An Attack Resilient PUF-based Authentication Mechanism for Distributed Systems

Mohammad Ebrahimabadi, Mohamed Younis, Wassila Lalouani, and Naghmeh Karimi
CSEE Department, University of Maryland Baltimore County, Baltimore, MD 21250

***Abstract*— In most PUF-based authentication schemes, a central server is usually engaged to verify the response of the device's PUF to challenge bit-streams. However, the server availability may be intermittent in practice. To tackle such an issue, this paper proposes a new protocol for supporting distributed authentication while avoiding vulnerability to information leakage where CRPs could be retrieved from hacked devices and collectively used to model the PUF. The main idea is to provision for scrambling the challenge bit-stream in a way that is dependent on the verifier. The scrambling pattern varies per authentication round for each device and independently across devices. In essence, the scrambling function becomes node- and packet-specific and the response received by two verifiers of one device for the same challenge bit-stream could vary. Thus, neither the scrambling function can be reverted, nor the PUF can be modeled even by a collusive set of malicious nodes. The validation results using data of an FPGA-based implementation demonstrate the effectiveness of our approach in thwarting PUF modeling attacks by collusive actors. We also discuss the approach resiliency against impersonation, Sybil, and reverse engineering attacks.**

I. Introduction

The Internet of Things (IoT) refers to interconnecting miniaturized devices at a large scale, to serve many application domains such as smart transportation, home automation, power grid, and digital battlefield [1]. However, the scale, heterogeneity, ad-hoc topology formation, and dynamic interaction of the connected devices make IoT security a major challenge. If left unguarded, the network could be joined with malicious nodes that can apply a wide variety of attacks, such as leaking sensitive data, introducing black hole in the routing topology, and impersonating nodes [2]. Hence, device authentication is highly crucial. The Public Key Infrastructure (PKI) and Identity-Based Encryption (IBE) schemes [3] have traditionally been used for authentication purposes. However, their high overhead makes them unfit for IoT devices.

Lightweight hardware security primitives, and in particular Physically Unclonable Functions (PUFs) have received a lot of attention in recent years as an alternative to PKI and IBE schemes [3]. Thanks to the unintentional process variations accruing during the manufacturing of integrated circuits, a PUF generates a unique signature that corresponds to its input and output pairs, so-called Challenge-Response Pairs (CRPs). A PUF is embedded in each device during the fabrication, and a subset of its CRPs are registered after the device fabrication and during the enrollment phase. These CRPs are then used during operation to authenticate the device [4]. Most existing PUF-based authentication protocols, e.g., [5], rely on a central server, which is not always practical due to intermittent connectivity or occasional server unavailability.

To fill the technical gap, this paper promotes a novel Distributed Authentication Using PUFs (DAUP). In DAUP, a pair of IoT devices can mutually authenticate each other through direct exchange of CRPs. Each device stores a small subset of the CRPs of other nodes in the IoT framework. DAUP addresses two fundamental issues related to the use of CRPs in distributed systems: 1) How many CRPs a node N_i should share with any other IoT node? Indeed, there is a tradeoff between achieved security and storage overhead in each node; 2) How to counter the threat of CRPs sharing among compromised nodes. To elaborate, assume that a node stores M CRPs for each other node in the IoT framework. We refer to the set of CRPs of node N_i stored in N_j as $S_{i,j}$. A question would be how to select these sets, particularly, how different $S_{i,j}$ and $S_{i,k}$ should be, e.g., disjoint, intersecting, or completely similar. If $S_{i,j}$=$S_{i,k}$, an adversary who captures one of these nodes (say N_j) can impersonate N_i when interacting with N_k. On the other hand, storing completely dissimilar CRPs for N_i on other nodes, i.e., $S_{i,j} \cap S_{i,k} = \phi$, increases the vulnerability to the PUF modeling attack in case N_j and N_k collude and share the CRPs of N_i that they are aware of.

DAUP employs a novel challenge-bits scrambling scheme that is a function of the node identifier and also changes per packet. A node N_i determines the response for a challenge by factoring in the actual PUF output and the verifier identity , yet verifiers N_j and N_k would expect different responses for the same challenge given to N_i, and thus N_j and N_k fail in modeling the PUF of N_i even if they collude to do so. DAUP is applicable to the widely-used arbiter PUF and its derivatives and results in increased protection against modeling attacks even through collusion of multiple malicious nodes. In summary, the paper makes the following contributions:

- Devising an effective PUF-based distributed authentication protocol while safeguarding against PUF modeling attacks through collusive group of malicious nodes;
- Studying the impact of scrambling the challenge bit-stream on the success of the PUF modeling attacks launched via state-of-the-art ML techniques;
- Developing an approach for node- and packet-specific challenge scrambling to thwart collusion attacks that opt to model the embedded PUF;
- Analyzing the resiliency of DAUP against different attack scenarios, in particular cases where the adversary eavesdrops on communication links to intercept transmissions.
- Evaluating the proposed method using the data extracted from FPGA implementation of the target PUF.

Note that the novelty of our work is in the distributed aspect of the PUF-based authentication process while countering vulnerability to PUF modeling attacks even when multiple malicious nodes collude.

II. Related work

Although supposed to be unclonable, a PUF's behavior may be modeled using Machine Learning (ML) schemes [6]. To tackle modeling attacks hardware- and protocol-based methods have been proposed. The former mainly introduces additional

circuits, e.g., [7]. However, such a strategy imposes significant area, power, or traffic overhead (e.g. [8]), or the provisioned protection can be voided if one node is captured and the function is revealed since the same function is used for all nodes (e.g., [7]). Protocol-based methods can be classified into: (i) CRP obfuscation, (ii) controlled challenge bit-streams, (iii) noise injection. CRP obfuscation is applied either through encryption [9] or challenge bit-shuffling [10]. The latter can also be realized through challenge partitioning over multiple packets [11]. However, these methods are only applicable where a central controller is available.

The second category of the protocol-based schemes controls the used challenge bit-streams as a means to deprive an eavesdropper from collecting CRPs for modeling the PUF [12]. Yet, this category is only applicable when authentication is not conducted very often and their utility in IoT is questionable. Also, a central server needs to be engaged. Finally, adversarial ML has been used to introduce noisy data that degrades PUF modeling attempts [13] [14]. Wang et al. [13] proposed to poison the PUF's response based on the challenge bits. However, such a protection has been defeated by Ebrahimabadi et al. [14]. Although successful against modeling attacks, the approach of [14] imposes high computational overhead due to conducting ML-based modeling quite frequently.

Other PUF-based authentication schemes, e.g. [15], are vulnerable to security threats such as modeling, replay, and impersonation attacks. To alleviate such vulnerability, the use of a cryptosystem along with PUF has been pursued, where the PUF is leveraged to generate keys for securing the data exchange [3], [16]. Despite its effectiveness, such an approach imposes significant overhead on IoT devices. Overall, existing PUF-based authentication schemes rely on a central server. This paper enables distributed authentication while safeguarding against PUF modeling attacks by a collusive group of malicious nodes.

III. System Model and Preliminaries

A. Arbiter-PUF Architecture

The arbiter-PUF is characterized by its low implementation overhead and large CRP count; hence it is deemed effective for supporting device authentication. This PUF operates based on manufacturing process variations that result in a race between two signal paths (e.g., blue and green paths shown in Fig. 1), to generate a response bit for a challenge bit-stream [17]. The race corresponds to difference in propagation delays over these two paths, and affects the value latched by the arbiter circuit. Indeed, the arbiter output (PUF response) is affected only by the sign of the delay difference and not its amount. Although the delays vary based on the queried challenge, and the corresponding response is unpredictable, the interception of CRPs can make the arbiter-PUF (as well as its derivatives, e.g., XOR PUF) vulnerable to a modeling attack, where the adversary deploys ML to mimic the PUF behavior.

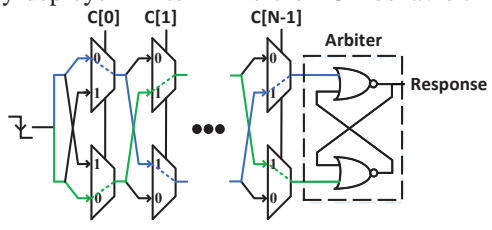

Figure 1: Illustrating the design of an arbiter-PUF.

B. System and Threat Models

We assume that an arbiter-PUF or one of its derivatives is embedded in each IoT device during fabrication, to be used in identifying the device. To authenticate a target node N_i (so called prover) by node N_j (so called verifier), the latter sends a challenge bit-stream to N_i. Then N_i applies the challenge to its PUF and sends the PUF response to N_j. Upon the receipt of the PUF response, N_j compares it with a pre-known value to confirm the identity of N_i. In DAUP, each node can play the role of a verifier and hence should have a subset of CRPs of all other nodes it interacts with. This is done during the enrollment phase, when a device joins the IoT framework.

In order to collect sufficient CRPs to model the target PUF and impersonate the corresponding device N_i, an adversary is assumed to either eavesdrop on the wireless links between N_i and other nodes (verifiers) in the system or hack some of these verifiers and read the stored CRPs of N_i in their memory. We categorize such a threat as collusion since it involves multiple nodes. Our proposed method thwarts such a collusive attack.

IV. DAUP Authentication Scheme

DAUP opts to enable PUF-based device authentication without engaging a server as an intermediary, and while countering the threat of PUF modeling by: (i) an eavesdropper on the communication links, and (ii) hacking (intruding) to one or multiple nodes. A server is involved only during the enrolment of a node N_i to securely provide some shared CRPs between N_i and other nodes. The shared CRPs could be updated, e.g., to prevent replay attack, when the server is reachable. This section describes DAUP in detail.

A. Detailed Design

Support for Distributed Authentication: Similar to other PUF-based authentication schemes, DAUP assumes that a PUF is embedded in each IoT device in the fabrication process. During the system initialization phase, when a device N_i is enrolled in an IoT framework, a set of challenge bit-streams, Γ^i, are generated and given to the device's embedded PUF, and their related responses are tabulated to be used for authenticating N_i after field deployment. We note that in DAUP the response of a challenge bit-stream depends on the verifier's ID where each challenge in Γ^i will be subject to scrambling before applying to the PUF of N_i, as we explain below; hence the server will store for N_i the sets $CRP_{i,j}$ corresponding to Γ^i for $\forall\ N_j$ in the system where $j \neq i$. *We note that the initialization phase would engage a server where the CRPs for all enrolled devices are saved. Such a server, which is assumed to be trusted, is not involved in the mutual authentication of IoT nodes*; this prevents the server from becoming a bottleneck, especially when the communication links among nodes are intermittent and frequent authentication is necessary. We also note that in DAUP the server will not store the actual response of PUF of N_i for Γ^i during the enrolment, instead, it only stores the response of a scrambled version of Γ^i, and hence the server does not constitute a modeling threat.

Each device N_j will receive from the server a subset $crp_{i,j}$ of $CRP_{i,j}\ \forall\ j \neq i$. The cardinality of $crp_{i,j}$ is usually subject to tradeoff. On the one hand, having many CRPs in $crp_{i,j}$ allows a device N_j to switch among multiple challenges over time and thus increases the robustness of the authentication

process. On the other hand, the aggregate memory size needed for N_j for storing $crp_{i,j}$ $\forall$ $j \neq i$ constitutes overhead and minimization of such overhead could be desirable to cope with resource constraints, especially for a large system with many nodes. We expect, nonetheless, that the lifetime of the system will play a dominant role since the required storage space is not much by today's standard, e.g., for a PUF with 64-bit challenge and 32-bit response, and a 100-node system, each node will need about 120K bytes for storing 100 CPRs for all other devices in the system. Note that to save power consumption, a multi-bit response is usually extracted by querying the PUF multiple times using the challenges generated internally based on the given challenge [10]. Indeed, the susceptibility to modeling attacks is not a factor in determining the size of $crp_{i,j}$ since challenge bit scrambling proves to safeguard the individual PUFs against an adversary that intercepts challenge response exchange among nodes or even hacks a verifier to read its memory, as we discuss below. Finally, we would like to stress that the initialization phase is a byproduct of the distributed operation of an IoT rather than something that DAUP dictates. Particularly, how to provide $crp_{i,j}$ to nodes when the system is set up and how to handle node addition and departure (e.g., when a node fails) is a general issue for distributed operation and not particularly imposed by how authentication is conducted.

Underlying PUF Modeling Countermeasure: PUFs, and in particular arbiter-PUFs, are vulnerable to ML-based modeling attacks, i.e., by accessing some CRPs an adversary can predict the PUF response of an unseen challenge bit-stream. Our protection strategy against such type of attacks is through scrambling the challenge bit-stream, C, in order to de-correlate C from the PUF response as seen by an eavesdropper. The basic idea is to reorder the challenge bits before applying it to the PUF [11]. For example, instead of applying the challenge bit-stream received from the verifier as $C[0], C[1], ..., C[N-1]$ in Fig. 1, a shuffled version like $C[15], C[N-4], ..., C[40]$ is given to the target PUF. Such reordering misleads the eavesdropper's ML model, as the intercepted response R would be for the Scrambled Challenge (SC) and not for C itself. Note that in DAUP the scrambling pattern is different per challenge and also per verifier node. As will be discussed in Section V, the adversary is unable to brute-force DAUP and uncover the unscrambled challenges.

Thwarting Collusive Attacks: By scrambling the challenge bit-stream, DAUP tackles modeling attacks launched by an adversary who captures CRPs of the targeted node, N_i, through eavesdropping on N_i communications or hacking and reading the memory of verifiers that deal with N_i. As noted, such mitigation is sufficient to protect an individual node as long as the adversary does not know how to unscramble the bit-steam, as otherwise by intercepting enough CRPs the adversary could model the PUF [11]. Meanwhile, if a similar scrambling pattern is used by multiple verifiers, hacking one verifier or intercepting the CRPs it uses for a node N_i, will allow the response of N_i to be known for certain challenges and does not prevent impersonation. Worse, if the scrambling function is fixed, hacking a node will make the entire network vulnerable.

To alleviate such vulnerability, DAUP considers a scrambling function that varies per verifier and per challenge. Basically, node N_i will scramble the challenge bit-stream C from verifier N_j based on the ID_j and the value of C itself. Hence, not only the response for the same challenge at different verifiers will be different, but also the scrambling pattern for challenges sent by the same verifier is different. This is a very powerful feature as conflicting data will be fed to the adversary's ML model which degrades its accuracy. To illustrate, let $R = F_i(C)$ be the PUF function for node N_i. According to DAUP, a verifier N_j will have $\hat{R} = F_i(\zeta_{i,j}(C))$, where $\zeta_{i,j}$ is the scrambling function that N_i applies for N_j, when N_j authenticates N_i with C. Generally $\hat{R} \neq R$; hence N_j tabulates $(C, \hat{R})$ to be used for authenticating N_i. DAUP benefits from the PUF not only in device authentication but also to devise the challenge scrambling pattern, as explained in the next subsection. Thus the scrambling pattern is both node-specific and challenge-specific, and differs from one node to another and one challenge to another. Indeed, this is one of the DAUP's main advantages.

B. DAUP Implementation

Fig. 2 shows a block diagram description of DAUP, where the steps are grouped into two sets: (1) those implemented in software and mainly reflect the communication interface, and (2) steps realized in hardware and correspond to the underpinning tamper-proof protection. The software part of DAUP deals with receiving requests from verifiers; a request from a verifier N_j will include its ID, which along with the queried challenge, C, are used to determine the scrambling function (pattern). Such a pattern is generated in hardware (bottom block), and is applied to C on the fly to generate the corresponding response that will be sent back to N_j. This response will then be checked against the value stored during the enrolment for such a challenge in N_j. DAUP operates in two phases:

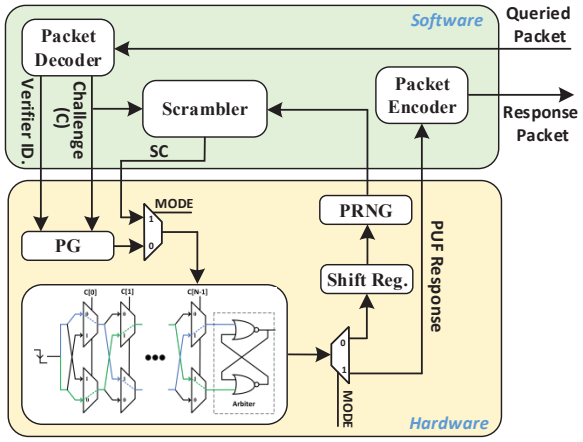

Figure 2: DAUP Block Diagram.

1) Determining a scrambling pattern: When node N_i is queried by N_j with a challenge bit-stream C, DAUP goes to phase 0 ($MODE = 0$) to define how C will be transformed. The mode signal controls the multiplexer (MUx), and de-multiplexer (de-Mux). The PUF is engaged in generating the scrambling pattern as we explain later;

2) Generating response for a verifier's request: In this phase, the picked scrambling pattern is used to reorder C and generate SC which in turn is fed to the PUF. The PUF response is then sent to N_j to be used for authenticating N_i. By setting ($MODE = 1$), the Mux. (and de-Mux.) will allow SC to be a PUF input and the corresponding response to be directed to the packet encoder within the software part of DAUP.

Alg. 1 summarizes the steps taken in DAUP when a verifier N_j authenticates N_i. After extracting the challenge (C) and the verifier ID, ID_j, from the request (line 1), the $MODE$ is set to "0" to determine the scrambling pattern being applied to C. An LFSR-based Pseudo Random Number Generator (PRNG) is employed to devise the scrambling pattern. To make DAUP more resilient against modeling attacks as well as brute forcing of the scrambling patterns, for each challenge, a different seed is used to initialize the embedded PRNG based on the verifier's ID as well as the value of the given challenge itself (lines 3-9).

Algorithm 1: DAUP authenticating node N_i by N_j

input : Queried packet received from the verifier node N_j
output: Response packet to be sent to the verifier N_j

1 Decode the queried packet to extract Challenge (C) and the verifier ID (ID_j)
2 $MODE \leftarrow 0$
3 $MC_1 \leftarrow C[0{:}N-S-1] \parallel ID_j$
  // S: bit-length of ID_j, N: PUF Challenge-length
4 $K \leftarrow log_2 N$
5 **for** $h \in \{1, \ldots, K\}$ **do**
6 $\quad R_h \leftarrow PUF(MC_h)$
7 $\quad Shift\ Reg.[K-1:0] \leftarrow Shift\ Reg.[K-2:0] \parallel R_h$
8 $\quad MC_h \leftarrow MC_h >> 1$
9 $PRNG(Init) \leftarrow Shift\ Reg.$
10 $SC[0] \leftarrow C[0]$
11 **for** $h \in \{1, \ldots, N-1\}$ **do**
12 $\quad H_h \leftarrow PRNG(h)$
13 $\quad SC[h] \leftarrow C[H_h]$
14 $MODE \leftarrow 1$
15 $R \leftarrow PUF(SC)$
16 **return** Response packet that includes R

Assume that the challenge length, N, is larger than ID_j bit-length (S), and the initial Mutated Challenge (MC_1) is built by concatenation (shown as ‖ in line 3) of ID_j and $N-S$ bits of the received challenge. We feed the PUF with MC_1, and push the related PUF response to a shift register that builds the PRNG seed bits gradually. Indeed as shown in lines 5-8, the Pattern Generator (shown as PG in Fig. 2) builds the bit values of the LFSR's seed, stored in the Shift Register, one by one by feeding the PUF with the last mutated challenge each time. To generate each mutated challenge (MC_h) DAUP circularly shifts the previous mutated challenge, (MC_{h-1}) one bit to the right. Note that any other function can be used to generate mutated challenges as well; due to its low implementation overhead, we have decided to use a circular shift function.

The PRNG is implemented as a $K = log_2 N$ bit LFSR with *primitive polynomial*, e.g., for a 64-bit challenge a 6-bit LFSR suffices. By definition, a primitive polynomial will cycle through all possible non-zero states [18]. After initializing the LFSR with the K bit PUF response stored in the shift register, we clock the LFSR $N-1$ times. This will result in $N-1$ distinct values, $H_1, H_2, ..., H_{N-1}$, for each of K bits, thanks to the primitive polynomial of LFSR. As shown in lines 10-13, concatenating the generated list with "0" will form the ordered list of $[0, H_1, H_2, ..., H_{N-1}]$ (line 12), which in turn is used to map the initial challenge bits, C, to its scrambled counterpart, SC. Upon applying SC to N_i's PUF in phase 1 ($MODE = 1$), the corresponding response is sent to the verifier N_j (lines 14-16), where it is matched with the expected (pre-tabulated) values. Even in the rare case of having a PUF response of '0' for all K queries in phase 0 (line 6), and thus initializing the LFSR with a seed equal to zero, DAUP still works. In this case, as the LFSR gets stuck in state zero (i.e., all of its bits are '0'), based on lines10-13, SC[h] would get C[0] for $h \in \{1, .., N-1\}$. In this case C is mutated, instead of scrambled, and the related SC can be used for authentication as the verifier has stored the response for such a mutated challenge.

Given that the LFSR generates $N-1$ (rather than N) unique values, DAUP does not change the location of $C[0]$ which is the furthest bit from the arbiter (see Fig. 1), while it maps $C[1], C[2], .., C[N-1]$ to $C[H_1], C[H_2], .., C[H_{N-1}]$, denoted as $SC[1]$ to $SC[N-1]$, respectively. We argue that keeping the value of C[0] intact before and after scrambling does not have a considerable effect, as the further the challenge bit is from the arbiter, the lower the impact it has on the delay chains [6]. Note that in case $S > N$, the initial mutated challenge MC_1 is formed of the $F < N$ (e.g., N/2) least-significant bits of ID_j instead of the whole bit-stream, concatenates with $N-F$ bits of C. This constitutes a special case for Line 3 and is not shown in the algorithm for the sake of simplicity.

V. Experimental Results and Discussions

We validated DAUP using 6 Xilinx ARTIX7 FPGA boards, each representing an IoT device and assigned a 32 bit unique ID. The nodes are connected using Zigbee transceivers. A 64-bit arbiter-PUF and a 6-bit LFSR with the $X^6 + X^5 + 1$ primitive polynomial have been implemented on each device. The latter is to generate the scrambling patterns. The adversary is assumed to intercept the exchanged CRPs between the targeted node N_t (i.e., prover) and the verifier nodes N_v where $1 \leq v \leq 5$. We extracted the response for a set of 22,000 randomly chosen challenges applied to the PUF of N_t.

To show the resiliency of DAUP against modeling attacks that use state-of-the-art ML schemes, we consider the cases where the adversary uses Neural Network (NN), Support Vector Machine (SVM) or Logistic Regression (LR) as the representatives of ML techniques. We used a 5-layer fully connected NN with one input layer (with 64 neurons reflecting the PUF size), three nonlinear hidden layers (with 5, 10 and 15 neurons) and one output neuron with a sigmoid function. A rectified linear unit (ReLU) is used as an activation function in every layer. The learning rate, momentum, and # of epochs are 0.01, 0.99, and 2000 respectively.

A. Experimental Results

Challenge Scrambling: The first set of results, shown in Fig. 3, demonstrates the efficiency of DAUP in diminishing the accuracy of the ML-based model that an adversary builds based on the intercepted CRPs. Here, we assume that the adversary eavesdrops on the communication line between N_v (v=1,2,...,5) and the target node (i.e., prover) N_t, and trains a NN model to predict the responses of the unseen challenges

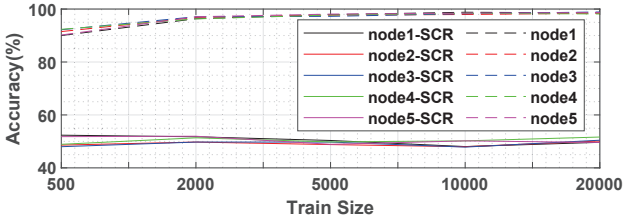

Figure 3: Accuracy of the adversary's ML model with and without using DAUP. The PUF model is built using NN based on the CRPs exchanged between the verifier ($N_v, v = 1, 2, ..., 5$) and prover(N_t) nodes while the training size is increased to 20K.

fed to N_t's embedded PUF. In Fig. 3, the solid and dotted curves relate to the cases where challenge-scrambling is and is not performed, respectively. As shown, when there is no protection, with as low as 500 CRPs, the adversary can train a model successfully and achieve an accuracy of $\approx 90\%$; such accuracy grows to 99% with the training size of 3,000 CRPs. However, when DAUP is applied (solid lines), the accuracy stays close to 50% even with a training size of 20,000 CRPs. It is important to note that, because of the binary nature of a PUF response, a 50% accuracy reflects a random guess of the PUF response. Thus, by benefiting from challenge scrambling, DAUP can successfully prevent PUF modeling attacks.

To demonstrate the success of DAUP in thwarting collusive PUF modeling attacks, we consider the three following scenarios where the adversary uses NN models:

Scenario I: In this case, the adversary eavesdrops on communications between the target device (N_t) and verifier N_i. The adversary opts to build a model of N_t's embedded PUF based on the intercepted CRPs in order to predict the responses of N_t to another verifier, say N_j, where $j \neq i$. The accuracy of such prediction is listed in Table I when 100 (shown preceding the parentheses) and 1000 (shown inside parenthesis) CRPs of N_t are intercepted. Here, we assume that a total of 500 (5000 in the second case) randomly selected CRPs for N_t are shared with all five versifiers to authenticate N_t. Note that the verifier nodes may or may not have tabulated similar challenges of N_t, yet the stored challenge set can have some overlap. The N_t's responses held by each verifier for the same challenge may be different as the verifier's ID affects the scrambling pattern; recall that the verifier stores the response of the scrambled rather than the original version of the challenge.

The results in Table I confirm the effectiveness of DAUP. Since DAUP uses a different scrambling pattern based on the verifier's ID and challenge bit-stream, the adversary is unsuccessful in predicting the target PUF's response in all cases. Specifically, if the model is built based on the CRPs exchange between N_t and N_1 (i.e., i=1), a verifier N_2 (i.e., j=2) is unable to predict the responses of the other challenges exchanged between N_t and N_1; even with intercepting 1000 CRPs the success rate is 51%. Recall that the results for the cases where $i = j$ are shown in Fig. 3.

Table I: Adversary's gained accuracy in predicting the response of a prover N_t to a verifier N_j when the adversary builds the N_t model based on the CRPs exchanged between N_t and a verifier N_i, $1 \leq i, j \leq 5, i \neq j$ using NN. The numbers preceding and inside parenthesis reflect the modeling accuracy when 100 and 1000 CRPs used for building the model, respectively.

	j				
i	1	2	3	4	5
1	———	51(51)	42(47)	49(48)	52(50)
2	55(47)	———	52(50)	45(49)	54(48)
3	43(49)	53(49)	———	51(50)	41(49)
4	34(48)	44(50)	48(51)	———	48(51)
5	52(49)	52(54)	47(52)	51(50)	———

Scenario II: In this case, the adversary is able to eavesdrop on multiple communication links, specifically, between verifiers N_i and N_k, and the prover N_t. The adversary used the intercepted CRP between N_t and those two verifiers to model the PUF of N_t and predict the responses of N_t to another verifier, say N_j where $j \neq i, j \neq k$. The prediction accuracy is reported in Table II the adversary trains the PUF model using 100 and 1000 CRPs per verifier, i.e., a total of 200 and 2000 CRPS. The results in Table II affirm DAUP's ability to counter modeling attacks even if multiple nodes collude or if the CRPs exchanged between them and the prover are intercepted. The modeling accuracy for more than 2 colluding nodes follows a similar trend, but is not shown due to space constraints.

Table II: Adversary's accuracy in predicting the response of a prover N_t to a verifier N_j when using NN to build a model of N_t's PUF based on CRPs exchanged between N_t and both N_i and N_k verifies. The numbers preceding (inside) parentheses depict the modeling accuracy when 100 (1000) CRPs are exchanged between each verifier and N_t used to train the model.

	j				
i,k	1	2	3	4	5
1,2	———	———	48(47)	50(49)	50(48)
1,3	———	65(52)	———	48(50)	58(51)
1,4	———	51(50)	46(51)	———	56(49)
1,5	———	56(50)	49(50)	48(50)	———
2,3	50(49)	———	———	47(51)	48(48)
2,4	48(49)	———	65(49)	———	46(49)
2,5	52(47)	———	54(51)	51(50)	———
3,4	47(49)	55(51)	———	———	47(50)
3,5	50(48)	56(51)	———	42(50)	———
4,5	57(48)	50(52)	41(51)	———	———

Scenario III: This scenario assumes that the adversary can intercept a portion (L%) of all communications between IoT nodes and N_t, and builds the model based on the captured CRPs to predict the response of unseen challenges sent to N_t from verifier N_j. The modeling accuracy is provided in Table III, when L=10%, 20%, .., and 50% of all exchanged CRPs between each node and N_t are intercepted and used to train the NN model and in turn to predict the responses for unseen challenges. The obtained results indicate that increasing the intercepted percentage of exchanged CRPs does not have any significant impact in increasing the accuracy of modeling N_t, i.e., in all cases the accuracy is$\approx 50\%$. This is a clear testimony for the effectiveness of DAUP.

Table III: The success rate for predicting the response of prover N_t to a verifier N_j when the adversary builds a NN model of N_t based on L% of the CRPs exchanged between N_t and all other verifiers. The numbers preceding (inside) parenthesis depict the modeling accuracy when 100 (1000) CRPs are transferred between each verifier and N_t.

	j				
L%	1	2	3	4	5
10	52(48)	45(53)	45(49)	47(54)	47(47)
20	42(49)	50(49)	52(52)	42(51)	42(49)
30	44(46)	52(51)	50(48)	50(48)	50(47)
40	50(47)	40(49)	58(50)	58(51)	58(49)
50	58(46)	52(52)	48(46)	48(48)	48(51)

Resiliency Against Various ML Modeling Techniques: This set of results gauge the resiliency of DAUP against modeling attacks that utilize SVM or LR. Due to space limitation, we only consider training and test set sizes of 20K and 2K, respectively. Fig. 4 shows the results. Test 1 refers to Scenario I where the adversary eavesdrops on the communication link between N_1 and N_t and tries to predict N_t's response when queried by N_2. Test 2 realizes the case where the adversary intercepts the exchanged CRPs between N_t and N_1 as well as the ones transferred between N_t and N_2 (Scenario II). The same results can be obtained if N_1 and N_2 collude and share the N_t CRPs they have stored. The trained model is then used to predict the response of N_t to the challenges submitted by N_3. Finally in Test 3, we selected the last item of scenario III, i.e., the best case scenario for the adversary among those cases where he intercepts 50% of all communication traffic. The trained model is then used to infer the rest (other 50%) unseen responses. As Fig. 4 shows the modeling accuracy does not exceed 55% in all these tests when NN, SVM, or LR is used. The CMA-ES based attack [17] is not applicable in our case as the response to the same query changes per verifier.

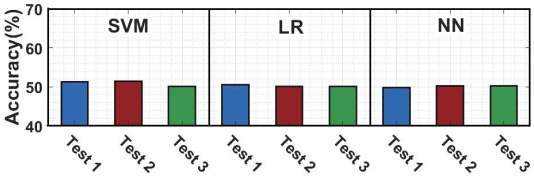

Figure 4: Modeling accuracy in each of the considered test cases when different ML schemes are used to model the target PUF in presence of DAUP.

VI. Overhead and Security Analysis

Protection against Impersonation and Sybil attacks: Impersonation refers to the scenario when a malicious node tries to identify itself as a legitimate node. In DAUP, since the challenge bit-streams are scrambled, the attacker is unable to build the prover's PUF model to impersonate it. Also the scrambling pattern changes per verifier and per packet, as well as per prover (due to the use of PUF in the process of generating scrambling patterns). Thus, an adversary cannot infer the scrambling algorithm even by colluding with other malicious nodes. This makes impersonation impossible. Similarly, a Sybil attack where an adversary claims multiple valid identities, is not feasible as an adversary cannot even impersonate a single IoT node.

Protection against Reverse Engineering attack: DAUP makes the chip resilient against reverse engineering even if the adversary has access to the IoT device itself and opts to uncover the chip using imaging, delaying, etc. This is because DAUP generates the scrambling pattern for each chip by using the chip's embedded PUF, thus a unique and unpredictable signature, thanks to the process variations. Thereby, even if the algorithm is known, it cannot be compromised.

Protection against Brute-Forcing the scrambling pattern: Even if an adversary knows that scrambling is used, he needs to guess the seed value generated based on PUF responses to find the scrambling pattern. Thus for an N bit challenge, the seed value ($= log_2 N$ bit) should be guessed. As the seed changes per challenge, for modeling the PUF using M challenges, all possible N^M cases should be checked in order to find the unscrambled pattern and model the PUF.

Required storage per node: The size of tabulated CRPs in each device (i.e., verifier) depends on the number of devices in the IoT framework (ND), authentication rate (AR), and time interval between enrollments (TI). The fewer ND is, the more CRPs per node can be stored. On the other hand, in case of higher AR and TI rates, more CRPs are tabulated to prevent replay and impersonation attacks. Eq. (1) shows the memory size (in bit) required by DAUP for each IoT device. Here, each CRP includes an N-bit challenge and R-bit response.

$$Memory\ Size = TI \times AR \times (ND - 1) \times (N + R) \quad (1)$$

DAUP Overhead and noise impact: DAUP imposes a negligible hardware overhead. The overhead for an IoT device with a 64-bit arbiter-PUF includes: (i) the PG block which performs concatenation and circular shifting operations, being implemented via a 64-bit register and rewiring, (ii) a 6-bit Shift register to store the $PRNG$ seed, (iii) a 6-bit primitive LFSR built of 6 flip-flops and 1 XOR gate, and (iv) one 64-bit MUX and a 1-bit De-MUX. The $Scrambler$ is implemented in software as a simple algorithm that repositions the challenge bits based on the LFSR's outputs in consecutive clock cycles. Finally, a small controller, implemented as a 6-state finite state machine, manages DAUP. Moreover, for this 64-bit PUF, 6 clock cycles are needed to initialize the PRNG and 63 clock cycles to generate the scrambling patterns. Such latency for an IoT device operating at 1GHz is in the order of 100ns, which is quite insignificant compared to the time needed to transfer/encode/decode packets (order of ms). Similarly, the added hardware will draw negligible power. Finally, we evaluated the impact of measurement noise in our setup by repeating each experiment. In $\approx$1% of the cases we observed noisy responses when DAUP is used. Error correction codes (ECC) and majority voting over multiple measurements can be used to mitigate this negligible noise [3].

VII. Conclusion

IoT is characterized by the large-scale involvement of nodes and spatial coverage. Therefore accessibility and availability of a server for all IoT devices is not always guaranteed. This makes distributed authentication plausible. In this paper, we have proposed DAUP, a low overhead and highly secure distributed PUF-based authentication scheme that fits the resource-constrained IoT devices. DAUP leverages PUFs as device fingerprints and employs challenge scrambling to safeguard them against modeling attacks. The scrambling pattern depends on the queried challenge, versifier's ID, and prover's unclonable PUF signature. The validation results confirm the efficacy of the proposed method against contemporary security attacks, both PUF modeling and conventional attacks launched in IoT frameworks such as impersonation and Sybil attacks.

Acknowledgement

This work has been supported by the National Science Foundation MRI Award (1920079).

References

[1] M. Younis, "Internet of everything & everybody: Architecture & service virtualization," *Comp. Communications*, vol. 131, pp. 66–72, 2018.
[2] Z. Wan *et al.*, "An internet of things roaming authentication protocol based on heterogeneous fusion mechanism," *IEEE Access*, vol. 8, 2020.
[3] U. Chatterjee et al., "Building PUF Based Authentication and Key Exchange Protocol for IoT Without Explicit CRPs in Verifier Database," *IEEE TDSC*, vol. 16, no. 3, pp. 424–437, 2019.
[4] A. Aysu *et al.*, "End-to-end design of a PUF-based privacy preserving authentication protocol," in *CHES*, 2015, pp. 556–576.
[5] A. Shamsoshoara *et al.*, "A survey on Physical Unclonable Function (PUF)-based security solutions for internet of things," *Computer Networks*, vol. 183, p. 107593, 2020.
[6] M. Ebrahimabadi *et al.*, "A novel modeling-attack resilient arbiter-PUF design," in *VLSI Design*, 2021, pp. 123–128.
[7] S. S. Zalivaka *et al.*, "Reliable and modeling attack resistant authentication of arbiter PUF in fpga implementation with trinary quadruple response," *IEEE TIFS*, vol. 14, no. 4, pp. 1109–1123, 2019.
[8] C. Gu *et al.*, "A modeling attack resistant deception technique for securing PUF based authentication," in *AsianHOST*, 2019, pp. 1–6.
[9] P. Gope et al., "Lightweight and privacy-preserving two-factor authentication scheme for iot devices," *IoT J.*, vol. 6, no. 1, pp. 580–589, 2019.
[10] M. A. Qureshi et al., "PUF-RAKE: A PUF-based robust & lightweight authentication and key establishment protocol," *IEEE TDSC*, 2021.
[11] M. Ebrahimabadi *et al.*, "A PUF-based modeling-attack resilient authentication protocol for iot devices," *IoT J.*, pp. 1–1, 2021.
[12] M. Barbareschi *et al.*, "A PUF-based mutual authentication scheme for cloud-edges iot systems," *FGCS*, vol. 101, pp. 246–261, 2019.
[13] S.-J. Wang *et al.*, "Adversarial attack against modeling attack on PUF," in *DAC*, 2019, pp. 1–6.
[14] M. Ebrahimabadi *et al.*, "Countering PUF modeling attacks through adversarial machine learning," in *IEEE ISVLSI*, 2021.
[15] Y. Lao et al., "Reliable PUF-Based Local Authentication with Self-Correction," *TCAD*, vol. 36, no. 2, pp. 201–213, 2016.
[16] F. Farha *et al.*, "SRAM-PUF based entities authentication scheme for resource-constrained iot devices," *IoT J.*, pp. 1–1, 2020.
[17] G. T. Becker, "The gap between promise and reality: On the insecurity of xor arbiter PUFs," in *CHES*, 2015, pp. 535–555.
[18] S. R. Ghorpade et al., "Primitive polynomials, singer cycles and word-oriented linear feedback shift registers," *Designs, Codes and Cryptography*, vol. 58, no. 2, pp. 123–134, 2011.