy (for a better hit rate) when there is no space left in the switch memory. We foresee the current offloaded protocol can be extended with fast path and slow path architecture.

Choosing target network device. We implement the proposed approach on Tofino programmable switch because of its accessibility, open documentation, and community support. Our approach can be extended to other programmable switches such as Intel FlexPipe [31], Texas Instruments's Reconfigurable Match Tables (RMTs) [32], and Cavium XPliant switches [33]. Offloading to host-based solutions like DPDK/XDP/SmartNICs scale well and are more flexible compared to switch-based solutions. But this comes at the cost of an increase in latency (due to additional hops in the network as shown in Fig. 1 and management overheads (due to additional server infrastructure). In general, there is a tradeoff among scalability, latency, security, and management

overheads – achieving one of them is not hard, but achieving all simultaneously is challenging.

*2) **Security analysis**:* In this section, we first discuss a vulnerability considering the switch target limitations with respect to support for a secure hash function. We then define an attack model and analyze the robustness of the offloaded protocol against various security attacks.

*a) **Target specific vulnerability**:* Different P4 data planes support different hash functions offering varying levels of security. In this paper, we use Intel's Tofino [22] switch target which supports by default only CRC-32 hash which is proven to be reversible [34] and subject to a collision attack. Thus, one can guess the data which was used to generate the CRC-32 hash. Also, by launching a brute force attack, one can guess the random numbers used in the communication and the responses. To combat such vulnerability, we need to use a larger PUF response i.e., greater than 128-bit length is considered to be secure against brute force attacks with the current available compute resources.

One could also build on top of recent work, SipHash [35] that successfully implemented a more secure hash-based message authentication code (HMAC) on Tofino with a graceful impact on packet-processing latency. Another option is to program FPGA hardware available on the next generation of Tofino [36] and implement secure hash functions like SHA and export it using P4 externs. We plan to take this as future work.

*b) **Attack model**:* Consider a threat model containing PUF-verifier, an IoT device, and an adversary. The PUF-verifier and the IoT devices are secure and trusted, that is, the adversary cannot access the PUF embedded in the IoT device and the CRP database stored at the PUF-verifier. The PUF under consideration is robust, unclonable, unpredictable, and tamper-proof [37]. The adversary (Man-in-the-middle(MITM) attacker) has access to the communication medium between the PUF-verifier and the IoT device. The adversary can modify, delay, or drop messages exchanged between the PUF-verifier and the IoT device. Now we analyze the robustness of the offloaded protocol against well know attacks.

Replay attack. Assume that the adversary sniff and captures all packets exchanged during an authentication session with the goal to replay it later. If we carefully observe, the authentication challenge messages $C1$ cannot be repeated in subsequent authentication sessions since we make use of a strong PUF prototype. Furthermore, the use of random number RN' generated using R, and another random number RN as shown in Fig. 2, makes it inherently hard to launch replay attacks.

Tampering attack. Authentication messages are sent in plain text, which can be modified by an active Man-in-the-middle attacker sitting in between the PUF-verifier and the IoT device. Doing so will cause the IoT device not to be authenticated which may cause a denial of service to the IoT device.

Spoofing attack. The attacker will not be able to spoof a legitimate IoT device because to spoof an IoT device, the attacker needs access to the response corresponding to the challenge sent by the PUF-verifier. As the PUF embedded in the IoT devices cannot be accessed either remotely or physically by an attacker nor the PUF can be replicated, the only possibility of an attacker getting access to the PUF response is by trying a brute force attack on the exchanged random numbers and hash. Since we are using a response size of 128 bits or longer, this will not be possible. Therefore, our system is secure against spoofing attacks.

Denial of service attack. Our system is robust towards denial of service attack because the PUF-verifier process only the requests from already registered legitimate IoT devices. State information is stored per IoT device. Since we process only requests from legitimate registered IoT devices, for *N* registered IoT devices, P4-verifier will have at most N states irrespective of the number of authentication requests received. Thus the verifier continues to serve the IoT devices without any service disruption.

IV. Implementation

This section presents the implementation details of the PUF-verifier. We implement and execute the PUF-verifier logic on two targets: (1) Intel Tofino switch and (2) General purpose x86-based CPU. For the Tofino switch (Wedge100BF-32X), we implement the verifier logic in P4-16 [38] language in 617 lines of code. A snippet of the same is shown in Fig. 6. We store the CRPs in Match-Action Tables (MATs) as shown in Fig. 5(a)

On the x86 machine, we implement PUF-verifier logic using a multi-threaded User Datagram (UDP) socket program written in C++11 and the program contains 286 lines of code. We stored the CRPs in an array of struct type to access the CRPs in constant time. For calculating CRC32 hash, we use *boost* [39], a C++11 library to implement CRC hashes. To implement threading, we used *pthread* library [40]. To emulate IoT authentication requests, we generate custom requests and send them using a multi-threaded User Datagram (UDP) socket program written in C++11 and the program contains a total of 255 lines of code.

V. Evaluation

Our main goal for evaluation is to study 1) the latency and throughput improvements obtained by offloading the verifier logic to a programmable switch and 2) the overheads incurred in terms of data plane resources for executing the verifier logic in the data plane.

A. Experimental setup

To study the offloading benefits, we executed our experiments on two setups as shown in Fig. 7 and Fig. 8. In both setups, we have a host machine running an Ubuntu 18.04.6 LTS OS (on the left) sending authentication requests to the PUF verifier. The host machine is equipped with a Ryzen9 8-core 3.8 GHz CPU and dual-port 100 Gbps Netronome smartNICs [41] (used as regular network interface cards).

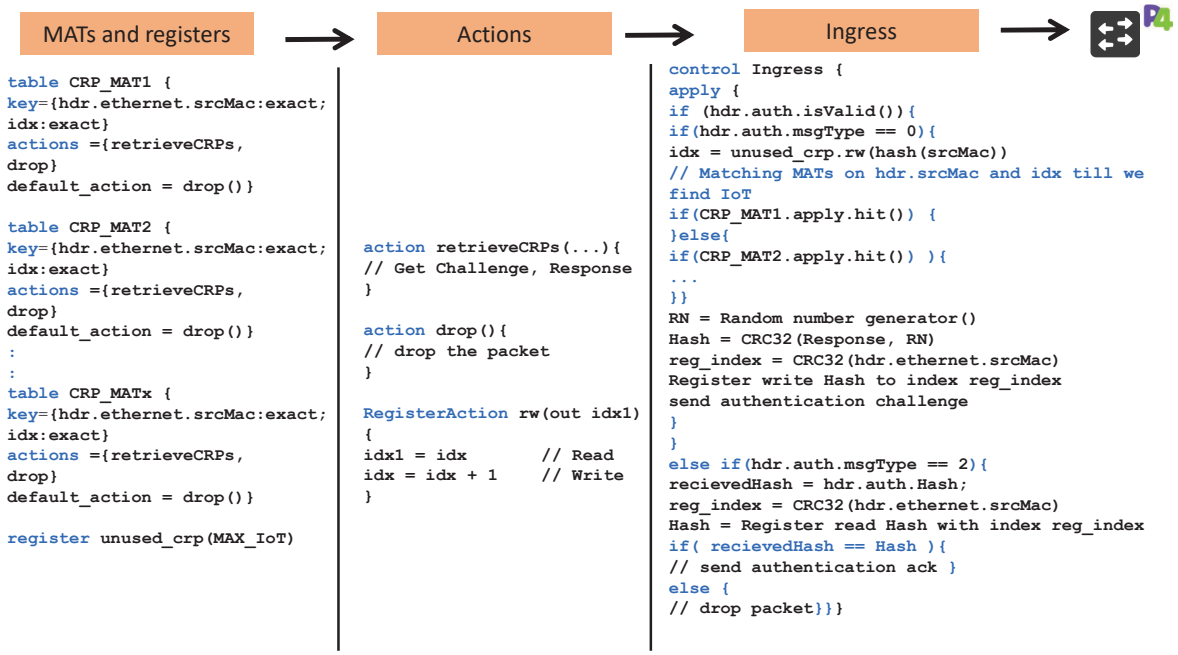


Fig. 6. **P4-16 code snippet of PUF-verifier logic**

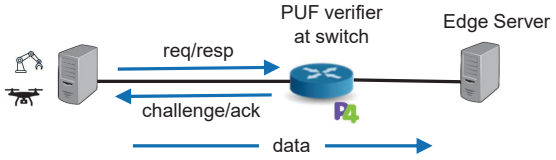


Fig. 7. **Host-Switch setup**

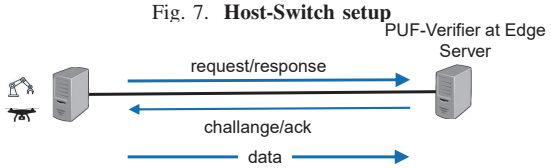


Fig. 8. **Host-Host setup**

Host-Switch. This setup (Fig. 7) comprises of the host machine connected to a Wedge100BF-32x Tofino [22] switch running an Ubuntu 18.04.6 LTS. One of the switch ports is connected to the host (left). The switch runs PUF verifier program(Fig. 6) written in P4 language.

Host-Host. The setup in Fig. 8 comprises two host machines, the one on the left issues authentication requests to the PUF verifier running on the other machine. We execute the PUF verifier on a machine running an Ubuntu 18.04.6 LTS OS, equipped with Intel(R) Core(TM) i9-7900X 10-core 3.3 GHz CPU and 100 Gbps Netronome smartNIC.

Experiments. We perform experiments by sending authentication requests to the PUF verifier. As discussed in section III-B2, we use weak hash CRC32 in the implementation due to Tofino constraints. We want to understand the performance implications and resource overheads as response size increases. For this, we consider different PUF response sizes of 64, 128, and 256-bits. We fix the PUF challenge size to 32-bit. In all experiments, we set the CRPs per IoT to 500. For each experiment, we captured pcap traces and analyzed the end-to-end latency, per-packet processing latency, and throughput variation.

B. End-to-end latency

To study the end-to-end latency, the host (left) generates 10K authentication requests. Fig. 9(a) shows CDF of authentication latency, that is, time to finish all steps in Fig. 2 for 64 and 256-bit PUF responses. For 64-bit PUF response, we observe 99% of the requests are finished in less than 0.25 milliseconds when the verifier is running on the switch (switch-based verifier) whereas it is up to 0.55 milliseconds when the verifier is running on the host (host-based verifier). Similarly, for 256-bit PUF response, 99% of the requests are finished in less than 0.4 milliseconds for the switch-based verifier whereas it is up to 1 millisecond for the host-based verifier.

It is also notable that the latency increase due to variation in PUF response size is much lesser (0.15ms) for the switch-based verifier when compared to the host-based verifier where the latency is increased to approximately 0.45ms. For the switch-based verifier, we notice that the increase in the latency is mainly because of the bottleneck at the sender. This is based on the observation that the packet-processing latency at the switch remains majorly unaffected though PUF response size increases (see Fig. 9(b)). On the other hand, for the host-based verifier, the packet-processing latency increased by 350 microseconds (see Fig. 9(c)). This indicates minimal performance implications with enhanced security for the switch-based verifier.

Furthermore, Fig. 10 shows the average latency gains of the switch-based verifier over the host-based verifier. We observe that as the size of response bits increases, the switch-based verifier gives gains from 100% to 472%. From the results, we

infer that the switch-based verifier significantly reduces the time taken to authenticate an IoT device.

C. Per-packet processing latency

We also study the impact of the increase in PUF response size on packet-processing latency at a hop level. That is, we measure the time the switch-based verifier or host-based verifier has taken to process two types of packets: authentication request and response. Note that a challenge is generated after processing the authentication request packet, and an acknowledgment is generated after validating the response packet. We send $10K$ authentication requests to both the switch-based verifier and the host-based verifier and measured the time to process these two types of packets.

To measure per-packet latency at the switch, we use *global_tstamp* intrinsic metadata available at both the ingress and egress pipeline. We compute the time difference between ingress and egress global timestamps and tag this information in the packet. For this, we add a custom header field named as *switchTime* (see Fig. 4). In the host-based verifier, we consider the time elapsed between receiving and sending socket calls at the host.

From Fig. 9(b) and Fig. 9(c), we observe that as the size of response bits increases, the per-packet latency of the switch is increased by only 1% whereas it is up to 150% for the host-based verifier. This is because as response size increases, the verifier has to perform more operations such as CRC32 hashing, XORing, ORing, and random number generation (see Fig. 2). For instance, CRC32 hashing on a host-based verifier for a 256-bit long value takes more CPU cycles than what is required for a 64-bit response. On the other hand, the programmable switch is designed to execute a small set of operations with minimal-to-no impact on line rates, thus the increase in response size has a minimal-to-zero effect on switch-based verifier performance – compared to 64-bit response size, we observe processing latency is increased by only 4 nanoseconds for 128-bit response size.

D. Throughput

We measure throughput or authentication rate in terms of the number of requests authenticated per second by a verifier. From Fig. 11, we observe there is 53-81% gain in authentication rate when the PUF verifier is offloaded to the switch. We notice that the gains go down as response size increases. We highlight that this is not because of more load on the switch, instead, it is due to the bottleneck at the sender as it is not able to generate more requests. More specifically, as the size of the response increases, it puts more load not only on the PUF verifier but also on the sender. That is, compared to the 64-bit response, for 256-bit, the sender has to perform xor and generate hash on more bits. Also, because of the same reason, in the host-based verifier, we did not observe much difference in throughput as the size of the response increases.

E. Scalability

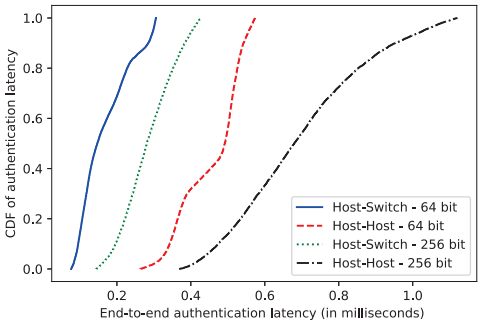
In this section, we evaluate the maximum number of CRPs that the switch can support given that the Wedge100BF-32x Tofino switch has a fixed-length pipeline with fixed hardware resources. We use MATs defined as static tables at compile time to store CRPs. Depending on the type of match, the compiler allocates either SRAM or TCAM blocks to these MAT units for the exact or ternary matches, respectively. In our implementation, we use MATs for the exact match and retrieve CRP. We evaluate our system scalability by finding the maximum number of CRPs the switch can support given its available SRAM resources. We did this by exhausting the SRAM available across multiple stages. For a 64-bit response size, the switch can support a total of 46 million CRPs with an average SRAM utilization of 82%. Average SRAM utilization is less than 100% because few of the pipeline stages are used for stateful registers and for other operations like hashing and concatenation. SRAM utilization is about 96.3% for the stages where MATs were placed. Assuming that we wish to support 100 CRPs per IoT device, our system can support 468K IoT devices concurrently. We can scale up or scale down the number of IoT devices supported by modifying the number of CRPs per IoT device. Fig. 12 shows the switch SRAM utilization and number of IoT devices supported (assuming 100 CRPs/IoT) for 64, 128 and 256-bit response.

Each transaction consumes one CRP, thus total memory consumed per transaction is 16B, 24B, and 40B for 64-bit, 128-bit, and 256-bit response sizes, respectively. And per-transaction stateful register memory required to store a challenge's hash is 4B.

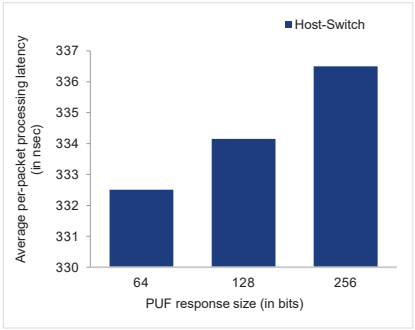
VI. Related Work

Crypto-based authentication. Most of the existing work uses crypto-based symmetric/asymmetric techniques [4] for authenticating IoT devices. [42], [43] propose a symmetric key-based authentication mechanism where they use AES to generate a secret key and the key is shared between the IoT device and the authentication server. [44] proposes asymmetric cryptography-based authentication where public key infrastructure (PKI) is used. Along with the PKI framework, Transport Layer Security [45] (TLS/SSL) and Datagram Transport Layer Security (DTLS) are extensively used for mutual authentication of IoT devices [46], [47]. In contrast, we focus on PUF-based authentication protocols.

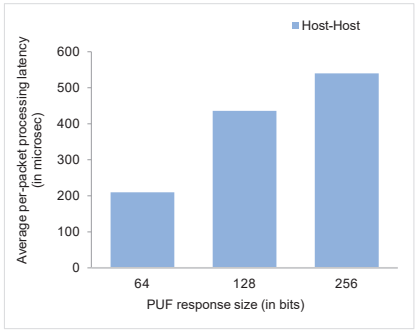
PUF-based authentication. Many works [5], [9], [9]–[11], [11]–[15], [48] advocate PUFs to be used for mutual authentication of both devices to device and device to a server. They use either strong PUFs, weak PUFs with crypto-techniques, or a combination of strong and weak PUFs. [49] proposes a lightweight authentication mechanism that uses 12% lesser memory compared to the traditional DTLS-based authentication. In [50], PUF-based authentication is extended to use a certificate-less, fast, TLS authentication. [48] proposes a combination of strong arbiter PUF and weak SRAM PUF for authentication. The existing PUF-based authentication approaches focus mainly on the security aspects of IoT. In contrast, we focus on the performance aspects of IoT-based latency-critical applications without compromising security.



(a) **CDF of end-to-end authentication latency**



(b) **Average per-packet authentication latency at the switch**



(c) **Average packet processing latency at the PUF verifier host**

Fig. 9. **End-to-end and per-packet latency**

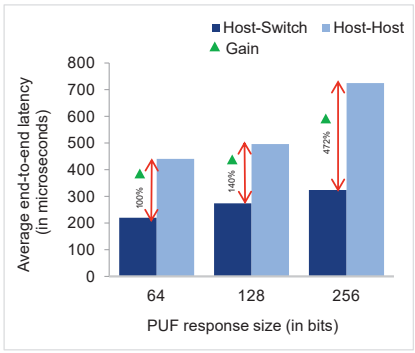


Fig. 10. **Average end-to-end authentication latency gain over H-H**

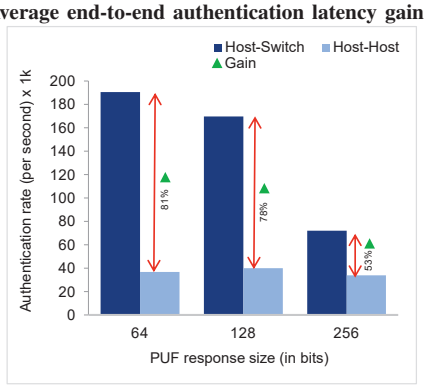


Fig. 11. **Authentication rate per second**

P4-based authentication. [51], [52] leverage programmable data planes to authenticate hosts in a network. These works are different from ours as we focus more on offloading PUF-based authentication protocol to a switch. Compared to our preliminary work [53], this work provides a more detailed design, implementation, and evaluation in terms of throughput and latency.

IoT and P4. Cloud-based solutions for industrial processes are simple to manage and flexible, yet they cannot satisfy URLLC requirements of real-time industrial use cases like robot arms, conveyor belts, etc. With the use of P4 switches, in-network computing has gained a lot of attention. [54] shows how cloud-robotics use cases can be accelerated if a whole

Size of Response (in bits)	TNA1		TNA2	
	SRAM (in %)	No. of IoT devices supported (per pipeline)	SRAM (in %)	No. of IoT devices supported (per pipeline)
64	82	468K	82	884K
128	75.3	360K	75.3	680K
256	66	180K	66	340K

Fig. 12. **Switch SRAM utilization and IoT devices supported per pipeline as response size varies.**

or some part of server-based computations are offloaded to p4 targets. Similarly, [55] proposes an in-network computing system where AI/ML models are offloaded to P4 devices. We foresee our work to be used in such IoT use cases, especially for URLLC applications.

VII. Conclusion and Future Work

In this paper, we motivate the need for acceleration of PUF-based authentication protocols and offload the PUF verifier of one of the protocols to a tofino-based programmable switch. Our evaluation on a real testbed demonstrates that the switch-based PUF verifier improves average authentication latency by 100-472%, authentication rate by 53-81%, and scales up to hundreds of thousands of IoT devices. In our future work, we plan to build a system on top of the offloaded protocol that handles: (1) proactive installation of CRPs based on current usage; (2) scale to more IoT devices by using an efficient eviction policy; and (3) secure communication using secure hash functions.

Acknowledgements

We thank the anonymous reviewers for their insightful feedback. We extend our gratitude to Harish S A for his valuable comments on the earlier drafts, Sarat Chandra Sai Danda, Yuvraj Makkena, and Dhiraj Saharia for helping in setting up the experimental testbed. Our work is supported by NM-ICPS TiHAN sanctioned by DST, India and a start-up grant by IIT-Hyderabad.

REFERENCES

[1] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, p. 100218, 2020.

[2] J.-Y. Lee, W.-C. Lin, and Y.-H. Huang, "A lightweight authentication protocol for internet of things," in *2014 International Symposium on Next-Generation Electronics (ISNE)*. IEEE, 2014, pp. 1–2.

[3] N. Chikouche, P.-L. Cayrel, E. H. M. Mboup, and B. O. Boidje, "A privacy-preserving code-based authentication protocol for internet of things," *The Journal of Supercomputing*, vol. 75, no. 12, pp. 8231–8261, 2019.

[4] A. Ghani, K. Mansoor, S. Mehmood, S. A. Chaudhry, A. U. Rahman, and M. Najmus Saqib, "Security and key management in iot-based wireless sensor networks: An authentication protocol using symmetric key," *International Journal of Communication Systems*, vol. 32, no. 16, p. e4139, 2019.

[5] A. Setyawan Sajim, "Open-source software-based sram-puf for secure data and key storage using off-the-shelf sram," 2018.

[6] M. N. I. Khan and S. Ghosh, "Information leakage attacks on emerging non-volatile memory and countermeasures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[7] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[8] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A puf taxonomy," *Applied Physics Reviews*, vol. 6, no. 1, p. 011303, 2019.

[9] T. Alladi, G. Bansal, V. Chamola, M. Guizani *et al.*, "Secauthuav: A novel authentication scheme for uav-ground station and uav-uav communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15 068–15 077, 2020.

[10] V. Pal, B. S. Acharya, S. Shrivastav, S. Saha, A. Joglekar, and B. Amrutur, "Puf based secure framework for hardware and software security of drones," in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2020, pp. 01–06.

[11] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, pp. 1–25, 2017.

[12] A. Braeken, "Puf based authentication protocol for iot. symmetry 10, 352 (2018)."

[13] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database," *IEEE transactions on dependable and secure computing*, vol. 16, no. 3, pp. 424–437, 2018.

[14] Y. Yilmaz, S. R. Gunn, and B. Halak, "Lightweight puf-based authentication protocol for iot devices," in *2018 IEEE 3rd international verification and security workshop (IVSW)*. IEEE, 2018, pp. 38–43.

[15] M. Barbareschi, A. De Benedictis, E. La Montagna, A. Mazzeo, and N. Mazzocca, "A puf-based mutual authentication scheme for cloud-edges iot systems," *Future Generation Computer Systems*, vol. 101, pp. 246–261, 2019.

[16] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. Alcaraz-Calero, "Xdp-based smartnic hardware performance acceleration for next-generation networks," *Journal of Network and Systems Management*, vol. 30, no. 4, pp. 1–26, 2022.

[17] Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction. Accessed: October 2021. [Online]. Available: https://icnp22.cs.ucr.edu/acceptedpapers.html

[18] M. A. Siddiqi, H. Yu, and J. Joung, "5g ultra-reliable low-latency communication implementation challenges and operational issues with iot devices," *Electronics*, vol. 8, no. 9, p. 981, 2019.

[19] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5g urllc: A case study on low-latency intrusion prevention," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 35–41, 2020.

[20] E. TR, "5g: Study on scenarios and requirements for next generation access technologies (3gpp tr 38.913 version 14.2. 0 release 14)," *ETSI TR 138 913*, 2017.

[21] J. Vestin, A. Kassler, and J. Åkerberg, "Fastreact: In-network control and caching for industrial control networks using programmable data planes," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 219–226.

[22] (2020) Intel Intelligent Fabric Processors. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[23] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling State-Intensive Network Functions on Programmable Switches," in *ACM SIGCOMM*, 2020.

[24] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, and H. Hu, "FastFE: Accelerating ML-based Traffic Analysis with Programmable Switches," in *ACM SIGCOMM SPIN workshop*, 2020.

[25] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, and S. Hirche, "Towards in-network industrial feedback control," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, 2018, pp. 14–19.

[26] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, and G. Pongrácz, "Towards low latency industrial robot control in programmable data planes," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 165–169.

[27] Maxim Intros MCU with PUF Technology. Accessed: October 2021. [Online]. Available: https://www.eetimes.com/maxim-intros-mcu-with-puf-technology

[28] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong pufs: models, constructions, and security proofs," in *Towards hardware-intrinsic security*. Springer, 2010, pp. 79–96.

[29] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

[30] P4_16 Intel Tofino Native Architecture - Public Version. Accessed: October 2021. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch-Document.pdf

[31] (2020) Intel FlexPipe - OZDAG, R. Intel R© Ethernet Switch FM6000 Series-Software Defined Networking. . [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf.

[32] G. G. K. H.-S. V. G. M. N. I. M. M. F. BOSSHART, P. and M. HOROWITZ, "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN." in *ACM SIGCOMM Conference (2013)*, 2013.

[33] (2020) Cavium XPliant switches. [Online]. Available: https://people.ucsc.edu/ warner/Bufs/Xpliant-cavium.pdf

[34] H. P. W. M. Stigge, Martin and J.-P. Redlich., "Reversing crc –theory and practice," 2006.

[35] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," ser. ACM SIGCOMM SPIN '21, 2021.

[36] (2022) Aps networks 2140d. Accessed: 2022-10-21. [Online]. Available: https://www.aps-networks.com/products/aps2140d/

[37] K. R. V.-S. A. R. V. I. . W. C. Katzenbeisser, S., "Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon." ser. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 283-301). Springer, Berlin, Heidelberg., 2012.

[38] P4_16 Language Specification. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html

[39] (2018) CRC Library. [Online]. Available: https://www.boost.org/

[40] pthread. [Online]. Available: https://man7.org/linux/man-pages/man7/pthreads.7.html/

[41] (2016) Netronome Agilio CX SmartNICs. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[42] A. Ghani, K. Mansoor, S. Mehmood, S. A. Chaudhry, A. U. Rahman, and M. Najmus Saqib, "Security and key management in iot-based wireless sensor networks: An authentication protocol using symmetric key," *International Journal of Communication Systems*, vol. 32, no. 16, p. e4139, 2019.

[43] A. Braeken, "Highly efficient symmetric key based authentication and key agreement protocol using keccak," *Sensors*, vol. 20, no. 8, p. 2160, 2020.

[44] M. M. Haque, A.-S. K. Pathan, C.-S. Hong, and E.-N. Huh, "An asymmetric key-based security architecture for wireless sensor networks," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 2, no. 5, pp. 265–277, 2008.

[45] TLS. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5246/

[46] B. Pismenny, H. Eran, A. Yehezkel, L. Liss, A. Morrison, and D. Tsafrir, "Autonomous nic offloads," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 18–35.

[47] P. M. Kumar and U. D. Gandhi, "Enhanced dtls with coap-based authentication scheme for the internet of things in healthcare application," *The Journal of Supercomputing*, vol. 76, no. 6, pp. 3963–3983, 2020.
[48] A. Mostafa, S. J. Lee, and Y. K. Peker, "Physical unclonable function and hashing are all you need to mutually authenticate iot devices," *Sensors*, vol. 20, no. 16, p. 4361, 2020.
[49] Y. Yilmaz, S. R. Gunn, and B. Halak, "Lightweight puf-based authentication protocol for iot devices," in *2018 IEEE 3rd international verification and security workshop (IVSW)*. IEEE, 2018, pp. 38–43.
[50] U. Chatterjee, R. Sadhukhan, V. Govindan, D. Mukhopadhyay, R. S. Chakraborty, S. Pati, D. Mahata, and M. M. Prabhu, "Pufssl: an openssl extension for puf based authentication," in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 2018, pp. 1–5.
[51] A. Bhattacharya, R. Rana, S. Datta, and U. Venkanna, "P4-sknock: A two level host authentication and access control mechanism in p4 based sdn," in *2022 27th Asia Pacific Conference on Communications (APCC)*. IEEE, 2022, pp. 278–283.
[52] E. O. Zaballa, D. Franco, Z. Zhou, and M. S. Berger, "P4knocking: Offloading host-based firewall functionalities to the network," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2020, pp. 7–12.
[53] K. Ranjitha, D. Pathak, P. Tammana, T. Alladi *et al.*, "Accelerating puf-based uav authentication protocols using programmable switch," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2022, pp. 309–313.
[54] S. Laki, C. Györgyi, J. Pető, P. Vörös, and G. Szabó, "In-network velocity control of industrial robot arms," 2022.
[55] G. C. Sankaran, K. M. Sivalingam, and H. Gondaliya, "P4 and netfpga based secure in-network computing architecture for ai-enabled industrial internet of things," *IEEE Internet of Things Journal*, pp. 1–1, 2021.