NAND Flash-Based Digital Fingerprinting for Robust and Secure Hardware Authentication

***Abstract*—In today's world, embedded systems and microcontroller-based modules have become increasingly integrated into our daily lives. However, the security of these embedded devices and the assurance of hardware authenticity have raised concerns within the expanding realm of the Internet of Things (IoT). In this study, we established an experimental environment to observe the disturbance of SLC flash memory programming for the purpose of designing a physical unclonable function (PUF). Our findings revealed that intra-page disturbance is more easily generated compared to inter-page disturbance. Additionally, we observed a pairing pattern where adjacent pages are paired in a (2n, 2n+1) manner, and disturbances only occur within a pair. Finally, we discovered that as the page number increases from 0 to 63, it becomes progressively more challenging to detect the initial bit flip within a page, thereby making it more difficult to achieve a stable disturbance state.**

I. Introduction

Authentication has emerged as a potential solution to address the integrity concerns in the IoT ecosystem. As traditional methods of distinguishing and identifying individuals based on physical traits became increasingly challenging, biometric authentication stepped in to fill the gap [1]. Similarly, electronic devices such as RFIDs often share a similar physical appearance. The primary distinguishing factor among them lies in the unique identification (ID) stored in the chip memory. However, the vulnerability of this ID makes it susceptible to attacks and compromises. PUF is a technique used to establish a digital fingerprint on physical semiconductor devices. This fingerprint is highly unique and remains mostly unaffected by variations in temperature, humidity, or stability. Research that investigates PUF from volatile memories and non-volatile memories can be found in [2] and [3] respectively.

In electronic devices, memory plays a crucial role in storing information. With the advantages like lower power consumption, faster write and erase times (high access speed), and a low cost per bit, NAND flash stands out as one of the most widely used among all. These qualities make NAND Flash a preferred choice in many electronic devices such as USB drives, media players, digital cameras, and smartphones.

With the introduction of PUF, IoT security, especially in healthcare domain [4], [5], can be enhanced as PUF provides a unique, unclonable ID. Additionally, if data is stored with PUF encryption on memory, the data is secured as it is not possible to decrypt without the PUF. Moreover, PUF can generate random output values, and there are numerous applications of a good and reliable random number generator. This function is unpredictable for even an attacker with physical access to the system. Also, It is impossible to produce a copy of the same physical system even when the functionality is known. PUFs offer a unique signature [6], [7], which comprises the bits that remain stable regardless of the number of read and write operations performed on a particular memory cell. This extracted signature serves as a mean to authenticate a chip and generate a random number suitable for cryptographic key generation.

In this paper, we specifically conducted a novel PUF extraction using the Samsung NAND Flash Memory (K9F1G08U0E) and the Flexible Memory Controller (FMC) interface of the experimental discovery board (STM32F429ZIT6). The software utilized was STM-CUBE32IDE, and we employed the C programming language. Our work focused on exploring various new program disturb algorithms and applying them to the NAND flash for PUF extraction purposes. We also evaluated the performance of these algorithms during the extraction process.

II. Related Works

Shijie et al. [8] introduced a PUF-based key generator specifically designed for NAND flash chips. Their approach involved proposing three distinct methods for extracting raw PUF output numbers. These methods included utilizing a position map, partial programming, and partial erasure techniques to identify and select reliable keys from the PUF output. Prabhu et al. [9] tested seven NAND flash-based PUFs that leverage program disturb, read disturb, and program operation latency. They conducted experiments involving fourteen devices to evaluate the proposed PUFs. The evaluation process employed Pearson correlation as a metric to assess the

robustness of the generated signatures. The results indicated that NAND flash PUFs based on program latency and program disturb exhibited the highest utility and effectiveness among the tested approaches. They have observed that NFPUF based on program disturb is the best one to distinguish between different chips but takes more time for extraction and NFPUF based on program latency is the fastest among all PUFs. Cai et al. [10] demonstrated the effects of two-step programming on MLC NAND flash. Their work exposed the dangers of two-step programming and how it could be exploited. The process of two-step programming involves programming a single cell's LSB and MSB at different times. Most of the existing work is limited to analyzing and generating PUF for a single NAND flash chip either SLC or MLC. In this paper, we will observe SLC Single Page Program disturbance with different approaches.

III. Background

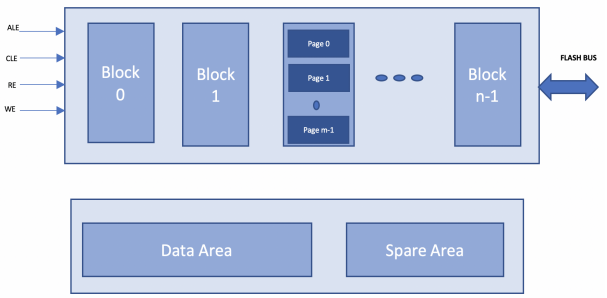
A. NAND Flash

The NAND flash cell is made of a floating gate transistor, where the electrons held by the floating gate decide the threshold voltage (V_{th}) of the cell. Normally, to switch on an NMOS transistor we need $V_g > V_{th}$. During the program stage, if we inject electrons in the floating gate, we will require a relatively higher voltage (V_g) to offset the negative charge of electrons and turn on the channel ($V_g - V_\delta > V_{th}$). If there are no electrons captured in the floating gate, a relatively small gate voltage is sufficient to make $V_g > V_{th}$ and turn on the transistor. NAND Flash memory is a non-volatile storage technology that retains data even without power. A metal-oxide-semiconductor known as the Floating gate provides additional charges to the memory cell, which helps in preserving the stored data in the absence of power. Distinguished by the number of bits stored per cell, NAND flash storage is available in various types, including SLC (Single-Level Cell), MLC (Multi-Level Cell), TLC (Triple-Level Cell), QLC (Quad-Level Cell), and 3D NAND. NAND Flash memory organizes data into blocks, with the block being the fundamental unit for erase operations, while the page serves as the basic unit for write operations. Each page comprises a data area and a spare area. Flash memory organization is described in Figure‘1.

B. PUF within NAND Flash Memories

The dense packing of flash cells makes process variations in feature geometry have a notable impact on the interaction between neighboring cells and the behavior of individual cells. As a result of these variations, certain cells may exhibit varying levels of susceptibility to write/read disturbances and wear. PUFs offer a distinctive signature consisting of stable bits that remain unchanged regardless of the number of read and write operations performed on a specific cell. This unique signature serves multiple purposes: it can authenticate a chip, verify its integrity, and generate a random number suitable for cryptographic key generation. The variations in signatures occur between blocks and pages within the chip. Consequently, the chip can possess multiple extractable unique signatures in different locations.

Fig. 1: Flash Memory Organization



C. PUF Generation Techniques

Various techniques exist for deriving raw PUF output values from NAND Flash memories. Firstly, Program disturb, a block gets wiped out first, followed by continuous programming of a single page within that block. The pages adjacent to the programmed page must be read between each programming sequence to detect errors induced by the Program disturb. For every bit on the neighboring page, the count of programming instances required to generate the first bit error is documented. This information collectively forms the signature. The extraction of a signature necessitates one erasure and numerous programming operations, but these procedures do not compromise the reliability of the chip. The disadvantage of program disturb technique is that it causes irreparable damage to the page, and it is quite slow.

The second technique is known as Read disturb. With this method, the entire block is initially erased and subsequently programmed with random data. After this, each page is read multiple times, often reaching several million reads in order to generate a read-disturb. Following every 1,000 read cycles, every page in the block is examined for potential errors. If an error is detected on a page, the bit, the page, and the error's

cycle are all recorded. This procedure is repeated up to 10 million times. As a result, the read cycle counts for all the bits within the block are utilized as the signature. While the Read disturb method is less destructive and generates less signature noise compared to the Program disturb method, it is a slower process. Another technique is called the Program operation latency; each bit is programmed individually on a page, keeping track of the latency for each operation. The benefit of this method is its speed, as well as the fact that it does not result in wear and tear on flash device.

IV. Setup and Implementation

A. Hardware Setup

The STM32F429I and NAND Flash are connected using female jumper wires. Among the 16 required connections to the NAND Flash, there are 8 data pins, 6 control pins, Vcc (Voltage supply), and GND (Ground). The same 8 data pins on the NAND Flash are used to transmit command, address, and data. The 6 control pins on the NAND Flash include: CLE (Command Latch Enable), ALE (Address Latch Enable), R/B (Ready/Busy Output), WE (Write Enable), RE (Read Enable), and CE (Chip Enable). Power is supplied to the NAND Flash from the microcontroller by connecting the Vcc 3V of the microcontroller to the Vcc of the NAND Flash, and the GND of the microcontroller to the GND of the NAND Flash. The corresponding FMC (Flexible Memory Controller) pins from the microcontroller are connected to the control and data pins of the NAND Flash. This integrated development environment includes the STM32CubeMX graphic tool and STM32CubeIDE.

B. Software Setup and Configuration

STM32CubeIDE, an integrated development environment, is utilized for the creation of software code in the C programming language. It comes equipped with the STM32CubeMX graphic tool, which allows for board visualization, pin selection, and configuration. Furthermore, STM32CubeIDE offers the capability to generate definitions and library files that correspond with the pins configured via the graphic tool.

As a part of the setup process, we first initialize the Hardware Abstraction Layer (HAL), followed by the configuration of the System clock. Subsequently, other peripherals such as GPIO, FMC, and USART1 are initialized. Next, we create an instance of a NAND Controller and configure the required setup values. These values include SetupTime, WaitTime, NandBank, ECC-computation, BlockSize, PageSize, and more.

C. Erase and Read Operations

Before programming the page, an erase operation begins by issuing an Erase setup command (60h), followed by the row address. While setting the R/B pin to low, the Erase confirm command (D0h) then initiates the internal Block Erase operation. In the Write Status bit (I/O 0), a value of 0 indicates that the erase operation has been completed, whereas a value of 1 signifies an error in the erase operation. The read operation is initiated with the opcode 0x00 while setting the CLE pin to HIGH, and the CE pin to LOW. Then the row and column address is provided. After signaling the completion of writing the address by using the opcode 0x30, the device now knows which bytes to read from its flash array. The data from the address is loaded onto the page register and sent through each of the 8 I/O pins to be read by the STM32F4 micro-controller.

D. Program Operation

At the page level, the program operation is initiated by issuing a Serial data input command (80h), followed by column and row addresses. Subsequently, the transmitter buffer, which comprises the data to be written on the page is set and passed in the next cycle. The NAND instance is initialized with parameters such as frequency, setup timing, and hold timing. A Read operation is performed to verify the data that has been written. The column and row addresses are passed in the following 4 cycles. The data is then read in the subsequent cycle. The output of the Read Page operation is stored in the receiver buffer, which holds the data that has been read from the page, thus initiating the programming process. While setting the R/B pin to low, the status of write operation is checked by issuing command 70h.

V. Methodology and Approach

A. Algorithm

After evaluating various methods for signature extraction, we elected to utilize the Program disturb technique for our project. We decided based on the higher frequency of disturbance observed in cells compared to the Read disturb method. Program disturb appears earlier, usually around ten thousand iterations, while Read disturb doesn't show significant disturbance within ten thousand iterations; more iterations are needed to detect Read disturb. Hence, we opted for the program disturb method.

B. Procedure

We identified stable bits within the NAND Flash memory in a variety of ways: on the same page, on the

adjacent page, and across all pages within the block. The algorithm 1 and 2 outlines the NAND flash ID extraction using single page program disturb and multi page program disturb respectively. It is important to note that Error Correction Code (ECC) needs to be disabled so that it would not automatically correct any errors and interfere with our process.

1) **PUF observing the same page:** Firstly, regarding the method of locating stable bits on the same page, we began by erasing block zero and programming page one with the value AA in hexadecimal. We then tracked the bit changes happening on the same page by comparing the bit values on that page with AA. If the value differed, it signified that the bit was not stable. If not, they were deemed stable bits and were stored in an array. This process of programming and the subsequent steps were repeated over ten thousand iterations. By the end of these iterations, we had stable bits that could generate a PUF. The technique is described in Algorithm 1.

2) **PUF observing the adjacent pages:** The second approach bears similarities to the first, but this time we are reading from the adjacent pages. In this method, we erased block zero, and then programmed page two with the hexadecimal value AA. Subsequently, we read from the adjacent pages, which are page one and page three. For these adjacent pages, we compared their bits with the value FF (FF is the value resulting from the erase operation, which sets all bits to ones). If the value remains the same, it indicates stability. However, if it changes, the bit is considered unstable. We then stored the stable bits in an array, and continued comparing them through each of the ten thousand iterations. During each iteration, we wrote to the same page, page 2, and read bits from page 1 and page 3.

3) **PUF observing a number of pages:** In the final approach, we erased block zero and programmed all pages with the value AA. We then selected page one and programmed it again with AA ten thousand times. Subsequently, we read the adjacent pages, which are page zero and page two, and compared their bit values with AA. If there was any change, the bit was deemed unstable; otherwise, it was considered stable and stored in an array. We then proceeded to the next page, page two, and programmed it ten thousand times with AA. The adjacent pages, page one and page three, were then read, and the values were compared with AA. If there was no change, they were added to the stableBits array. We applied the same steps to all 64 pages in the block. The technique is described in Algorithm 2.

Algorithm 1: NAND Flash ID Extraction Using Single Page Program Disturb

```
Result: Read the flipped bits
while While true do
    for i ← 1 to 64 do
        Write 0xAA to every byte of the page 0;
        Read page 0 and store to a buffer;
        Print 128 bytes of buffer through USB CDC
          stack;
        Delay 100 ms
    end
end
```

Algorithm 2: NAND Flash ID Extraction Using Multi-page Program Disturb

```
for i ← 1 to 64 do
    Erase Block
    for j ← 0 to 10k do
        Program Page i is programmed by 0XAA if
        (j+1) % 1000 then
        end
        Read page (i-1), i, and (i+1)
    end
end
```

VI. Experimental Results

A. PUF Identification

1) **PUF observing the same page:** Algorithm 1 has been implemented on the Discovery board to showcase single-page program disturbance. This technique was tested on two different NAND Flash devices. Both devices used the same page programming and observed the page for program disturbance. The number of iterations conducted on both devices was 10,000. During the experiment, stable index bytes were identified for each NAND Flash device. The number of stable bytes identified differed between the two devices, and the indexes of the stable bytes varied as well. Interestingly, through the experiment, it was observed that after approximately 306 iterations, the bits began to flip and the situation deteriorated significantly thereafter. The above observation is explained as follow:

Flash memory can only transition from a state of 1 to 0 during the write process. Once it has reached the state of 0, it can only be switched back to 1 through an erasure. Initially, each cell of the flash memory is programmed with 1s, and then repeatedly overwritten with 0xAA to each byte. In an ideal scenario, the constant read output would be 0xAA. However, after several iterations, some bits unavoidably flip from 1 to 0 and remain so. Since we did not implement an erase operation at each step,

these irreversible bit flips accumulated over time, leading to an increasingly disordered state.

2) **PUF observing the adjacent pages:** After executing 10,000 cycles of writing and reading, we successfully determined the count of stable bytes. To evaluate if the results were unique, we carried out similar operations on two different devices. The outcome showed that all the bytes on page number two in both devices remained stable. However, only a finite number of bytes were stable on page number 4.

3) **PUF observing a number of pages:** We utilized Algorithm 2 to examine both intra-page and adjacent-page disturbances across pages 1 to 64. However, in this study, our evaluation was confined to 7 pages. Table I shows the iteration point at which program disturbs on a page starts to affect itself and its neighboring pages. We can see that, disturbances from page 1 starts affecting Page 0, which occurs within less than 1,000 write iterations. Interestingly, Page 2 remains undisturbed. This leads us to infer that disturbances only occur in paired pages. However, this characteristic is dependent on the physical layout of Samsung's SLC NAND flash memory and the proximity between two of every three pages.

The page distance is close enough to trigger bit flipping, while the pairing distance is far enough to prevent one pair from interfering with another. Pages tend to become more resistant to bit flipping as their distance from page 0 increases. While we lack exact information about the physical layout of Samsung's SLC NAND flash memory, we hypothesize that page 0 is nearest to the block's voltage source (or plane), given that the page pairs are stacked vertically. The speed at which program disturbances occur is not only dictated by the physical proximity of the pages, but also by the heat generated from the voltage source. As we ascend the vertical page stack, heat dissipation occurs, resulting in an exponential increase in the number of iterations to generate program disturbances. This increases the complexity of signature extraction in the ensuing manuscript, a subject we will delve into as we compare the signature extraction of pages 1 and 2. Following data recording into log files, we created a Python script to analyze the data and visualize our findings through plots.

Our analysis confirmed that the pages are physically programmed in pairs, conforming to the equation (2n, 2n + 1), where 'n' represents the pair number (0 to 31). Therefore, programming to page 0 only disturbs page 1, programming on page 2 exclusively affects page 3 and vice versa. Further analysis reveals that the instances of program disturbances necessitate more iterations as we progress from page 0. This pattern is evident when we plot Table I in Fig. 2. We have applied an exponential curve fit to both inter- and intra- page disturbances, which facilitates the estimation of when a given page and its pair will experience their first bit flips.

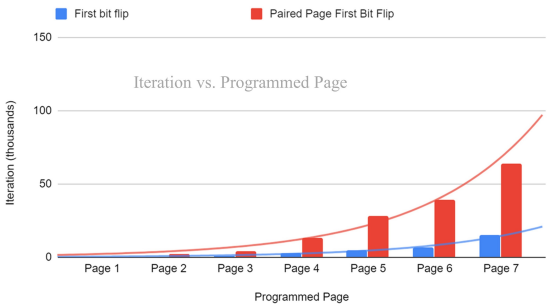


Fig. 2: Estimating Bit Flip Pattern.

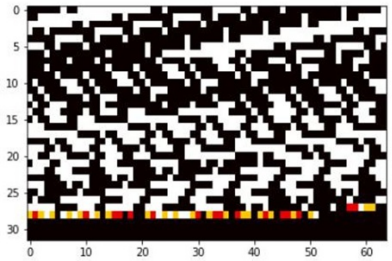


Fig. 3: Page 1 signature after 100k multi-page writes.

The black colored entries in the heatmap from Fig. 3 shows bytes that have completely flipped. Meaning after erasing the page, all the bits in a byte have changed from 0xFF to 0x00. The white entries show erased bytes that remained 0xFF after page erase. Using the non black colored entries, we can use these markings to show the bytes and bits of a page that have not flipped from program disturbs. After 83k iterations, the signature became stable. The colored entries in Fig. 4 represents bit flips within a byte. In Page 2's signature, it is quite evident that all the bits in each of the 2,048 bytes do not completely flip. Only a few of the 8 bits flip. We can use the colored entries to show the bytes. Similar to page 1's heatmap where the black colored entries represent the flipped bytes and white entries show stable bytes, we observe that 100k iterations is not enough program writes to observe a stable signature specifically for page 2. Examples can be seen in Fig. 3 and Fig. 4. The higher numbers are represented by warmer colors, which are indicative of bits that were highly resistant to disturbance and only flipped after running the experiment for high number of cycles.

Program on Page n	Page 1	Page 2	Page 3	Page 4	Page 5	Page 6	Page 7
First bit flip in Page n-1	<1	No bit flip	4	No bit flip	28	No bit flip	64
First bit flip in Page n	<1	<1	1	3	5	7	15
First bit flip in Page n+1	No bit flip	2	No bit flip	13	No bit flip	39	No bit flip

TABLE I: Iteration points of disturbances on individual pages and their neighbors (in thousands).

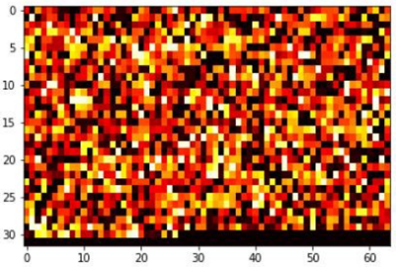


Fig. 4: Page 2 signature after 100k multi-page writes.

VII. Conclusion

PUF can play a significant role in strengthening the security of hardware and systems. The strength of a PUF against attacks is directly linked to the uniqueness of its generation process. In this paper, we employed various techniques of program disturb to extract PUF from NAND flash memory devices. The PUF was captured on the same page, on adjacent pages, and across all pages. As we observed, a large number of bits remain stable even after 50,000 operations when performing bit-wise operations on data read from a page. To manage this, we designed an algorithm that can execute the outer loop for 1,00,000 writes and ascertain the number of stable bits to an array after every 10,000 writes. Consequently, we will have 10 arrays of stableBits [0-9]. By comparing all of these arrays against each other, we can further reduce the number of stable bits. We then utilize these stable bit information to generate PUF. To proof the PUFs uniqueness, we conducted our research on two different NAND flash memory devices. The results from these two devices were then compared to examine the signature differences. We observed that PUF generated from both devices are different and unique. This extracted PUF could potentially be utilized as a key to strengthen hardware authentication and system security.

References

[1] N. Karimian, P. A. Wortman, and F. Tehranipoor, "Evolving authentication design considerations for the internet of biometric things (iobt)," in *Proceedings of the eleventh IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis*, 2016, pp. 1–10.

[2] 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, 2008.

[3] H. Gordon, J. Edmonds, S. Ghandali, W. Yan, N. Karimian, and F. Tehranipoor, "Flash-based security primitives: Evolution, challenges and future directions," *Cryptography*, vol. 5, no. 1, p. 7, 2021.

[4] P. A. Wortman, F. Tehranipoor, N. Karimian, and J. A. Chandy, "Proposing a modeling framework for minimizing security vulnerabilities in iot systems in the healthcare domain," in *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. IEEE, 2017, pp. 185–188.

[5] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "Investigation of dram pufs reliability under device accelerated aging effects," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[6] W. Yan, C. Jin, F. Tehranipoor, and J. A. Chandy, "Phase calibrated ring oscillator puf design and implementation on fpgas," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[7] W. Yan, F. Tehranipoor, and J. A. Chandy, "Puf-based fuzzy authentication without error correcting codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1445–1457, 2016.

[8] S. Jia, L. Xia, Z. Wang, J. Lin, G. Zhang, and Y. Ji, "Extracting robust keys from nand flash physical unclonable functions," in *International Conference on Information Security*. Springer, 2015, pp. 437–454.

[9] 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.

[10] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in mlc nand flash memory programming: Experimental analysis, exploits, and mitigation techniques," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 49–60.