A Proof of Concept SRAM-based Physically Unclonable Function (PUF) Key Generation Mechanism for IoT Devices

Ashwija Reddy Korenda, Fatemeh Afghah, Bertrand Cambou, Christopher Philabaum
School of Informatics, Computing and Cyber Systems, Northern Arizona University, Flagstaff, AZ, USA
{ashwijakorenda, fatemeh.afghah, bertrand.cambou, cp723}@nau.edu

***Abstract*—This paper provides a proof of concept for using SRAM based Physically Unclonable Functions (PUFs) to generate private keys for IoT devices. PUFs are utilized, as there is inadequate protection for secret keys stored in the memory of the IoT devices. We utilize a custom-made Arduino mega shield to extract the fingerprint from SRAM chip on demand. We utilize the concepts of ternary states to exclude the cells which are easily prone to flip, allowing us to extract stable bits from the fingerprint of the SRAM. Using the custom-made software for our SRAM device, we can control the error rate of the PUF to achieve an adjustable memory-based PUF for key generation. We utilize several fuzzy extractor techniques based on using different error correction coding methods to generate secret keys from the SRAM PUF, and study the trade-off between the false authentication rate and false rejection rate of the PUF.**

***Index Terms*—PUF, key generation, IoT, SRAM, Fuzzy Extractors**

I. Introduction

Internet of things is one of the booming technologies in the current era. It allows the connection of various sensors, devices and everyday objects to interact with each other, transfer and retrieve data, intelligently respond and trigger actions accordingly. Today, many devices which paved the way to the advancement of smart homes to smart cars are connected through IoT networks. Medical devices like pacemakers and neurosimulators which allow monitoring of patients from afar, are connected through IoT [1].

Though IoT allows massive advancements in various fields, it comes with unique challenges considering its heterogeneous nature and the large number of devices. By the year 2020, it is expected that about 20 billion devices will be connected to IoT network. Sensitive information is exchanged between different IoT "things", which is responsible for the proper functioning of smart car, pacemakers, neurosimulators or smart home. Bio-medical devices such as pacemakers and neurosimulators usually send small electric pulses to the heart and brain, respectively. If adversaries manage to send unwanted electric pulses to the brain or heart, it may lead to serious threats to patient safety. For instance, if such sensitive information is hacked or tampered, it may cause serious injuries or death of a person [2]. An article published by CNBC in 2016 discusses the possibility of cyber-criminals targeting medical devices [3]. Therefore, it is very important to ensure safe and secure encryption of messages. Success of IoT depends on the security it can provide over innovation.

This material is based upon the work supported by the National Science Foundation under Grant No. 1827753.

Security challenges in IoT encompass different aspects of identification, authentication, encryption, confidentiality, jamming, cloning, hijacking and privacy as explained in Table I. IoT networks are prone to various types of attacks such as spoofing, data manipulation, replay routing, Denial of Service (DoS), node capture and Sybil attack which are briefly explained in table II

Several characteristics of IoT networks including the low power of IoT nodes, the low computational capability combined with the heterogeneity and large scale nature of IoT networks limit the application of standard security mechanisms to the IoT networks. In this paper, we focus on the problem of identification, authentication and encryption of the messages sent between two IoT devices to protect the confidentiality of the sensitive information transmitted.

Encryption has been widely used by several mechanisms in order to send the messages without the risk of being understood by the hackers and to eliminate the risk of data manipulation. Hence, cryptographic methods play a crucial role in the security of IoT systems. The majority of common cryptographic mechanisms such as public key infrastructure (PKI), advanced encryption standard (AES), and elliptic curve cryptography (ECC) rely on private cryptographic keys [4]. These secret cryptographic keys are expected to be robust, reliable and reproducible and there are usually stored in the non-volatile memory (NVM) of the devices. However, the NVMs are highly susceptible to physical attacks due to their robust electrical nature. Several hardware-based security solutions have been developed to enhance the security of private-key based cryptographic methods [5], [6].

Physically unclonable functions (PUFs) are a hardware-based security primitive introduced in 2002 [7]. The PUF utilizes the intrinsic manufacturing variations in a device to generate a fingerprint of the hardware that offers the valuable advantage of unclonability. This means that the device cannot be cloned even when a hacker has physical access to the device. Therefore, the PUFs are unique to their device and can be used as a security primitive to enable device-based

TABLE I
DEFINITIONS OF DIFFERENT SECURITY THREATS IN IOT NETWORKS

Name	Definition
Identification	Unique information describing the subject, which can be utilized to identify it
Authentication	Secret information confirming the identity of the subject
Encryption	Conversion of plain text to protected message, called cipher
Confidentiality	State of keeping a secret
Jamming	Denial of Service attack in a wireless medium
Cloning	Replicating the process
Hijacking	Attacker gaining control of the system
Privacy	Ability to protect sensitive information

TABLE II
ATTACKS IN IOT NETWORKS

Attack	Impact
Spoofing	Malicious user impersonating another device to launch attacks on network hosts
Replay routing information	Fraudulent repeating or delaying of valid data transmission
Data manipulation	Altering sensitive information
Denial of Service (DoS)	Reduction in networks' capacity, disable the network
Node capture attack	Stealing sensitive information from a captured node to compromise the entire network
Sybil attack	Node claiming multiple identities

identification, and authentication. More importantly, PUFs can provide a low cost alternative solution for on-demand generation of cryptographic keys from the device rather than the conventional methods, where the secret keys are produced and distributed by the server and stored in the IoT device memories [8].

In this paper, we developed a proof-of-concept key generation mechanism using static random-access memory (SRAM) PUFs, which does not involve the key storage in the devices' memory. This developed technology utilizes the idea of ternary PUFs [9] that has proven to create stronger PUFs compared to common binary PUFs. The developed key generation mechanism has been tested using different SRAM devices. We also evaluated the ability of various fuzzy extractors proposed in the literature to extract the key of the noisy PUF input.

The rest of this paper is organized as follows: In section II, we introduce the concepts of PUFs and focus on SRAM based PUFs and its characterization. In section III, we explain, how Fuzzy extractors can be used to correct the noisy input PUF response. In section IV, we describe the key generation and regeneration process using fuzzy extractors proposed in the literature. In section V, we discuss our experimental setup, the custom-made software and Arduino shield which allows us to extract SRAM responses on demand. We then use these responses to generate secret keys using Fuzzy extractors.

II. PHYSICALLY UNCLONABLE FUNCTIONS

A. PUF-based technology

Physical unclonable functions (PUF) are the equivalent of human fingerprints; the variations created during fabrication makes each PUF authenticable from each other. Authentication protocols based on PUFs [7], [10]–[17], can be effective when these PUFs show intra-PUF stability, and offer inter-PUF randomness. Memory structures [18]–[25], SRAM [18], [19], DRAM [20], Flash [21], ReRAM [22], [23], and MRAM [24], [25], are all suitable to generate strong PUFs.

One method to generate PUFs from memory devices is to characterize a particular parameter $\mathcal{P}$ of the cells of its array. The values of parameter $\mathcal{P}$ vary cell to cell, and follow a distribution with a median value T. The cells with $\mathcal{P} < T$ generate "0" states, and others generate "1" states. The PUF "Challenges", also called "initial" responses is the data stream generated during "enrollment" of the clients devices by the server. In some protocols, the Challenges can be generated by processing, or averaging, multiple queries and measurements of the PUFs. These Challenges can be represented by binary, ternary, or any other radix.

During enrollment, the "Challenges" of the PUFs located in each client device are securely downloaded in a data base of the server. The PUF responses is the data streams generated by the PUF during the life of the client devices. These PUFs are physical elements that can age, and be subject to temperature changes, electro-magnetic interferences, and other environmental effects. Challenge-response-pairs (CRP) are generated on demand by the server of the PUFs of client devices. When the PUF is a strong PUF, the server needs to send instructions to the client devices to find the particular address in the PUF, to generate challenge-response-pairs. The Helper is the data stream generated by the server from these challenges which is transmitted to the client devices to correct the responses. Some authors call the instructions the challenges, the CRPs then become the pairing of the initial responses to the freshly generated the responses.

When the CRP error rates are below 10%, the responses can be used as part of the authentication protocol, which has significant commercial value to protect cyber physical systems. Satisfactory authentications have low false rejection rates (FRR), and low false acceptance rates (FAR). The use of PUFs to generate cryptographic keys from the responses is

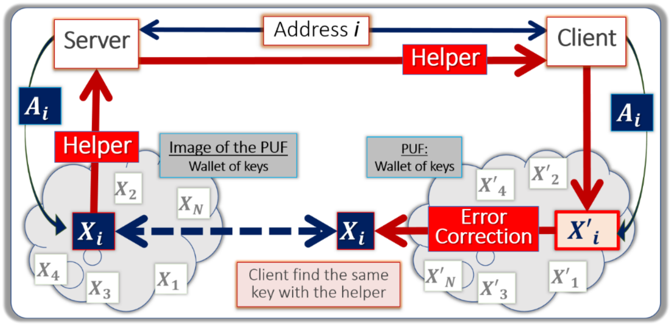

Fig. 1. PUF-based cryptography with an error correcting scheme.

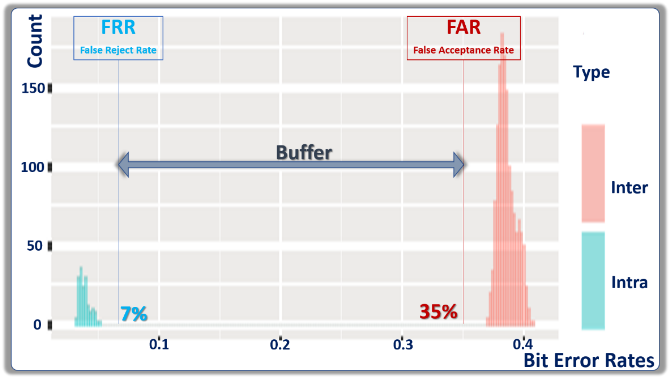

Fig. 2. Characterization of the intra-PUF, and inter-PUF error rates.

more challenging than generating responses for authentication; a single-bit mismatch in a cryptographic key is not acceptable for most encryption protocols. This requires schemes that fully correct the responses of the PUF.

A typical architecture to drive a network of client devices with PUFs is shown in Fig.1. The server initiates the process by sending instructions to the client device to find the addresses $\boldsymbol{i}$ within the PUFs which act as wallet of keys, and to extract the Responses X_i. The server independently analyses the challenges X_i stored in the table at address $\boldsymbol{i}$, and generates the Helper with error correcting methods. The Helpers are transmitted to the client devices as part of the protocol. The client device corrects the response X_i with the Helper and correcting methods such as a fuzzy extractors. To be acceptable for encryption, the corrected responses, and the challenges of the server should be perfectly identical. Thereby, the same key X_i is independently generated for encryption schemes.

Such PUF-based protocols have several weaknesses when utilized in IoT networks. The client devices are burdened, and need to consume additional computing power to run the error correcting codes; and such protocols increase the vulnerability to side channel attacks, differential power analysis, and the potential exposure of the Helpers. In this paper, we develop a PUF-based key generation scheme based on using a ternary state key generation method to exclude fuzzy cells in the PUF. We also utilize low power correction schemes to minimize the length of the Helper, and computing power needed to deliver error free keys to address the needs of a key generation mechanism in IoT networks.

B. SRAM-based PUFs

SRAM-based PUFs exploit the lack of perfect symmetry of their cells, which are designed with flip-flop logic. The SRAM cells tested in this work have six transistors per cell, four transistors for the flip-flop, one transistor to control the programming of the cell, one transistor to control the read. During the programming cycle, the flip-flop is set at “0”, or “1”; during read cycle, the state of each flip-flop is detected. Power-off/power-on cycles are used to get PUF functionality from SRAM devices. During power-off/power-on cycles, the cells have an equal opportunity to recover a 0 or 1 state; however, the majority of the cells of the SRAM arrays have predictable behavior. In most cases, they have a preferred state either a 0 or a 1 due to natural variations, introduced during the manufacturing of the devices; the flip-flop cells are never perfectly symmetrical. Examples of variations include slight differences in the size of the transistors due to non-uniformities of the photolithography of the process, micro-defects, natural variations of the chemical composition of the metallic lines, and many other small fabrication parameters.

Each SRAM cell is different from other cells; each SRAM device is in general, different from the others. When 256 cells are randomly selected, and subjected to power-off/power-on cycles, the vast majority of the data stream stored in these 256 cells is predictable, and can be used as a PUF challenge-response-pair. While the asymmetry is strong enough for most of the cells, the CRP error rates of SRAM PUFs can be prohibitive because some cells are too close to a symmetrical configuration, and they can flip randomly on both sides.

C. Characterization of SRAM PUFs

The experimental work to analyze the CRP error rates is based on 45 different 32kByte commercial SRAMs, produced by Cypress Semiconductor. The SRAMs with the circuitry needed for power-off/power-on cycles are assembled in custom PCB boards, driven by Arduino boards. A complete power-off of the SRAMs is needed in spite of capacitive and inductive effects, which can prevent the power from switching off. To mitigate the problem, all SRAM I/Os are shorted to the ground during these cycles to drain the charges, and during periods of time that are long enough, one second in this case. The intra-PUF and inter-PUF error rated of these SRAM devices is summarized in Fig.2. The intra-PUF error rate is defined as the CRP error rates of each cell compared to itself. The inter-PUF error rate is defined as the CRP error rates of each cell compared to the cells located at the same address on a different SRAM. For this analysis, 10 devices were characterized. When used for authentication, this allows the modelling of FRRs, and FARs

As shown in Fig.3, the error rates of the SRAM PUFs are reduced with an enrollment that incorporates successive

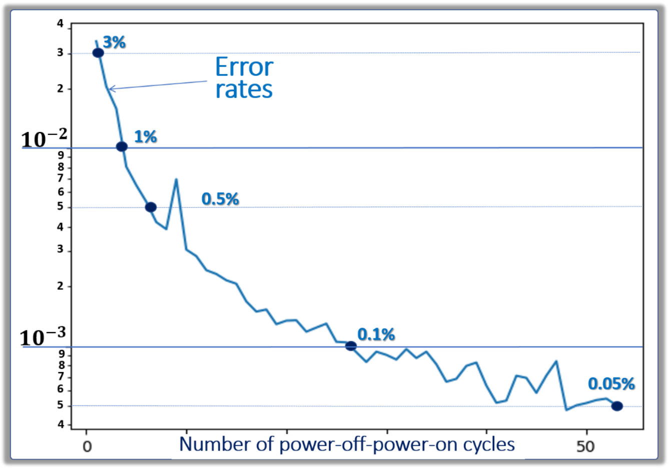

Fig. 3. Error rate reduction by eliminating the fuzzy cells after power-off/wait state/power-on cycles.

power-off/power-on cycles, to remove the erratic cells from the population. During each cycle, the cells that are flipping between 0, and 1 are removed. The estimated error rates of the remaining cells and plotted in Fig. 3. Initially, before eliminating the weak cells, the CRP error rates are in the 3 to 5% range. After four cycles the CRP error rates are reduced to about 1%. After 25 cycles the CRP error rate drops below 0.1%; then the reduction is slower. About 50 cycles are needed to get 0.05% error rate, and 1000 cycles to get 0.01%. Such error rates are still too high for cryptography. In the experimental work presented in this paper, the error rate before error correcting schemes can be adjusted from 3% to 0.01%, to study the respective efficiency of the schemes.

III. Fuzzy Extractors

A fuzzy extractor will extract a uniformly random string R and non-secret string P (helper data) from its initial input w. This mechanism will allow the string R which can be used as a key, to be reproduced exactly with the help of P, even though the input changes to some w' but remains close to w. Fuzzy extractors are said to be *information-theoretically secure* i.e., a crypto-system whose security is derived only from information theory, where a hacker will not have enough information to break the encryption, allows them to be used in cryptography. Fuzzy extractors are constructed using *Secure Sketch* (SS),

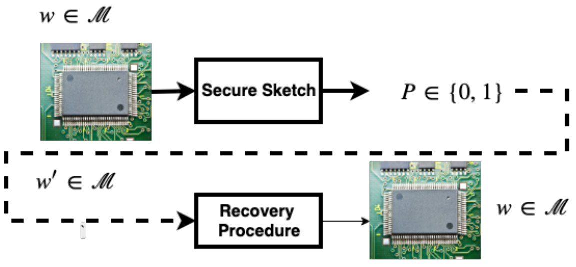

Fig. 4. Secure Sketch

which is a pair of randomized procedures "sketch" and "recover" which will allow precise reconstruction of the initial input from noisy input by making use of some helper data P.

In the "sketch" phase, *Helper Data* P is extracted from initial input w, which can be made publicly available. This output P will be used in the "recover" phase along with noisy input w' to recover w. This method is secure as the publicly available *Helper Data* reveals little to no information about w. Fig. 4 summarizes the description of a secure sketch.

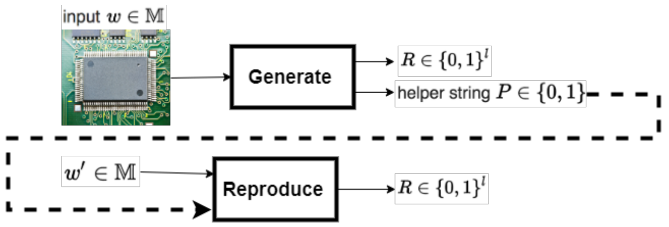

Fig. 5. Fuzzy Extractor

Fuzzy extractor is defined by a pair of randomized procedures "generate" and "reproduce". In the "generate" phase, the fuzzy extractor uses the "sketch" phase of the SS where *Helper data*, P and *Key*, R from the given input w. The "reproduce" phase uses the "recover" phase of the secure sketch which makes use of the *Helper data* to recover the original input w from a noisy input w' along with the random extractor used in the "sketch" phase, to extract the randomness from the recovered w. The ability to recover w from w' is highly dependent on the technique, usually an error correction scheme, used in the "sketch" phase of the Fuzzy Extractor. If the distance between the noisy input w' and input w is too large, it may not be possible to recover w from w'. Fig. 6 shows the construction of a Fuzzy Extractor using a secure sketch. Due to the error tolerance capability of Secure

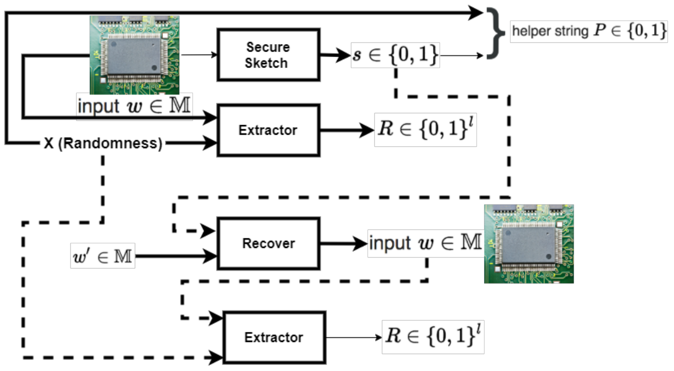

Fig. 6. Construction of a fuzzy extractor using secure sketch extractor

Sketches, their construction is based on error correcting codes. The error correcting code C is used to correct errors in w', even though w' may not be in C, by shifting the codeword. Two different constructions are used for secure sketch are provided [4] :

- Code-Offset Construction: For input w, select a uniformly random codeword $c \in C$, and set SS(w) to be the shift needed to get from c to w: $SS(w) = w - c$. To compute $Rec(w', s)$, subtract the shift s from w' to get $c' = w' - s$: decode c' to get c and compute w by shifting back to get $w = c + s$. When code C is linear, the information in s is essentially the syndrome of w.
- Syndrome Construction: The sketch SS(w) computes $s = syn(w)$, where *syn* is the syndrome. To recover the key, a unique vector e is chosen such that $syn(e) = syn(w') - s$ and output $w = w' - e$.

IV. Generation of cryptographic keys from PUFs using Fuzzy Extractors

Secret key generation using PUFs will allow users to produce a key from their own device which need not be stored in the devices' memory. Using PUFs to produce keys, will make the device unclonable and hence less susceptible to hacking. Using PUFs eliminates the complications and security issues related to key storage and distribution. Different PUFs have been used in the past to generate reliable and reproducible cryptographic keys using fuzzy extractors.
Different schemes have been proposed in the literature for generating reproducible keys using PUFs [26]–[31].Key generation scheme proposed in [27] uses BCH codes and random number generators for the construction of fuzzy extractors. The sole purpose of the BCH codes was to help reconstruct the PUF estimate from noisy PUF data, therefore, it will serve in the "secure sketch" phase of the Fuzzy extractor. This scheme is one of the primitive schemes and does not require any complex decoders, hence the error correction capability is not good. This scheme may be good for authentication where a certain error margin is tolerated. In key generation schemes, where the key will be used for encrypting or decrypting the data, the error in reproducing the key has to be zero.

Different fuzzy extractor architectures have been proposed in the literature, whose performances are compared in the Table III. In a recent work, a fuzzy extractor structure based on Polar codes for SRAM PUFs was proposed [35]. This work utilized complex Hash-Aided syndrome construction (SC) decoder to ensure that the key was reproducible. The results showed that the key was reproducible with a failure probability of 10^{-9} and utilized 896 helper bits for a key of 128 bits. This work utilized polar codes for extraction of key and helper data, thus utilizing the property of zero mutual information between the helper and key. In our previous work, we utilized serially concatenated polar and BCH codes to generate 250 bit keys with 262 helper data bits using SC decoder and Belief propagation decoders, while maintaining a comparable failure probability of 10^{-8} and 10^{-10}, respectively [36].

Here, we provide a proof of concept of key generation using SRAM using ternary state and compare the performance of the current state-of-the-art generators to see what is the trade-off between good error correction to achieve zero error key regeneration with the chance of getting false positive for a PUF which is not ideal in terms of intra-PUF distance. We would also like to determine the trade-off between the performance, the level of computations and delay, and the number of helper data bits needed to regenerate the key from Noisy PUF responses. The device can be utilized to improve the raw probability of error for each cell using the ternary state method. We can realise the need to use sophisticated error correction coding that will sufficiently correct the input bits, without increasing the false authentication rate of the device.

V. Experimental Results

As a proof of concept, the Cypress CY62256N was chosen as the SRAM to read from. It can store up to 32 KB of data, which was deemed more than sufficient source of entropy. A custom Arduino Mega 2560 shield was designed for the DIP variant to allow for easily switching between different packages. Every pin of the package is also connected to a toggleable switch such that a logical OFF corresponds to ground; without this design, the SRAM cannot be automatically controlled to dissipate the current.

First, the Arduino is programmed with three modes in mind: the first mode, upon receiving a special flag over the serial port over USB, reads the entire SRAM one byte at time, sends each back over the serial connection, and powers it down; the second mode, when receiving a different flag, will manually power the SRAM down; the third and last mode, when receiving neither of the former two flags, will interpret the value as a bit address and send back the read bit as the smallest possible word (in this case a byte). The last mode assumes to continuously read a series of bit addresses, until the second mode is triggered and powers off the SRAM to correspond to the PUFs lifetime.

A Python script communicates with the Arduino over the serial connection. By default, it indefinitely loops, requests the Arduino for the next PUF, and saves the entire contents of the SRAM onto a successive binary file. A separate script is then used to read every binary file and iterate over every cell based on the following criteria: if a cell's value has changed in any of the reads, it is from then on marked as an "X" to represent its unstable nature. If that cell has remained the same value (either 0 or 1) after iterating through every read, then that cell is given that value in the final enrollment. One can also tell the script to be less stringent, and instead set the final values based on the number of times the cell's value was 1 over the total number of reads. For instance, if the cell was 1 less than 30% of the reads, it is marked with "0"; if it was 1 more than 70% of the reads, then it is marked with a "1"; otherwise, it is marked with an "X". For the purpose of this experiment, the cell prone to remain constant at least 46%, it is marked with the constant ; otherwise it is marked with a "X".

We tested the various fuzzy extractor schemes in the literature using an SRAM PUF device produced by the NAU Cybersecurity lab shown in Fig. 7.

The SRAM device has a software package which allows us to interact with the device. It has various functions embedded in it, some of which include, reading the real time PUF

TABLE III
COMPARISON OF DIFFERENT FUZZY EXTRACTOR SCHEMES PROPOSED IN THE LITERATURE.

Fuzzy Extractor Construction	Key Length	Helper Data bits	Failure Probability	Flipping probability
Reed Muller Generalized Multiple Concatenated coding [32]	128	13952	10^{-6}	15%
BCH Repetition Code [33]	128	2052	10^{-9}	13%
Generalized Concatenated (GC) Reed Muller [34]	2048	2048	$5.37.10^{-10}$	14%
GC Reed Solomon [34]	1024	1024	$3.47.10^{-10}$	14%
Polar Codes SC [35]	128	896	10^{-6}	15%
HA SCL Polar Codes [35]	128	896	10^{-9}	15%
Serially Concatenated BCH and Polar codes using SC decoder [36]	250	262	10^{-8}	15%
Serially Concatenated BCH and Polar codes using Belief Propagation decoder [36]	250	262	10^{-10}	15%

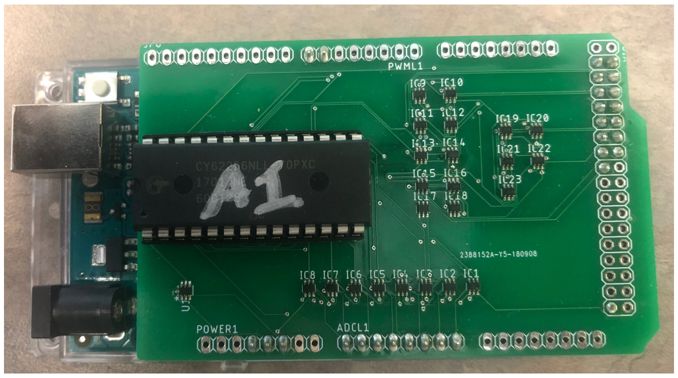
Fig. 7. SRAM Device

response after refreshing the SRAM every 2 seconds, counting the number of mismatches between every response, enrolling the PUF, where it utilizes 100 reads of the PUF to determine the stable cells in the SRAM device and stores the challenge in the server.
We use a technique called addressable PUF generator (APG), which allows us to extract a unique fingerprint from the device, by randomly selecting the cells which will be used as the fingerprint input to the fuzzy extractor.

A. Addressable PUF Generator

We use the technique of APG described in [37] to generate challenges for our PUF devices. In APG technique, a random number and the user ID are XORed and given as an input to a hash function. The hash digest is used to identify a location in the PUF. This location is given as an input to a pseudo random number generator, which will allow us to pick different memory cells from the device, giving a range of cells, which are randomly selected from the entire memory chip. Fig. 8, shows various cells selected in an SRAM fingerprint, using a random seed given to a pseudo random number generator (PRNG), allowing us to use bits spread across the finger print to extract the challenge and response.

B. Varying the error rate of the device

We use ternary PUF concept to mask the bits which are easily prone to flip in the memory device. The software allows us to manage the error rate by varying the number of reads used to average the SRAM challenge. We can mark the bits which are easily prone to flip with an "X". More number of reads used will lead to a more stable challenge as the bits prone to flip over a large number of reads will be marked with an "X". By this method, we can determine the cells which are able to produce stable "0" and "1" and eliminate cells with the "X". For example, the error rate when only 10 reads are used to average the challenge is higher when compared to using 100 reads to average the challenge.

C. Generating keys and storing them in the server

We simulated the fuzzy extractors proposed in the literature [32]–[35] in MATLAB. We utilize the fuzzy extractors to generate keys from challenges by varying their error rate by changing the number of reads used to average the challenges and mask the flipping bits with"X". We generate secret keys and helper data using the registration phase of the fuzzy extractor.

D. Regeneration of Keys using PUF response

Using the helper data produced in the key generation process, the key is regenerated. The ability of the Fuzzy extractor to regenerate the key under different error rates is evaluated. The False Authentication Rate and False Rejection Rate while using different Fuzzy extractors under various error rates is determined.
We use real-time SRAM responses of the device along with the helper data produced in the key generation phase to regenerate the key. The average error when an SRAM PUF is used to generate keys using the fuzzy extractors described in [4], [35], [36], [38] is reported in the Table IV. The FAR when a different user imposters as an authenticated user is also reported in the Table IV. To calculate the FAR, another PUFs response was utilized with the helper data provided by the registered challenge. The FAR was low when simpler fuzzy

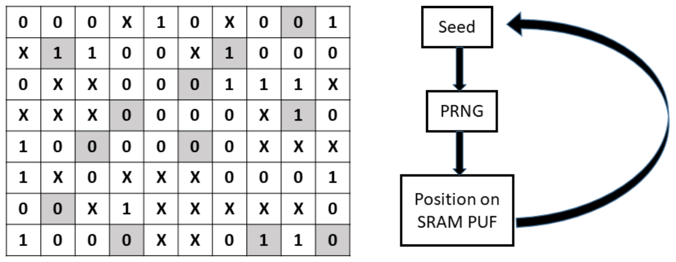

Fig. 8. Using Pseudo random number generator (PRNG) to select bits from the finger print of the SRAM Device

TABLE IV
EFFECT OF INTRA AND INTER PUF DISTANCE ON KEY GENERATION USING DIFFERENT FUZZY EXTRACTORS

Fuzzy Extractor	**% Error when the same PUF is used**	**% Error when different PUF is used**
BCH Fuzzy Extractor [4]	0.2 %	47.62%
Efficient Fuzzy Extractor using BCH [38]	12.5 %	48.98%
Polar using HA-SCL [35]	0 %	19.23%
Polar using serially concatenated BCH and Polar Codes [36]	0 %	23.68%

extractor methods like [4], [38] were utilized, but the FRR was high as they were not strong enough to correct the errors in the Noisy PUF. When stronger methods like [35], [36] were utilized, the FAR was higher compared to the weaker methods, as they were capable enough to correct more errors in the PUF, thereby reducing their FRR.

In Fig. 9, shows the ability of a Polar based Fuzzy extractor to derive the key at different error rates. The error rate of the SRAM is varied by using the ternary technique. Here, we used different number of cycles/reads to calculate the enrolled challenge, stored in the server. More number of bits will be marked with an “X” when more number of PUF responses are used to enroll the challenge. Marking the cell with an “X” indicates, removing the cell which flipped during one of the responses used to make the enrolled challenge to be stored in the server, thereby reducing the error of the PUF. The error in retrieving the key from Noisy SRAM responses using a fuzzy extractor was calculated for different error rates of the PUF. We observed the same pattern noted in Fig. 9 for 15 different SRAM devices. There is a clear decline in the error rate of the key as the error in the PUF reduces. The error in key tends to increase even when the PUF error is very low. This may be due to the fact that the error correction mechanism used in the Fuzzy extractor is over correcting the data, thereby changing the correct bits. This may be one of the factor for increase in the error of the key, even after decrease in the PUF error. Other factors may include different parameters in the error correction code. This is an ongoing work, therefore we will be working on testing each of the parameters of the error correction codes to check its effect on the key generation scheme.

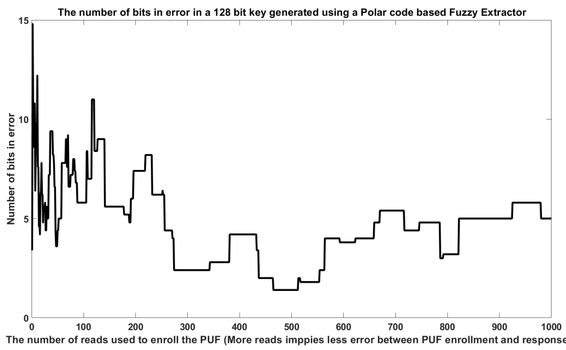

Fig. 9. Using Pseudo random number generator (PRNG) to select bits from the finger print of the SRAM Device

VI. CONCLUSIONS

In this paper we developed a proof of concept for a practical key generation scheme in IoT networks that does not rely on storing the private keys in the memory of the IoT devices, but rather generates a private key from the unique fingerprints of the embedded memory of the IoT devices on an on-demand basis. Such method can add another layer of security to the common cryptographic techniques (e.g., PKI) from physical attacks. Therefore, this method can offer a scalable security solution for large-scale IoT networks without the need to generate, distribute and store the private keys in billions of IoT devices.

We have utilized a recent idea of ternary-state PUFs rather than the traditional binary PUFs to identify the more reliable cells in the memory of the device that can substantially increase the robustness of the PUF responses for a very high sensitive application such as generating the private keys for encryption. The performance of this method has been tested for several SRAM chips using our device. We can vary the average error rate of PUF (i.e., selecting more stable cells which have lower probability of flipping) using different thresholds to define the ternary states, thus providing more stable PUF responses as needed. These responses can further be used to produce secret keys using fuzzy extractors. We tested the efficiency of the fuzzy extractors using real time SRAM PUF responses while varying its error to test the false authentication and false rejection rate. In summary, we like to note that the since SRAM PUFs depend on the flip flop logic and lack perfect symmetry, it is considered as a weaker PUF compared to some other memory-based PUFs (e.g. ReRAM based PUF), however, it’s a common NVM technology available in several IoT devices and offers an easy and accessible technology to explore.

REFERENCES

[1] M. Patton, E. Gross, R. Chinn, S. Forbis, L. Walker, and H. Chen, “Uninvited connections: a study of vulnerable devices on the internet of things (iot),” in *2014 IEEE Joint Intelligence and Security Informatics Conference*. IEEE, 2014, pp. 232–235.

[2] E. Marin, D. Singelée, B. Yang, V. Volski, G. A. Vandenbosch, B. Nuttin, and B. Preneel, “Securing wireless neurostimulators,” in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. ACM, 2018, pp. 287–298.

[3] H. Taylor, “How the ’internet of things’ could be fatal,” Mar 2016. [Online]. Available: https://www.cnbc.com/2016/03/04/how-the-internet-of-things-could-be-fatal.html

[4] Y. Dodis, L. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 523–540.

[5] S. L. Kinney, *Trusted platform module basics: using TPM in embedded systems*. Elsevier, 2006.

[6] E. Barker and A. Roginsky, “Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths,” *NIST Special Publication*, vol. 800, p. 131A, 2011.

[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*. ACM, 2002, pp. 148–160.

[8] 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*, 2018.

[9] B. Cambou and D. Telesca, “Ternary computing to strengthen cybersecurity,” in *Science and Information Conference*. Springer, 2018, pp. 898–919.

[10] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical one-way functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[11] Y. Gao, D. C. Ranasinghe, S. F. Al-Sarawi, O. Kavehei, and D. Abbott, “Emerging physical unclonable functions with nanotechnology,” *IEEE access*, vol. 4, pp. 61–80, 2016.

[12] 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.

[13] R. Maes and I. Verbauwhede, “Physically unclonable functions: A study on the state of the art and future research directions,” in *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 3–37.

[14] Y. Jin, “Introduction to hardware security,” *Electronics*, vol. 4, no. 4, pp. 763–784, 2015.

[15] M. Delavor *et al.*, “Puf based solution for secure communication in advanced metering infrastructure,” *ACR publication*, 2014.

[16] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, “Physical unclonable functions and public-key crypto for fpga ip protection,” in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. IEEE, 2007, pp. 189–195.

[17] J. Plusquellic *et al.*, “Systems and methods for generating pufs from non-volatile cells,” 2015.

[18] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-up sram state as an identifying fingerprint and source of true random numbers,” *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[19] R. Maes, P. Tuyls, and I. Verbauwhede, “A soft decision helper data algorithm for sram pufs,” in *2009 IEEE international symposium on information theory*. IEEE, 2009, pp. 2101–2105.

[20] T. A. Christensen and E. S. I. John, “Implementing physically unclonable function (puf) utilizing edram memory cell capacitance variation,” Oct. 30 2012, uS Patent 8,300,450.

[21] P. Prabhu, A. Akel, L. M. Grupp, S. Y. Wing-Kei, G. E. Suh, E. Kan, and S. Swanson, “Extracting device fingerprints from flash memory by exploiting physical variations,” in *International Conference on Trust and Trustworthy Computing*. Springer, 2011, pp. 188–201.

[22] A. Chen, “Comprehensive assessment of rram-based puf for hardware security applications,” in *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2015, pp. 10–7.

[23] B. Cambou, F. Afghah, D. Sonderegger, J. Taggart, H. Barnaby, and M. Kozicki, “Ag conductive bridge rams for physical unclonable functions,” in *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 151–151.

[24] X. Zhu, S. M. Millendorf, X. Guo, D. M. Jacobson, K. Lee, S. H. Kang, and M. M. Nowak, “Physically unclonable function based on resistivity of magnetoresistive random-access memory magnetic tunnel junctions,” Mar. 12 2015, uS Patent App. 14/077,093.

[25] E. I. Vatajelu, G. D. Natale, M. Barbareschi, L. Torres, M. Indaco, and P. Prinetto, “Stt-mram-based puf architecture exploiting magnetic tunnel junction fabrication-induced variability,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 1, p. 5, 2016.

[26] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *Proceedings of the 44th annual design automation conference*. ACM, 2007, pp. 9–14.

[27] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura, “Performance analysis for puf data using fuzzy extractor,” in *Ubiquitous Information Technologies and Applications*. Springer, 2014, pp. 277–284.

[28] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, “Reconfigurable physical unclonable functions-enabling technology for tamper-resistant storage,” in *Hardware-Oriented Security and Trust, 2009. HOST’09. IEEE International Workshop on*. IEEE, 2009, pp. 22–29.

[29] T. Ziola, Z. Paral, S. Devadas, G. E. Suh, and V. Khandelwal, “Authentication with physical unclonable functions,” Jul. 15 2014, uS Patent 8,782,396.

[30] B. Škorić, P. Tuyls, and W. Ophey, “Robust key extraction from physical uncloneable functions,” in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 407–422.

[31] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, “Extracting secret keys from integrated circuits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[32] R. Maes, P. Tuyls, and I. Verbauwhede, “Low-overhead implementation of a soft decision helper data algorithm for sram pufs.” in *CHES*, vol. 9. Springer, 2009, pp. 332–347.

[33] R. Maes, A. Van Herrewege, and I. Verbauwhede, “Pufky: A fully functional puf-based cryptographic key generator,” *Cryptographic Hardware and Embedded Systems–CHES 2012*, pp. 302–319, 2012.

[34] S. Puchinger, S. Müelich, M. Bossert, M. Hiller, and G. Sigl, “On error correction for physical unclonable functions,” in *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding; Proceedings of*. VDE, 2015, pp. 1–6.

[35] B. Chen, T. Ignatenko, F. M. Willems, R. Maes, E. van der Sluis, and G. Selimis, “High-rate error correction schemes for sram-pufs based on polar codes,” *arXiv preprint arXiv:1701.07320*, 2017.

[36] A. R. Korenda, F. Afghah, and B. Cambou, in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*.

[37] B. Cambou, “Encoding ternary data for puf environments,” Nov. 29 2018, uS Patent App. 16/036,477.

[38] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura, “Cryptographie key generation from puf data using efficient fuzzy extractors,” in *Advanced Communication Technology (ICACT), 2014 16th International Conference on*. IEEE, 2014, pp. 23–26.