NAND Flash Memory Devices Security Enhancement Based on Physical Unclonable Functions

Siarhei S. Zalivaka and Alexander A. Ivaniuk

1 Multimode Physical Unclonable Function as an Entropy Source for Generating True Random Bits

1.1 Introduction

True random number generators (TRNGs) are used in a wide range of applications (e.g., cryptography, statistical sampling, simulation, computer games, etc.) [51]. TRNG can be implemented as a part of NAND flash memory device controller and used to support Trusted Computing Group (TCG) standard [1]. The main advantage of TRNGs comparing to pseudorandom number generators (PRNG) is the uniqueness and unpredictability of their produced output values. TRNG is a device or a part of a device that generates random numbers based on some intrinsic physical process. One of the possible ways of extracting random data from electronic devices is to implement physical unclonable functions (PUFs) (e.g., [50, 52]).

Nowadays physical unclonable functions (PUFs) are becoming ubiquitous cryptographic primitives as an alternative to classical cryptographic algorithms in compact digital devices [2]. Main semiconductor manufacturers actively introduce them into their IoT solutions [3], cutting-edge field programming gate array (FPGA) chips [4], authentication protocols [5], etc. In general, PUF can be represented as a mapping of external inputs (challenges) to the outputs (responses). This mapping is called challenge-response pair (CRP) set, which is unique for each integrated circuit

S. S. Zalivaka (✉)
SK hynix memory solutions Eastern Europe, Minsk, Republic of Belarus
e-mail: sergey.zalivako@sk.com

A. A. Ivaniuk
Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus
e-mail: ivaniuk@bsuir.by

A. Iranmanesh (ed.), *Frontiers of Quality Electronic Design (QED)*,
https://doi.org/10.1007/978-3-031-16344-9_1

(IC) containing a PUF block even if the design and layout are the same [6]. This can be explained by intrinsic manufacturing process variations introduced during fabrication. Since physical properties of an IC may vary depending on temperature or voltage, some of the PUF response values are unstable. As a result, CRP set can be split into stable and unstable subsets and can be utilized for identification and random number generation, respectively.

PUF designs can be based on different physical phenomena, e.g., delay values [7], threshold voltages [8], operating frequencies [9], image sensor noise patterns [10], etc. Another subset of PUFs is utilizing memory to extract uniqueness from IC, e.g., SRAM PUF [11], DRAM PUF [12], Butterfly PUF [13], SR-Latch PUF [14], etc. NAND flash memory devices can be also successfully used to implement a PUF because some intrinsic effects, e.g., threshold voltages, erase times, bad block characteristics, program/read disturb, etc., uniquely characterize a memory device [15].

The proposed PUF design is based on using an inverter and a D-Latch which are controlled by enable (EN) signal. This circuit can operate in four modes, namely, initial memory, ring oscillator, metastability, and latch modes. All these modes can be used for different purposes, i.e., generating a unique identifier in initial memory mode, generating random numbers in ring oscillator or metastability modes, and storing generated ID or random value in latch mode. Thus, the proposed PUF design supports both PUF routines in a single device. One of the main challenges in TRNG design is consumed area and performance (rate of random bit generation). The proposed TRNG design is compact as it consumes a latch and an inverter gate and fast as ring oscillator mode operates on high frequency.

1.2 General Description of a Circuit

The proposed entropy source for random bit generation includes two elements, namely, Latch D-type (LD) and Inverter (INV). As shown in Fig. 1, INV is connected to LD and forms a negative feedback loop. The operation of this circuit is controlled by enable (EN) signal.

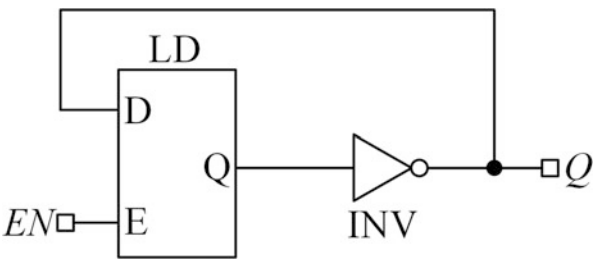

Fig. 1 Entropy source circuit

The proposed PUF supports four modes of operation:

1. **Initial memory**. This mode works only during start-up and EN="0". This is equivalent to SRAM PUF as the LD can generate either stable "0" or stable "1" or metastable value. According to SRAM PUF research [16], the output Q has a 10% chance of generating metastable value. If Q is stable, it can be used as a bit of a unique device ID.
2. **Ring oscillator (RO)**. If enable signal is kept as EN="1", the PUF will produce a meander signal with a unique frequency F_i (i is an index of individual entropy source) which is utilized for random bit generation similarly to RO PUF [17].
3. **Metastability**. Since LD is asynchronous and the value on data input (D) of LD is unpredictable, changing EN signal value from "1" to "0" can violate timing parameters of LD. In this case, LD may fall into a metastable state and the output Q can be either "0" or "1".
4. **Latch**. If enable signal is kept as EN="0", the LD stores random bit and the output Q value is stable.

As a result, the proposed PUF design can be used to generate unique stable ID bit (mode 1) or random bits (modes 2 and 3) or store generated ID or random bit (mode 4).

1.3 Operation of the Entropy Source

The circuit mentioned above (see Fig. 1) can be represented on gate level as shown in Fig. 2. The proposed circuit is named ROLD as a combination of different PUFs, i.e., ring oscillator and Latch D-type.

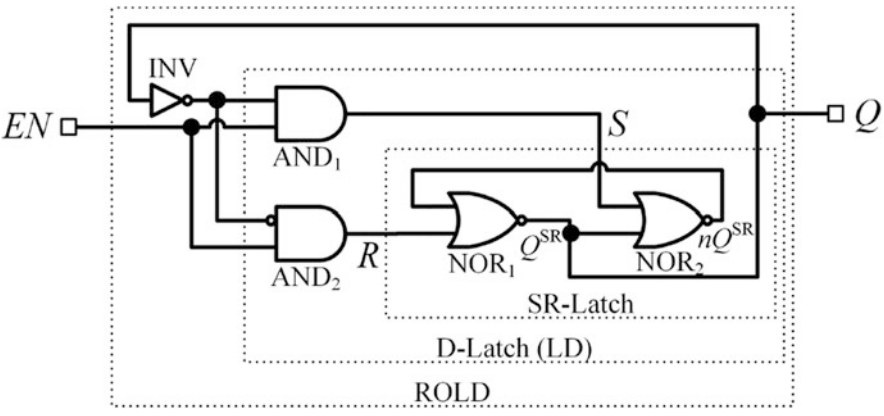

Fig. 2 Gate level of the entropy source circuit

The D-Latch component consists of basic SR-Latch circuit which has S (set), R (reset) inputs, and two complementary data outputs Q^{SR} and nQ^{SR}. In the case when SR-Latch is designed on NOR2 gates (NOR_1 and NOR_2 in Fig. 2), it has four operation modes: Setting "1" (when S="1" and R="0"), Resetting "0" (when S="0" and R="1"), Storing value (when S="0" and R="0"), and Forbidden mode (when S="1" and R="1"). The transaction from Forbidden state to Storing mode may cause generating metastable value on outputs Q^{SR} and nQ^{SR}. The D-type Latch is designed on the base of SR-Latch in such a way to prevent the occurrence of Forbidden mode by keeping S and R inputs in opposite values. The Storing mode is provided by additional input EN of enable signal and two additional AND2 gates (AND_1 and AND_2 in Fig. 2).

Let us describe equivalent circuits which are operating during four modes.

1.3.1 Initial Memory

When EN="0", the proposed circuit is equivalent to SR-Latch in storing mode (S="0", R="0"), which is shown in Fig. 3.

In this mode, AND elements (AND_1 and AND_2) generate constant "0" value and can be omitted for analysis of this circuit. NOR elements (NOR_1 and NOR_2) operate as inverters. Therefore, circuit in this mode operates as a bistable element as shown in Fig. 4.

During initialization (power-up) stage, the default value v is unknown due to manufacturing process variations (possible asymmetry of NOR gates NOR_1 and NOR_2 and connection wires between them). Therefore, unique ID values can be obtained from this PUF during power-up similarly to SRAM cells which are also based on bistable elements [16].

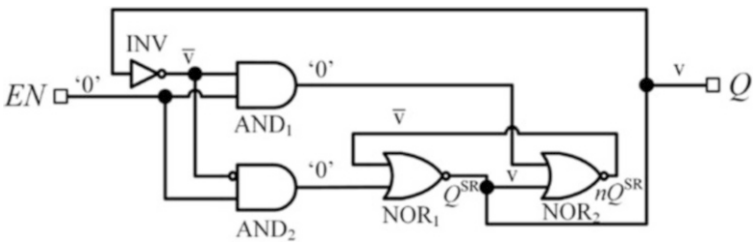

Fig. 3 ROLD circuit (EN="0")

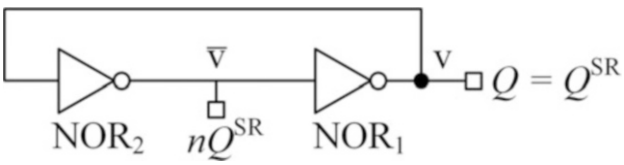

Fig. 4 Bistable element

1.3.2 Ring Oscillator

When EN="1", SR-Latch switches between Setting (S="1", R="0") and Resetting (S="0", R="1") modes based on the value obtained from inverter INV output as shown in Fig. 5.

In this mode, AND_1 element operates as a buffer repeating v or $\overline{v}$ values, NOR_2 element works as a constant "0" value generator, AND_2 and NOR_1 are identical to two inverters. This mode of operation is equivalent to the ring oscillator circuit with three inverters as shown in Fig. 6.

Thus, meander signal ($v \rightarrow \overline{v} \rightarrow v \rightarrow \ldots$) with unique frequency F_i appeared on the output Q. F_i is also determined by manufacturing process variations which make negative feedback loop delay unpredictable.

1.3.3 Metastability

Timing diagram in Fig. 7 shows three output values y_0, y_1, and y_2 from the output Q.

There are two possible ways how metastable state can appear on the output Q. First, initial value $y_0 \in \{v, X, \overline{v}\}$ (period of time from t_0 to t_1 as shown in Fig. 7) can be either stable zero, stable one, or a metastable state (X). In this case, metastability means the value with unknown stability, i.e., from time to time zero or one value appears on the output Q with different nonzero probability.

The second case is more complicated as it is based on SR-Latch phenomenon [18] which causes a high-frequency oscillation in addition to three values $\{v, X, \overline{v}\}$ in the first case. When both inputs S and R are fed with "1" value (forbidden state) for a short period of time and at this moment EN signal changes from "1" to "0", SR-Latch is trying to store forbidden state and generating damped

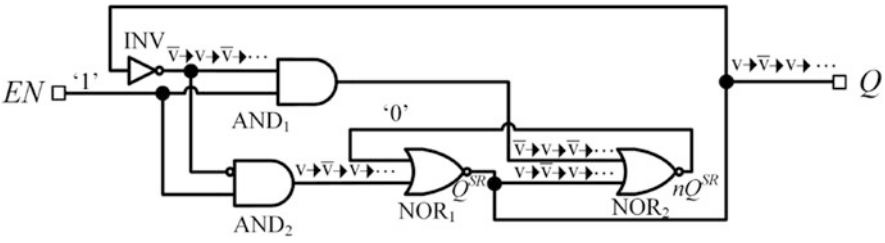

Fig. 5 ROLD circuit (EN="1")

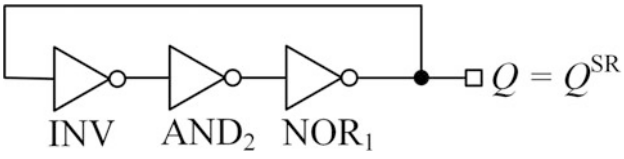

Fig. 6 Ring oscillator

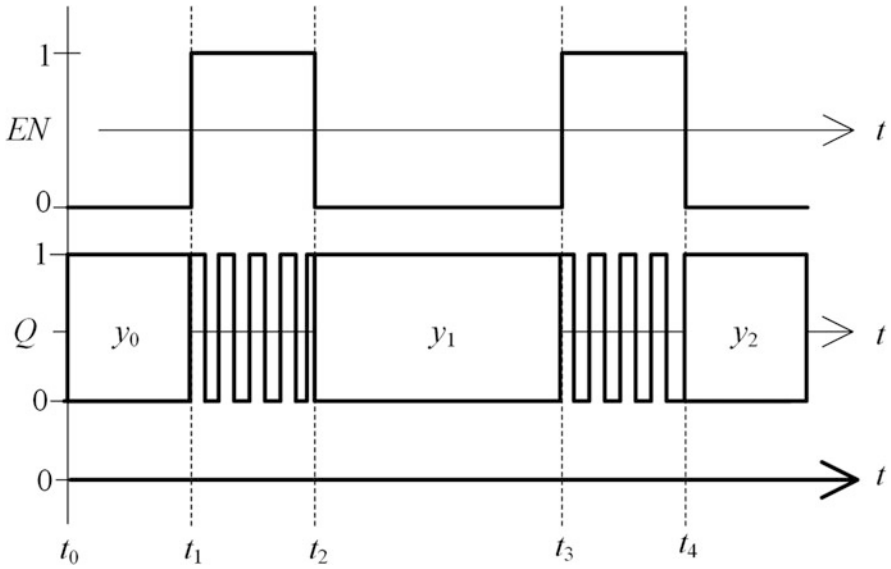

Fig. 7 SR-Latch timing diagram depending on changing EN signal

high-frequency oscillation. Metastable oscillation also dumps to the stable zero or one value after some time. So values y_1 (time period from t_2 to t_3) and y_2 (time period after t_4) will eventually get to stable zero or stable one value with or without metastable oscillation. This phenomenon is based on unique voltage and timing characteristics of SR-Latch and determined only after manufacturing.

Two mentioned scenarios of generating metastable value are shown in Fig. 8.

Possible values in the first case are shown in Fig. 8a and in second case (see Fig. 8b). Oscillation in the second case is eventually damped to the value v or $\overline{v}$, but the final value Q is more uncertain comparing to the first case.

As a result, transition of EN signal from "1" (ring oscillator mode) to "0" (latch mode) may cause high-frequency oscillation which leads to metastability state observed on the output Q. As a result, metastability can be used to generate true random numbers.

1.3.4 Latch

When EN signal is set into "0" value, it enables the possibility to store generated random values after initialization or ring oscillator mode or metastability caused oscillation. The circuit for storing N-bit unique ID (mode 1) or random number (mode 2 or 3) is shown in Fig. 9.

Thus, the proposed entropy source can be used for both purposes, unique bits producing and storing generated data.

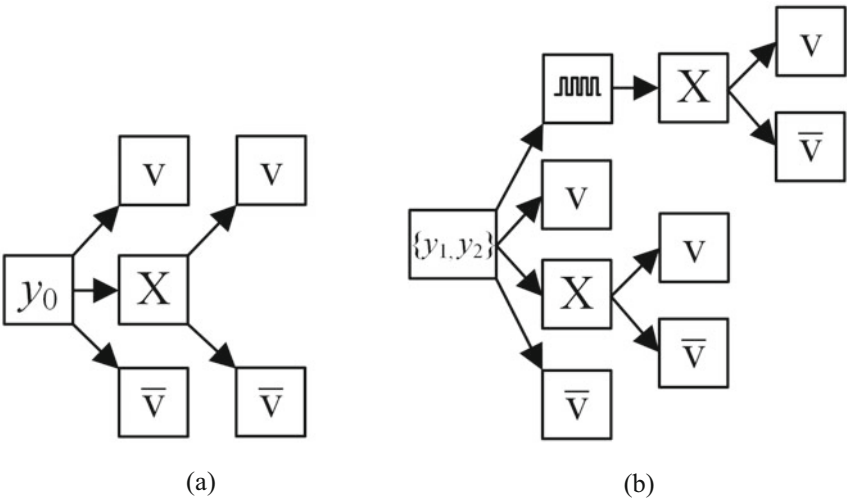

Fig. 8 Possible output values Q. (**a**) Period of time from t_0 to t_1. (**b**) Period of time from t_2 to t_3 and after t_4

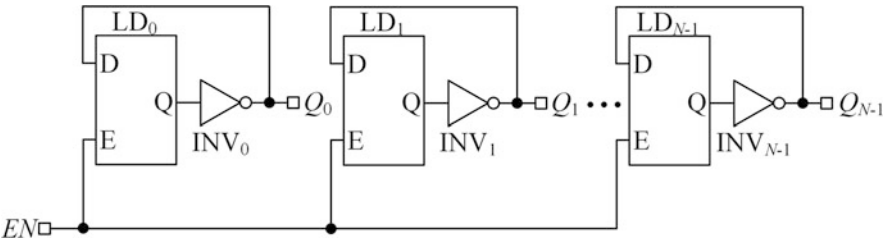

Fig. 9 Multi-bit latch for storing unique ID or random value

1.4 Experimental Results

The proposed entropy source has been implemented in Nexys 4 Xininx Artix-7 FPGA prototyping board [19], and characteristics for each mode have been collected.

1.4.1 Initial Memory

The total number of 128 entropy sources has been synthesized and implemented in FPGA. During $E = 100$ tests, each of the elements generated values shown in Fig. 10.

The distribution of probabilities of generating "1" value ($P_i^1(E)$) is the following: 61 elements with $P_i^1(E) = 0.0$, 56 elements with $P_i^1(E) = 1.0$, and 11 elements

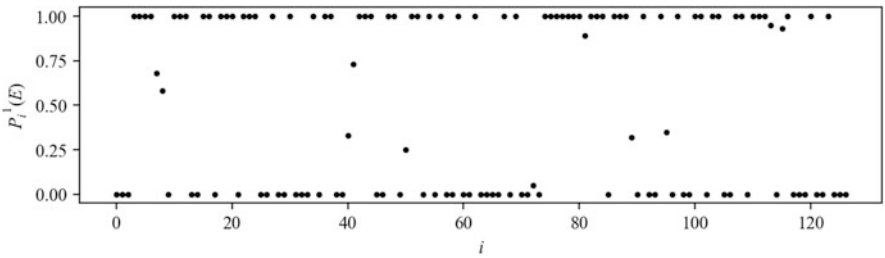

Fig. 10 Probabilities of "1" value ($P_i^1(E)$, $E = 100$) in initialization mode

with $0 < P_i^1(E) < 1$. Thus, reliable, unique, and reproducible ID can be generated using proposed method.

1.4.2 Ring Oscillator

These 128 generators have been tested in RO mode to show the uniqueness of generated frequency value F_i ($1 \leq i \leq 128$). The simulation frequency is 350 MHz (red line in Fig. 11); individual estimated frequency values F_i for entropy sources are shown in Fig. 11.

This experiment has been demonstrated that frequency value F_i for each generator is unique and unpredictable for each entropy source.

1.4.3 Metastability

Also the same 128 entropy sources have been tested $E = 100$ times in metastability mode (EN switches from "1" to "0"). The probabilities $P_i^1(E, k)$ of generating "1" value after k system clocks in RO mode ($EN =$ '1') for each element are shown in Fig. 12.

In contrast to initialization mode, the generated values have low reproducibility as all probabilities of generating "1" value ($P_i^1(E, k)$) are above 0.2 and below 0.8. Thus, this mode is more suitable for generating true random values.

1.4.4 Latch

To estimate the quality of random values produced by entropy sources, 128 elements have been utilized. As a result, a million 128-bit values have been generated by changing EN signal from "1" to "0". The duration of EN signal in "1" state is $k = 32$ system clocks. Each 128-bit value has been split into four 32-bit values. The histogram of the approximate distribution of generated four millions of 32-bit values is shown in Fig. 13.

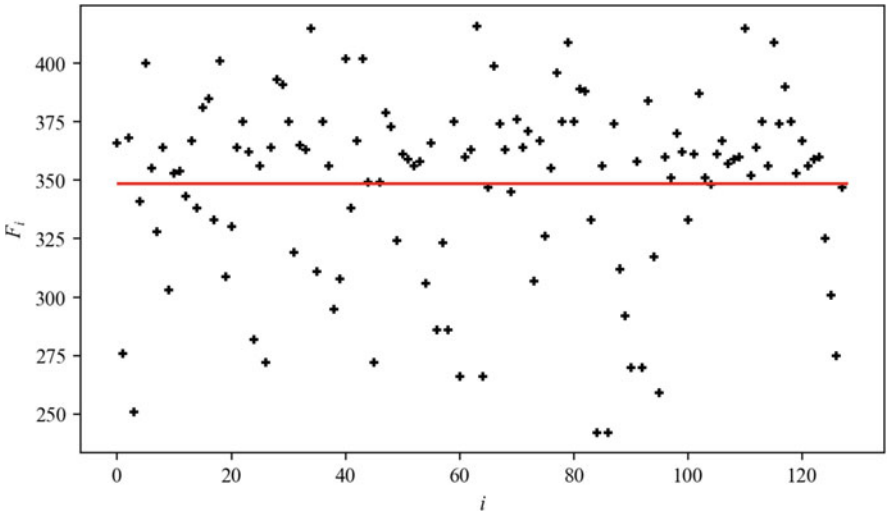

Fig. 11 Estimated frequencies F_i in RO mode

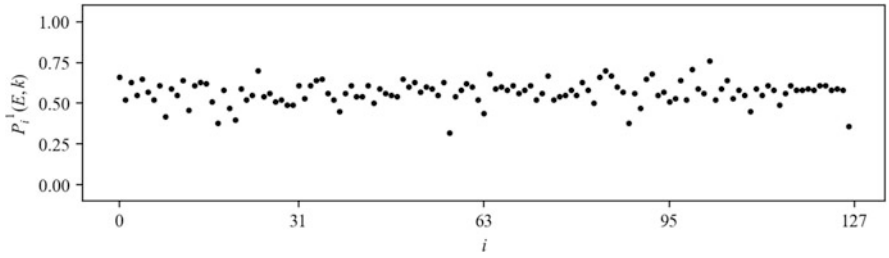

Fig. 12 Probabilities "1" value ($P_i^1(E,k)$, $E = 100$, $k = 32$) in metastability mode

The x-axis corresponds to the generated numerical value ranging from 0 to $\approx 4 \times 10^9$; the y-axis shows the estimated frequencies (the data is split into 100 bins) for each value. The generated values are truly random but not uniformly distributed. Therefore, the random sequence has to be post-processed in order to achieve required characteristics of randomness and be compliant with NIST standard [20].

1.5 Conclusion

The multimode physical unclonable function is presented. This design can be used for producing either stable unique ID or unpredictable random bit generation. The proposed design occupies smaller area comparing to classical PUF designs (e.g., Arbiter PUF, RO PUF, SRAM PUF) and can be used as an entropy source in

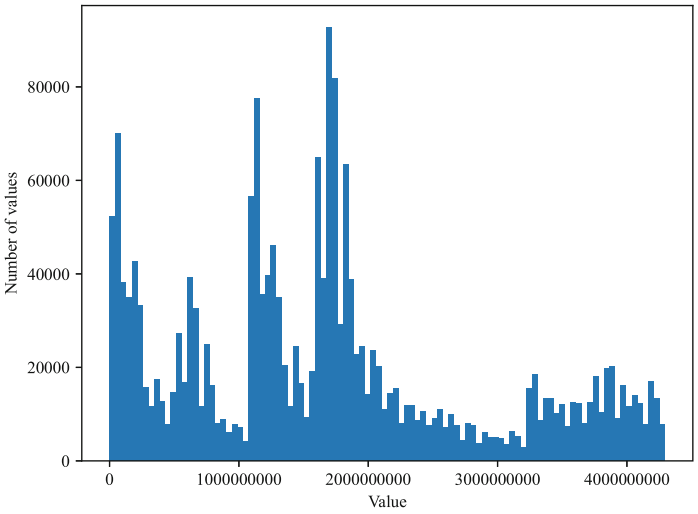

Fig. 13 Distribution of 32-bit random values ($k = 32$)

cryptography applications. For NAND flash memory devices, it can be utilized for entropy generation in encryption process and also for on-drive simulation purposes.

2 Raw Read-Based Physical Unclonable Function for TLC NAND Flash

2.1 Introduction

The increasing capacity of a single flash memory cell (SLC → MLC → TLC → QLC) has led to reliability issues with NAND-based storages [21]. This downside can be used for the opposite purpose, i.e., faults in blocks and pages can be utilized as a source of uniqueness for both chip identification and true random number generation. Modern TLC NAND flash memory devices have massive error correction code (ECC) engines which negotiate the effect of intrinsic NAND instability [22]. However, disabling ECC and scrambler modules during the read and write operation allows extracting less stable bits and using them to generate uniformly distributed random bits. As a result, one block of NAND can be separately used to generate a random number sequence during the read operation. The proposed flash memory operation is consistent with the definition of PUF, i.e., it provides a way to extract unique randomness characteristics from the physical super-high information content (SHIC) system [23]. Depending on the reliability of the obtained noise values, it can be used as random values (low reliability and high uniqueness) or unique identifiers (high reliability and high uniqueness). As a result, flash memory cells can be used

as an entropy source for TRNG which does not require additional circuitry for its implementation and random numbers can be extracted during the read operation in the raw mode. The proposed method does not require a redesign of the existing NAND flash controller and can be used directly from the firmware level.

2.2 Control of the Entropy Source

The proposed entropy source is controlled by a two-stage algorithm. The first stage is enrollment, i.e., the positions of noisy bits are located during the read operation. The second stage is generation, i.e., read noisy bits from the positions are determined during the enrollment stage.

2.2.1 Enrollment

1. Choose a block from the reserve area.
2. Erase the whole block.
3. Write all zeros pattern to the block in the raw mode, i.e., ECC and scrambler are disabled during this operation.
4. For every page p_i $(0 \leq i \leq P-1)$, perform read operation in the raw mode R times. P is the number of pages in the block.
5. Calculate noise characteristic (Ψ) for each bit b_j $(0 \leq j \leq B)$ within all P pages. If $\Psi = 0$–bit b_j is stable and if Ψ is bigger, it means that the chosen bit b_j is more random. B is the number of bits in a page.
6. Bits with highest Ψ scores should be chosen as a source of true random number sequence.

As a result, the page can be represented as shown in Fig. 29 (the heatmap shows Ψ scores for each bit b_j within page p_i).

Note: The Ψ scores should be stored offline to the array A containing $P \times B$ elements.

2.2.2 Generation

1. Determine size L of a register $\mathrm{R_{TRNG}}$ to store a random number.
2. Store the information about noisy bits from A with the highest Ψ scores to the special data structure shown in Fig. 14.

   Bit $p'_k : b'_l$ $(0 \leq k \leq K-1, 0 \leq l \leq L-1)$ corresponds to a Ψ score $A[i][j]$ of some bit b_j from page p_i. K is the number of pages chosen for random number generation.
3. Initialize index $k = 0$ for cyclic iteration.
4. Read page p'_k.

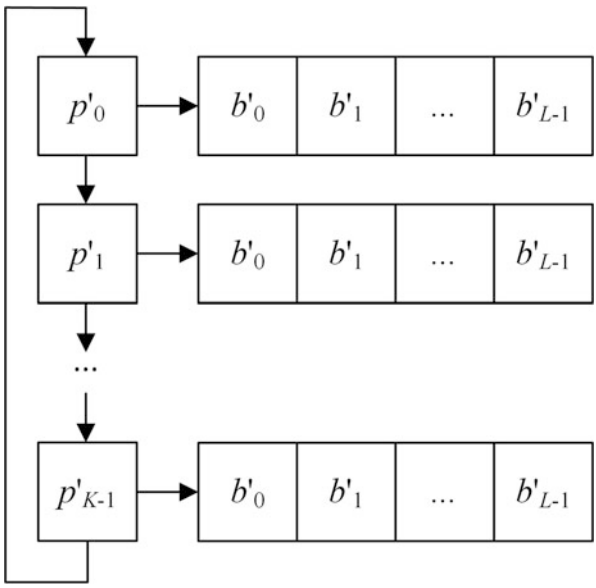

Fig. 14 The data structure for storing noisy bits for different pages within a block

5. Extract L bits $p'_k{:}b'_0 \ldots p'_k{:}b'_{L-1}$ and store them to the $\mathrm{R_{TRNG}}$ register.
6. Increment k by modulo K. Go to Step 4.

2.3 Experimental Results

SK hynix S72 512GB SSD drives have been tested in order to prove randomness of the proposed PUF design.

2.3.1 Enrollment

1–3. Block $0x84$ has been randomly chosen, erased, and written with zeros in the raw mode.
4. Read operation has been repeated in the raw mode for $R = 1000$ times.
5. For example, the randomness of each bit can be estimated as follows:

Calculate two metrics for each bit, namely, uniformity (U) and bit flipping rate (BFR):

$$U = 1 - 2 \times |\frac{R_1}{R} - 0.5| \tag{1}$$

R_1 is the number of bits with the value of "1".

For example, if there were 5 read operations and the values obtained were (1, 1, 0, 1, 0), then $U = 1–2 \times |3/5–0.5| = 1–2 \times 0.1 = 0.8$:

$$BFR = \frac{\sum_{i=0}^{B-2} b_i \oplus b_{i+1}}{B-1} \tag{2}$$

Based on U and BFR noise characteristic, Ψ can be calculated for each bit as follows:

$$\Psi = \alpha \times U + \beta \times BFR \tag{3}$$

α, β—tunable parameters which determine the importance of either uniformity or the bit flipping rate.

The example is summarized in Table 1.

Thus, increasing the importance of uniqueness sequence (0, 0, 0, 1, 1, 1) can be considered more random than (1, 1, 0, 1, 0, 1). However, usually, BFR is more important and correlated with uniqueness. Therefore, the third case is more realistic.

6. Array A has been computed based on the information obtained in Step 5.

   For example, a page with index $0x42$ has been chosen to demonstrate the uniqueness [24] of the noisy bit locations. Figure 30 shows the Ψ scores for the pages with index $0x42$ within block $0x84$ for different SSD samples.

2.3.2 Generation

1. R_{TRNG} size is set to $L = 32$.
2. To estimate the number of noisy bits per page, all data has been aggregated, and average Hamming distances between reads for all pages have been computed.

   The graph for the chosen block is shown in Fig. 31.

   As shown in Fig. 31, different pages have various Hamming distances (HD) between reads. The value of HD shows the number of noisy bits per page. Therefore, pages with a bigger value of HD are to be stored in the data structure.

   For example, pages $0x120$ and $0x1c5$ have the highest HD among all pages (see Fig. 31). The data structure containing these pages is shown in Fig. 15.
3. $k = 0$ ($K = 2$).
4. Read $p'_0 = 0x120$.

Table 1 Example of tuning α, β

Sequence	U	BFR	$\Psi, \alpha = 1, \beta = 1$	$\Psi, \alpha = 1, \beta = 0.1$	$\Psi, \alpha = 0.1, \beta = 1$
1 1 0 1 0 1	0.66	0.8	1.44	0.74	0.866
0 0 0 1 1 1	0.8	0.2	1.0	0.82	0.28

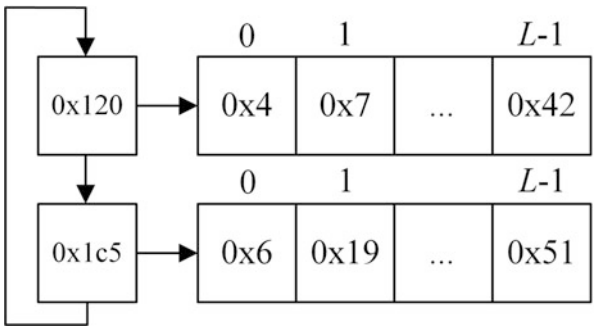

Fig. 15 Example of the data structure for noisy bits

5. $L = 32$ bits are extracted from the page p'_0 on the positions $0x4, 0x7, \ldots, 0x42$. $\mathrm{R_{TRNG}} = (1, 0, \ldots, 1)$.
6. $k = 1$.
4. Read $p'_1 = 0x1c5$.
5. $L = 32$ bits are extracted from the page p'_1 on the positions $0x6, 0x19, \ldots, 0x51$. $\mathrm{R_{TRNG}} = (1, 1, \ldots, 0)$.
6. $k = 0$.
4. Read $p'_0 = 0x120$.
5. $L = 32$ bits are extracted from the page p'_1 on the positions $0x4, 0x7$, ..., $0x42$. $\mathrm{R_{TRNG}} = (\mathbf{0}, \mathbf{1}, \ldots, 1)$.
6. $k = 0$.

The sequence of 800,000 bits has been obtained from SK hynix S72 SSD sample. The generated sequence contains 400188 zeros (50.02%) and 398812 ones (49.98%). The experiment confirmed the hypothesis of uniform distribution of noisy bits in TLC NAND.

2.4 Conclusion

The TLC NAND structure can be successfully utilized to extract uniqueness from the memory device. Existing NAND-based storage is quite unreliable for the write and read operations conducted without scrambling and ECC. Therefore, this disadvantage can be used to generate a true random number sequence. The proposed method is based on physical unclonable function (PUF) which is implemented using existing firmware functions.

The presented entropy source design has the following advantages:

- It does not require additional circuitry (hardware overhead) for its implementation.
- It cannot be reproduced on the different instance of the same device even knowing its configuration.
- It can be reconfigured using parameters L and K.

- Ψ metric can be tuned for particular requirements.
- It can generate true random numbers required for security protocols implementation using only firmware functions.

Thus, the proposed PUF-based entropy source can be utilized to enhance the security of the memory device without additional hardware cost and using only internal firmware commands.

3 Flash Memory Device Identification Based on Physical Unclonable Functions

3.1 Introduction

The memory cells of NAND flash devices have quite a low reliability, which leads to using error correction codes (ECC) with high correcting capability, e.g., BCH or LDPC code, in the data path [25]. On the other hand, excluding ECC from the data path creates a possibility of generating unique and unpredictable bits from the NAND memory cells. Thus, comparing the number of bits with one value between different pages is proposed as a source of unique and unpredictable identifiers.

The proposed ID generation method is based on the read operations, which bypass ECC and scrambling in the data path (raw read operations). The first stage (enrollment) includes erasing a block of NAND flash memory and writing an all-zero pattern to all pages within the block. Then, during multiple raw read operations, each page is characterized by an average number of ones obtained during the read operations. The second stage (uniqueness extraction) is using page statistics computed during enrollment to generate a sequence of page addresses (the number of pages is equal to the doubled ID length). Then, during the final third stage (ID generation), comparing the number of ones from the chosen pages allows generating unique ID bits, i.e., for two compared pages, if the first page has less ones than the second one during the raw read operation, zero is generated; otherwise, one is generated.

3.2 ID Generation Algorithm

A page is a minimal reading unit in the NAND flash memory, and it can be characterized by a number of bits which flip their values during the read operation. To easier highlight flipping bits, an all-zero pattern should be programmed in the page. Then, after multiple raw read operations (bypassing ECC and scrambling), the average number of ones obtained during the read operation can characterize the page. These statistics are obtained during the enrollment stage, which contains four steps:

1. Erase a block of memory.
2. Program in raw mode an all-zero pattern to all pages of a block.
3. Read in the raw mode each page N_r times.
4. Compute the average number of ones during N_r raw read operations.

For example, statistics for two blocks of memory with randomly chosen addresses 0xBE0 and 0x2F0 is shown in Fig. 16 ($N_r = 100$).

The distribution of the average number of ones in pages p_i^{avg} ($1 \leq i \leq N_p$, N_p– the number of pages in a block of memory) is unique for every block in the device. Therefore, the subtle intrinsic difference in this distribution can be utilized to design a NAND flash memory-based physical unclonable function. The block diagram for a proposed PUF design for ID generation is shown in Fig. 17, which has a similar principle as RO-PUF [17]. Instead of frequency comparison, the proposed algorithm compares the number of ones during raw read operations.

To generate a single response bit R, it is required to compare the number of ones during the raw read operation from two different pages p_i and p_j ($i \neq j$, $1 \leq i, j \leq N_p$), which are chosen based on challenge value $C = (i, j)$. C is an ordered pair of page addresses i and j which takes one of possible $\binom{N_P}{2}$ values. If $p_i < p_j$, $R = 0$; otherwise, $R = 1$. This PUF is able to generate $\binom{N_P}{2}$ possible response bits based on challenge value C. To generate an L-bit ID, identification server has to generate L challenges ($2L$ page addresses) and send them to the device. As a result, the device produces L response bits, which uniquely identify it.

Due to intrinsic NAND instability, values p_i and p_j may have different values from one read operation to another. This leads to instability of generated response

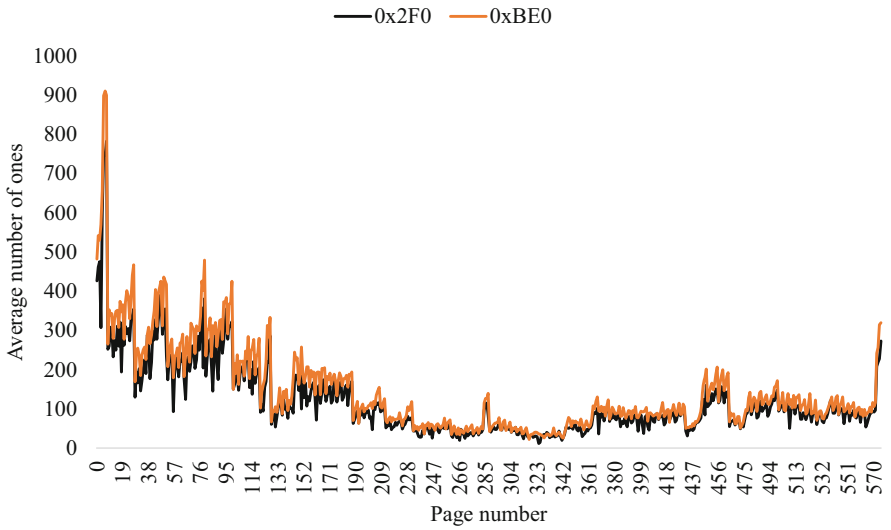

Fig. 16 The average number of ones obtained during raw read operations for two blocks of memory

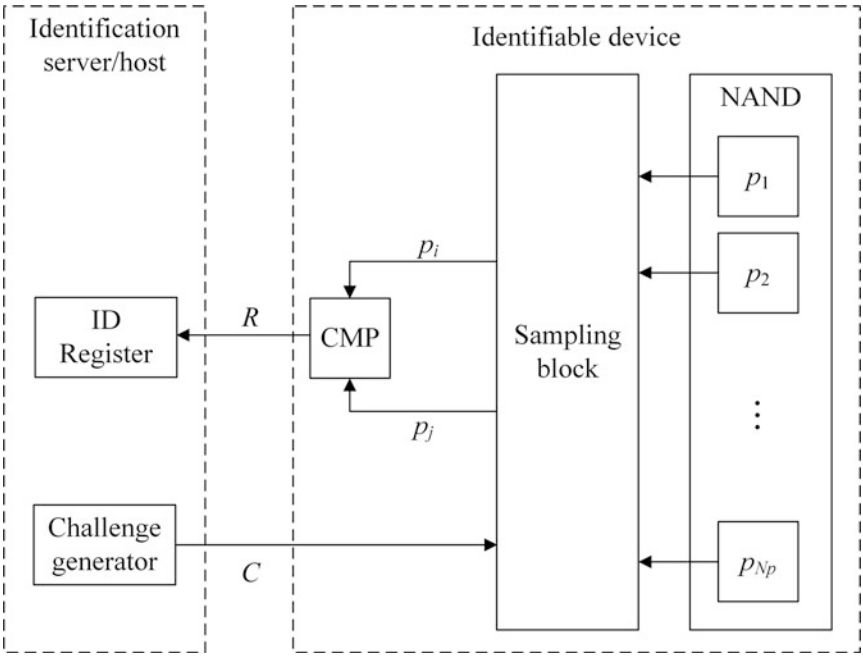

Fig. 17 ID generation based on NAND PUF

values R as during different read operations for the same address values i and j, the order of values p_i and p_j can be also different ($p_i < p_j$ or $p_i > p_j$). Thus, to provide a reliable identification, the subset of challenge values, which provide stable responses, has to be found.

If the average number of ones values obtained during the enrollment stage (see Fig. 16) are sorted, they can be separated into two groups, with lower and higher values of p_i^{avg}. Sorted values are shown in Fig. 18.

As shown in Fig. 18, the higher the difference between the average number of ones obtained for two pages p_i^{avg} and p_j^{avg} (e.g., $p_i^{\text{avg}} > p_j^{\text{avg}}$, $i \neq j$), the higher the probability to keep the order between the number of ones obtained during an arbitrary read operation ($p_i > p_j$). It also can be confirmed based on experimental data obtained from block 0x2F0. The data of 10-th and 100-th reads together with average values is shown in Fig. 19.

The value of the difference between two pages p_i (taken from pages with higher p_i^{avg} values) and p_j (taken from pages with lower p_i^{avg} values) ($p_i - p_j$) may change its value, but sign value ($p_i - p_j$) will be the same with a high probability for all read operations from 1 to at least 100. Therefore, to generate L-bit identifier, L challenges $C_k = (i, j)(1 \leq k \leq L)$ should be chosen based on enrollment data. There are multiple ways of doing this. For example, it can be done as shown in Fig. 32 in four steps:

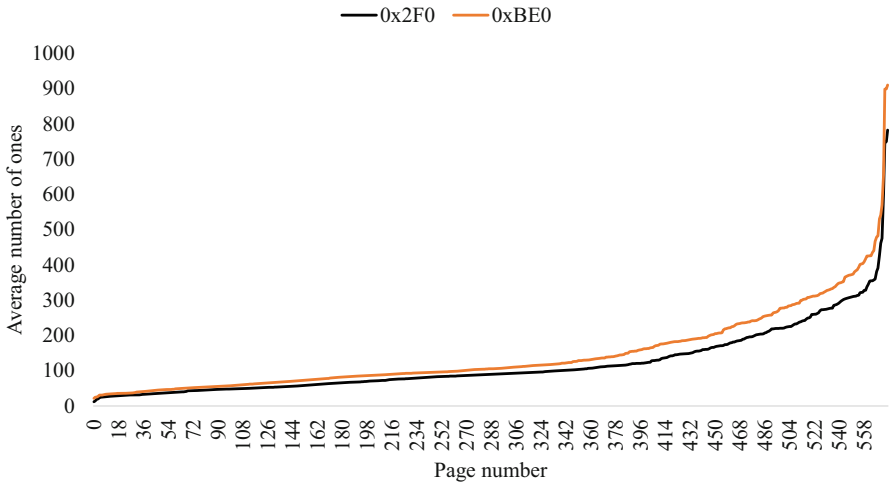

Fig. 18 The average number of ones (sorted) obtained during raw read operations for a block of memory

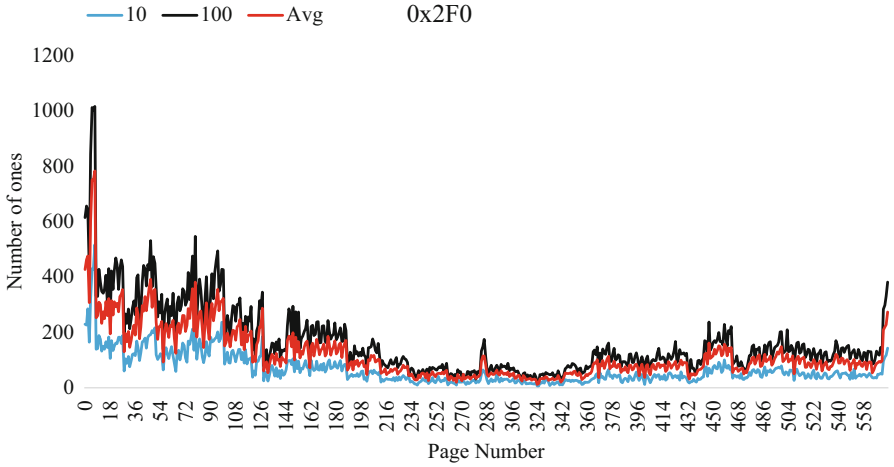

Fig. 19 The number of ones obtained during 10-th, 100-th raw read operations, and the average value

1. Sort pages by the average number of ones values obtained during raw read operations (p_i^{avg}) in ascending order. As a result, the sequence of page addresses corresponding to the sorted values can be represented as $A_1, A_2, \ldots, A_{N_p}$;
2. Split sequence into two L-element subsequences, namely, $A_{\text{low}} = (A_1, A_2, \ldots, A_L)$ with a lower value of p_i^{avg} and $A_{\text{high}} = (A_{N_p-L+1}, A_{N_p-L+2}, \ldots, A_{N_p})$ with a higher value of p_i^{avg};

3. To generate k-th bit of identifier, form an unordered pair of addresses $\{A_k, A_{N_p-L+1+k}\}$ $(1 \leq k \leq L < N_p)$, A_k is in A_{low} and $A_{N_p-L+1+k}$ is in A_{high}. If A_k and $A_{N_p-L+1+k}$ are chosen from A_{low} and A_{high} correspondingly, there is a high probability that $p_{A_k} < p_{A_{N_p-L+1+k}}$. Therefore, the unordered pair should be converted to the ordered pair (challenge value C_k) by some unique characteristics.
4. Each unordered pair $\{A_k, A_{N_p-L+1+k}\}$ can be converted to the challenge value $C_k = (A_k, A_{N_p-L+1+k})$ or $C_k = (A_{N_p-L+1+k}, A_k)$. This can be done based on the unique sequences of addresses in A_{low}:
   (a) Consider k-th element of $A_{\text{low}}(A_k)$ and the next one (A_{k+1}).
   (b) If $A_k < A_{k+1}$, unordered pair $\{A_k, A_{N_p-L+1+k}\}$ is converted to $C_k = (A_k, A_{N_p-L+1+k})$.
   (c) Otherwise, unordered pair $\{A_k, A_{N_p-L+1+k}\}$ is converted to $C_k = (A_{N_p-L+1+k}, A_k)$.
   (d) If $k = L$, A_{L+1} element is taken from a full sequence of sorted values.

The algorithm above is given for an exemplary purpose and can be changed to other ones in order to choose the most stable responses.

The final stage (ID generation) is to perform a raw read operation L times from two pages each time. To generate k-th bit, values p_{A_k} and $p_{A_{N_p-L+1+k}}$ are compared. If the pair of addresses is $(A_k, A_{N_p-L+1+k})$ in most cases, 0 value will be generated. If the pair of addresses is $(A_{N_p-L+1+k}, A_k)$ in the most cases, 1 value will be generated.

As a result, L-bit identifier can be generated using $2L$ raw read operations. The set of challenges $C_k = (A_k, A_{N_p-L+1+k})$ or $C_k = (A_{N_p-L+1+k}, A_k)$ can be either stored in the device memory for better reliability or generated by choosing L pairs from possible $\binom{N_P}{2}$ options.

3.3 Example of ID Generation

The results of the enrollment stage are shown in Fig. 16 for block 0x2F0. The uniqueness extraction stage is completed as follows:

1. The list of page addresses sorted by p_i^{avg} values is formed as follows:
   324, 325, 266, ..., 1, 5, 7 (576 addresses in total);
2. To generate $L = 128$ bit identifier, the sequence can be split into two groups:
   $A_{\text{low}} = (A_1, A_2, A_3, \ldots, A_{126}, A_{127}, A_{128}) = (324, 325, 266, \ldots, 254, 301, 242)$—128 addresses;
   $A_{\text{high}} = (A_{449}, A_{450}, A_{451}, \ldots, A_{574}, A_{575}, A_{576}) = (30, 159, 179, \ldots, 1, 5, 7)$—128 addresses;

3–4. These groups are merged into the sequence:

- The unordered pair $\{A_1, A_{449}\} = \{324, 30\}$ is converted to $C_1 = (A_1, A_{449}) = (324, 30)$ as $A_1 < A_2 (324 < 325)$;
- The unordered pair $\{A_2, A_{450}\} = \{325, 159\}$ is converted to $C_2 = (A_{450}, A_2) = (159, 325)$ as $A_2 > A_3 (325 > 266)$;
- The unordered pair $\{A_3, A_{451}\} = \{266, 179\}$ is converted to $C_3 = (A_3, A_{451}) = (266, 179)$ as $A_3 < A_4 (266 < 314)$;
- ...
- The unordered pair $\{A_{126}, A_{574}\} = \{254, 1\}$ is converted to $C_{126} = (A_{126}, A_{574}) = (254, 1)$ as $A_{126} < A_{127} (254 < 301)$;
- The unordered pair $\{A_{127}, A_{575}\} = \{301, 5\}$ is converted to $C_{127} = (A_{575}, A_{127}) = (5, 301)$ as $A_{127} > A_{128} (301 > 242)$;
- The unordered pair $\{A_{128}, A_{576}\} = \{242, 7\}$ is converted to $C_{128} = (A_{128}, A_{576}) = (242, 7)$ as $A_{128} < A_{129} (242 < 110)$.

ID generation stage is based on the sequence generated during the second stage:

- $\mathrm{ID}_1 = 0$ as $p_{324} < p_{30}$;
- $\mathrm{ID}_2 = 1$ as $p_{159} > p_{325}$;
- $\mathrm{ID}_3 = 0$ as $p_{266} < p_{179}$;
- ...
- $\mathrm{ID}_{126} = 0$ as $p_{254} < p_1$;
- $\mathrm{ID}_{127} = 1$ as $p_5 > p_{301}$;
- $\mathrm{ID}_{128} = 0$ as $p_{242} < p_7$.

3.4 Experimental Results

3.4.1 Reliability

The 128-bit IDs were generated from two different samples (10 blocks each with the same addresses)—total 20 IDs.

Reliability shows how stable is generated ID during T tests (repeated generations) [26]. It can be computed as follows (HD, Hamming distance; ID_t, ID generated during t-th test):

$$R = 1 - \mathrm{BER} = 1 - \frac{1}{T} \sum_{t=1}^{T} \mathrm{HD}(\mathrm{ID}, \mathrm{ID}_t) \tag{4}$$

The ideal value of reliability is 1.0, i.e., that generated ID is stable and does not change its value during repeated generations.

All IDs generated in the experiment have $R = 1.0$ except three of them which have 0.980, 0.989, and 0.990.

3.4.2 Uniqueness

Uniqueness shows the difference between IDs generated from different samples (inter-die uniqueness) or different blocks within the same sample (intra-die uniqueness) [26]. The ideal value of uniqueness is 0.5 which is the biggest distance from both 0 (no difference) and 1 (each bit of the vector is flipped).

Intra-die uniqueness for m IDs can be computed as follows:

$$U_{\text{intra}} = \frac{2}{m(m-1)} \sum_{u=1}^{m-1} \sum_{v=u+1}^{m} \text{HD}(\text{ID}_u, \text{ID}_v) \tag{5}$$

For $m = 10$ IDs (each sample) $U_{\text{intra}} = 0.502$ for sample 1 and $U_{\text{intra}} = 0.498$ for sample 2.

Inter-die uniqueness for m IDs situated at the same address in different two samples can be computed as follows:

$$U_{\text{inter}} = \frac{1}{m} \sum_{i=1}^{m} \text{HD}(\text{ID}_i^1, \text{ID}_i^2) \tag{6}$$

$U_{\text{inter}} = 0.518$ for two identical samples ($m = 10$ for each sample).

Also, the algorithm has been stress tested by 10,000 erases. The ID was generated after each erase for five times. Therefore, 50,000 IDs were generated during the test. Only 16 of them had single bit flip, and the rest 49,984 were the same (without bit flips).

Thus, the proposed algorithm can be used to generate a unique, reliable, unpredictable, and unclonable ID for flash memory devices.

3.5 Conclusion

This section describes the method of generating stable unique ID based on NAND flash memory. Produced IDs have high reliability (0.99) and uniqueness (0.502) and also survived after erase stress testing without losing their characteristics. The proposed method does not require additional hardware overhead in devices having onboard flash memory. One block of memory provides more than 500 unique IDs.

4 Design of Data Scrambler with Enhanced Physical Security

4.1 Introduction

Modern NAND flash memory devices [47] usually contain three parts, namely, host, controller, and NAND memory cell array as shown in Fig. 20. The host usually communicates with the device using high-speed interface and generates workload

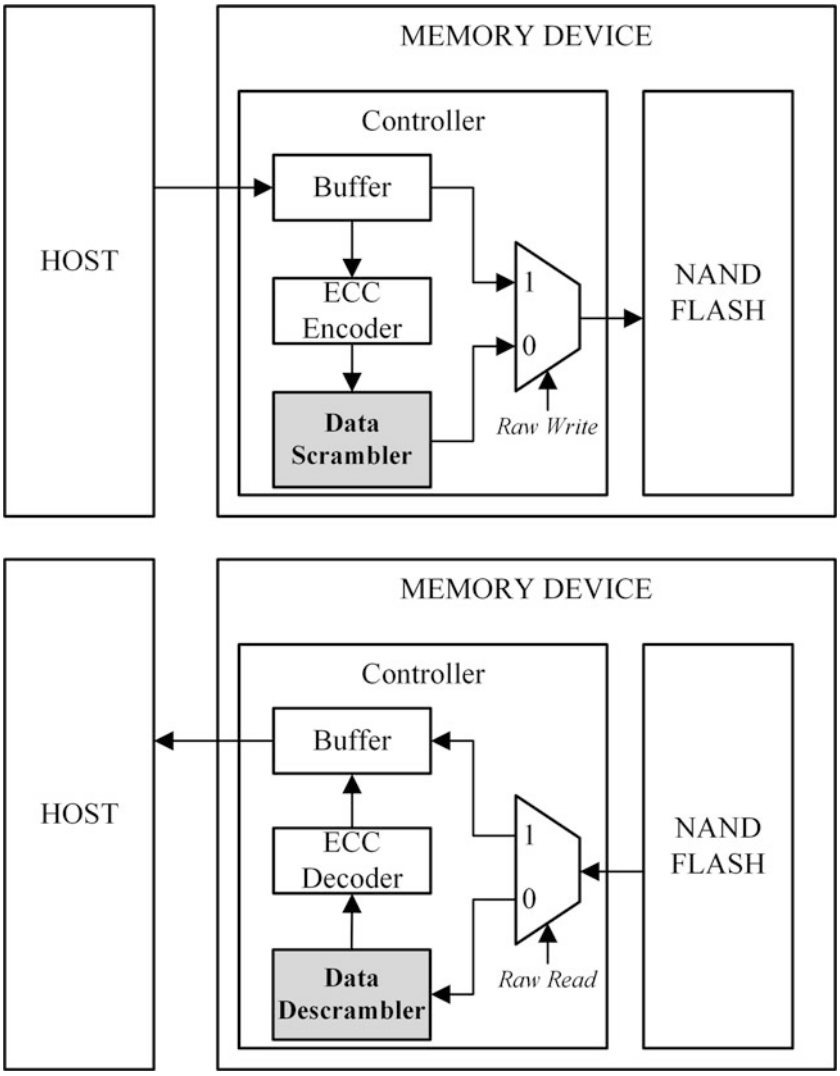

Fig. 20 Block diagram of a NAND flash memory device

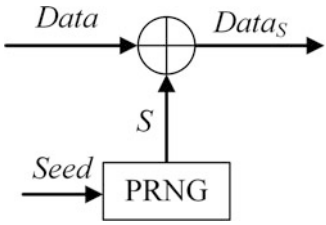

Fig. 21 Typical design of a data scrambler

for the controller. The data from the host is stored in buffer (usually DRAM) and then encoded by error correction codes (ECC). The encoding is required because basic reliability of NAND memory cells is quite low and this kind of memory introduces multiple errors during read and write operations [27].

One of the important blocks in NAND flash memory device is a hardware implementation of a scrambler (randomizer) which improves the reliability of memory cells (see Fig. 20). This can be achieved by transforming data patterns sent from the host to the uniformly distributed data [28]. The typical block structure of a data scrambler is shown in Fig. 21.

The scrambler usually contains a pseudorandom number generator (PRNG) block which is usually seeded by some value (e.g., logical block address (LBA) or a physical page number (PPN)). This block generates a uniformly distributed sequence S which is XORed with $data$ sent from the host. As a result, $Data_S = Data$ XOR S is programmed to NAND memory cells.

This way of data scrambling has a vulnerability which gives an attacker a chance to degrade the reliability of NAND memory cells [29]. Since scrambler processes the data using a pseudorandom number sequence, the attacker can collect enough outputs ($Data_S$) and restore the configuration data of the PRNG block (e.g., polynomial coefficients) [30]. Then, the attacker is able to build a mathematical model of a scrambler and obtain output values for any input data patterns.

For example, if an attacker wants to program some particular data pattern (D_p) to NAND, he/she processes this sequence using the mathematical model to obtain $D_x = (D_p$ XOR $S)$ value and sends the output D_x to the device. As a result, $Data_S = (D_p$ XOR $S)$ XOR $S = D_p$ will be programmed to the memory device. Thus, the attacker is able to get any data patterns (worst-case data patterns, e.g., all zeros) in order to degrade NAND reliability. Since many memory devices are manufactured with the same circuit design, the attacker can take advantage of using the same mathematical model of a scrambler (obtained from a single device) to degrade reliability of other devices.

The reliability of the NAND memory cells can be also degraded using the same data pattern programming [29]. For example, if same data pattern ($Data$) is sent from the host to the same LBA or PPN (the same seed value for PRNG) multiple times, it is transformed to the same data pattern on NAND ($Data_S$). As a result, memory cells are programmed with the same value, and this leads to increasing of bit error rate (BER).

In this chapter, a modified design of the data scrambler is proposed. The use of physical unclonable functions (PUFs) [2] as an additional data processing before scrambling provides a way to:

1. Significantly decrease vulnerability to building a mathematical model of a scrambler.
2. Encrypt the data without hardware costly algorithms (e.g., AES), which are not used in mobile flash and IoT (Internet of Things) devices [49].
3. Increase the reliability of the NAND by avoiding programming the same data patterns [29].
4. The use of PUF as an additional block for scrambler data encryption provides additional security against cold-boot attacks [31] as PUF response for the same challenge changes its value after each restart.

The proposed design of a data scrambler is based on adding PUF circuit to the data path of a flash memory device. This provides enhanced security to the existing scrambler design as it encrypts the data using unique PUF-generated key. It also requires much smaller hardware overhead comparing to the classical encryption algorithms (e.g., AES). Since PUF adds unique signature to the data, it becomes much harder for an attacker to mathematically model scrambler and send worst-case data patterns, which degrade the reliability of NAND memory cells. Furthermore, even if the attacker managed to know the configuration of a PRNG block for a single device, it does not give him/her the advantage for the other devices as PUF responses are unique for every device. The presented solution has two possible options of implementing the PUF:

1. Implementation of a PUF remains noisy which does not require hardware for stabilization. However, NAND ECC engine has to be strengthened in order to provide correction capability for errors brought by both NAND memory cells and PUF response.
2. Design two separate ECC engines, a stronger one for NAND errors and a weaker one for correcting errors added to data by PUF. According to experimental data, the first option requires more hardware for implementation because it utilizes NAND ECC engine with bigger correction capability.

4.2 Proposed Scrambler Circuit Operation

The usual data path of NAND flash memory device consists of ECC encoder (decoder) and scrambler, which can be placed in different order (ECC before scrambler and vice versa) depending on the design of a memory controller. Without loss of generality, consider block design of a data path shown in Fig. 22.

In this case, ECC encoder is located before scrambler. Also PUF component is added to the data path in order to provide lightweight encryption for user data. PUF block is seeded by the same value as scrambler and generates a signature R which

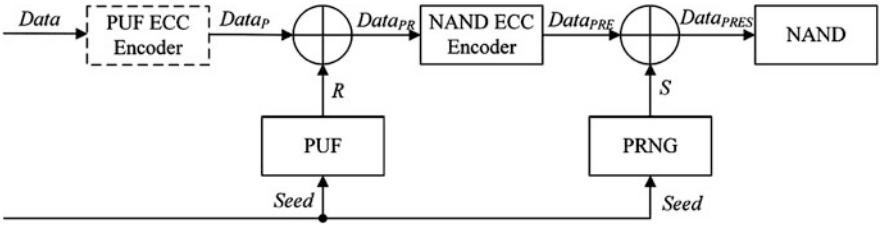

Fig. 22 Block diagram of the write data path including proposed scrambler design

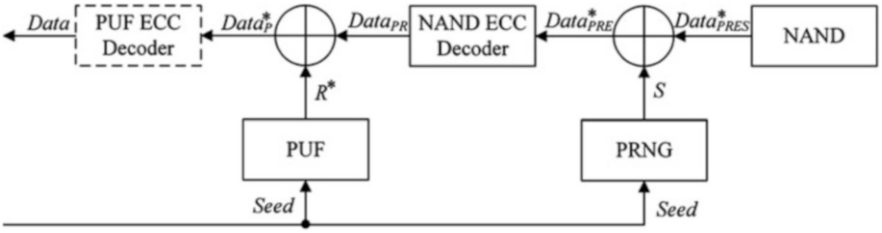

Fig. 23 Block diagram of the read data path including proposed scrambler design

is unique for every memory device even if it has completely same design. Since PUF output R can be noisy, optional small ECC engine (PUF ECC encoder) is added to the design. This engine encodes host $Data$ and converts it to $Data_P$. The PUF output R is XORed with encoded $Data_P$; encrypted data $Data_{PR}$ is further processed by NAND ECC encoder block in order to get a code word $Data_{PRE}$. The encoded and encrypted data is scrambled in a standard way by XORing with PRNG-generated value S as shown in Fig. 22. As a result, scrambled data $Data_{PRES}$ is to be programmed to the NAND. PUF ECC encoder is optional block which is used to protect data from PUF errors without modification of NAND ECC engine.

Decoding process is similar to the previously shown encoding (see Fig. 22) but performed in the opposite order. The decoding scheme is shown in Fig. 23.

First, scrambled data $Data^*_{PRES} \neq Data_{PRES}$ is read with errors as NAND-based storage usually produces multiple errors during read operation. Then, $Data^*_{PRES}$ is descrambled using value S generated by PRNG as $Data^*_{PRE}$. NAND ECC decoder corrects errors in $Data^*_{PRE}$ and produces $Data_{PR}$. Then, $Data_{PR}$ is decrypted to $Data^*_P$ using $R^* \neq R$ value produced by PUF block. As a result, $Data^*_P$ will be corrupted by noise from PUF which is basically not stable. Therefore, data sent to host is to be corrected by PUF ECC decoder as $Data$. In case of omitting PUF ECC decoder block, NAND ECC Decoder should be placed after XORing with PUF response as it has to correct both PUF and NAND noise.

Since the basic structure of the PUF can add errors to the data during decoding stage, the capability of ECC should be enlarged. This can be done using two techniques:

1. Enlarge the correcting capability of the NAND ECC engine.

2. Correct data after PUF using additional small ECC engine (PUF ECC decoder) [32] or enhancing PUF reliability [2].

Both techniques require additional hardware overhead for correcting unstable PUF outputs. This overhead is smaller than utilization of cryptographic algorithms (e.g., AES). The proposed design also decreases vulnerability to the same pattern programming [29] (because PUF response R is not stable) and to changing data pattern for every write operation. However, the presented implementation of the scrambler is still vulnerable to machine learning modeling attacks. For example, this issue can be addressed by adding obfuscating techniques to the challenge generator [33].

4.3 Experimental Results

Assume that NAND ECC engine can be implemented as BCH code, additional PUF ECC as Reed-Solomon code [34], and hardware overhead is estimated as FPGA LUT and flip-flop units. Host transmits 1023 bits of data and PUF also generates 1023-bit response:

1. PUF is noisy, and BER (bit error rate) is 0.01, i.e., that PUF generates around 11 errors in 1023-bit response.
2. NAND produces maximum 70 errors, and this can be corrected with BCH [$n = 1023, k = 323, t = 70$] code.
3. NAND ECC overhead for this implementation consists of 5441 flip-flop and 17413 LUT blocks (Xilinx Artix-7 FPGA [35]).

4.3.1 Option 1

Since PUF response should not be corrected, NAND ECC correction capability should be increased to $t = 81 = 70 + 11$. As a result, BCH [$n = 1023, k = 213, t = 81$] is to be implemented instead. Final hardware overhead for new ECC engine is 6512 flip-flop and 20840 LUT blocks. Therefore, additional hardware cost for PUF correction will be around 19.7%. However, the proposed approach can be used to improve reliability against same data pattern issue because PUF response is unpredictable.

4.3.2 Option 2

To correct errors brought by PUF responses, smaller PUF ECC engine (e.g., Reed-Solomon [$n = 1023$, $k = 1002$, $t = 11$]) is to be implemented. Therefore, it will require additional 624 flip-flop and 672 LUT blocks, which is less than 11%

of additional hardware cost. Furthermore, this approach includes additional latency overhead for PUF noise correction.

The estimation of hardware overhead is done in one of the possible ways (FPGA). It is not restricted to other technologies of scrambler implementation (e.g., ASIC).

Real implementation of a scrambler should be a trade-off between Option 1 and Option 2 in terms of hardware overhead and performance. Thus, the decision on a final implementation can be made based on constraints of a particular NAND flash memory device. Despite additional hardware cost, security and reliability enhancements are the benefits of implementation scrambler in the proposed way.

4.4 Conclusion

This section presents a new approach to designing a scrambler in NAND flash memory devices. The proposed design enhances physical security of data stored in a flash memory device and also provides better reliability comparing to the existing approaches. Scrambling algorithm has been implemented in Xilinx Artix-7 FPGA in order to compare with existing encryption schemes as AES which is usually not used in mobile NAND flash and/or IoT devices. In terms of hardware overhead, this approach is at least three times more efficient than existing encryption engines.

5 Physical Unclonable Function-Based Error Detection Algorithm for Data Integrity

5.1 Introduction

Data is usually stored in computer memory using many different representations (e.g., binary numbers, strings, compressed formats, etc.). The attribute-value pair format can be distinguished among the existing ones as it is widely used to represent data (e.g., header, email, query; string, URL; metadata, data, database entries, Internet messages, JSON objects, etc. [36]). Due to limitations of available memory space, some of these pairs can be stored externally on another device. For example, the general scheme of transmitting attribute-value pairs to the untrusted party is shown in Fig. 24. The memory controller extracts the data and generates an attribute (X) and value (Y) pair. This pair is further encoded by error correction codes (ECC) in order to avoid data losses during transmission. As a result, the encoder generates the value of X_e and Y_e and sends it via an untrusted channel to the untrusted party which stores the pair (X_e, Y_e) until requested by the device. The data should be sent back to the device and decoded to the original attribute-value pair ($X_d = X$, $Y_d = Y$).

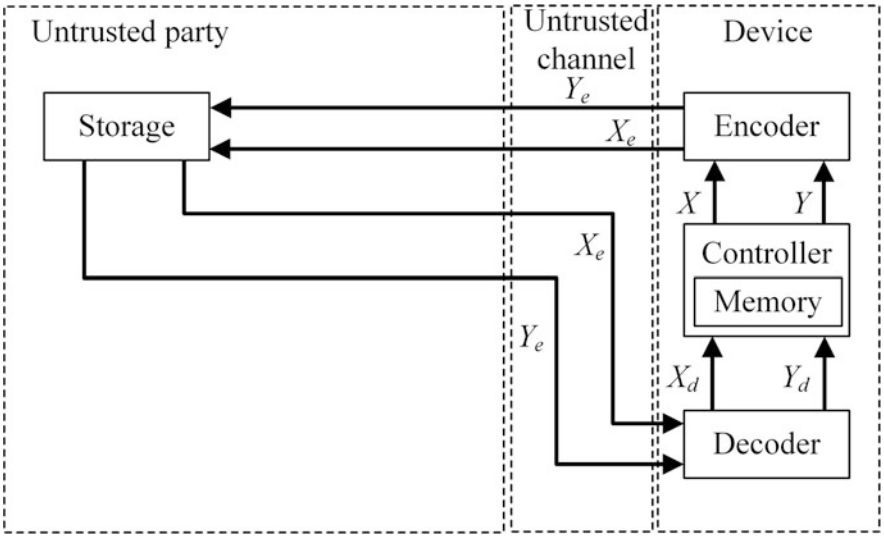

Fig. 24 General structure of data transmitting

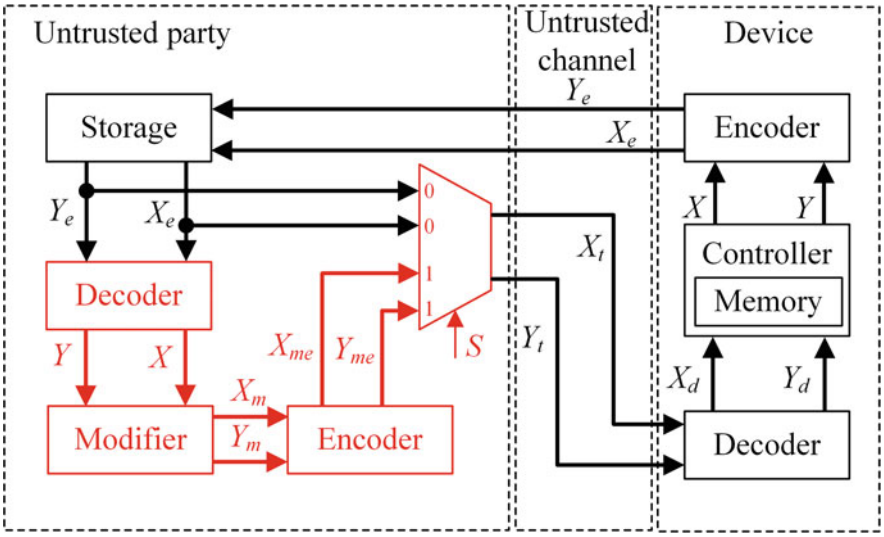

Fig. 25 Structure of untrusted party for the attack

However, since the data is stored on the untrusted party side, an attacker can observe and modify both the untrusted party and the channel [37]. Figure 25 shows one of the possible attack scenario implementations.

Since the ECC engine is used to encode the data from the device, it is possible to clone decoder and encoder blocks on the untrusted party side. Therefore, the

original attribute-value pair (X, Y) can be modified by an attacker in order to reveal the information or modify and send it back to the device to degrade performance or data integrity. The untrusted party can operate in two modes, namely, ordinary mode ($S =$ "0") when data is not modified and attack mode ($S =$ "1") when pair (X, Y) is transformed to (X_m, Y_m) and encoded to (X_{me}, Y_{me}), which is sent back to the device. Thus, if $S =$ "1", a pair (X_t, Y_t) is decoded to the $(X_d, Y_d) \neq (X, Y)$.

This way of data transmitting causes the following problems:

1. The attacker has access to the data sent via an untrusted channel as he can decode it knowing the ECC algorithm.
2. The attacker also can modify X or Y value or both values at a time in order to modify critical data on the device, degrade performance, corrupt the data transmitted, etc.
3. Encryption can prevent these problems, but it usually requires significant memory and hardware resources to be utilized as a part of the controller.

For the problem described above, physical unclonable functions (PUFs) [2] can be efficiently utilized to protect the data against unauthorized modification and prove that pair (X, Y) is generated by a particular device. PUF is a hardware security primitive which maps external input (challenge) into an output (response). This mapping is unique, unpredictable, and unclonable for the particular chip which has a PUF instance. In addition to hashing capability, PUF also extracts unique intrinsic features of an integrated circuit. This property is used for making pair (X, Y) protected against illegal access and modification.

The proposed method is based on using two PUF instances implemented on the same circuit. The first PUF is utilized to generate a hash value (R_x) for the attribute value (X) in order to use it as a key for masking linked value (Y). The encryption process can be as complicated as possible, but for simplicity and for the sake of hardware overhead reduction, the generated hash value R_x can be simply XORed with Y. The result of encryption (Y^*) is further hashed by second PUF instance, and the response of the PUF is used to check whether the unique pair (X, Y) is generated on a particular device. The PUFs utilized in this chapter should be stable (Reliability value $\approx$1.0) and strong (the number of challenge-response pairs should be exponentially large). For example, Arbiter PUF design with enhanced reliability is a good candidate for the proposed method [38]. Using PUF for data integrity is beneficial for the following reasons:

1. The attacker is not able to reproduce hash values generated by PUFs as he doesn't have access to the internals of the original device.
2. The generated response values can be used to check whether the pair (X, Y) is generated by a particular device or never existed before.
3. The proposed algorithm is more hardware-efficient than the existing encryption engines in terms of utilized chip area and power consumption.
4. Furthermore, encoding pair (X, Y) using PUF instances also allows detecting errors even if they were not injected by an attacker. So it can be also utilized instead of error detection engines.

Points 2 and 4 provide data integrity based on PUF usage for both errors brought by an attacker and errors caused by the noise in the channel and untrusted party.

5.2 Proposed Data Path Design

The proposed algorithm can be implemented by three modifications of the scheme shown in Figs. 24 and 25.

A modified encoder is shown in Fig. 26.

In order to obfuscate the value of Y and the explicit connection between X and Y, the following steps are to be completed:

1. The value of Y should be obfuscated using cryptographic salt value S produced by salt generator. The generator can be implemented as a PUF or pseudorandom number generator (PRNG). As a result, the value of Y_s is obtained as an XOR operation of Y and S ($Y_s = Y$ XOR S).
2. Obtain hash value $R_X = \mathrm{PUF}_0(X)$ (the response of PUF_0 on challenge X).
3. Encrypt the value of Y_s by XORing it with hash value R_X ($Y^* = Y_s$ XOR R_X).
4. Obtain a hash value $R_c = \mathrm{PUF}_1(Y_s)$. This value is used to prove that pair (X, Y) is generated on this device.
5. The values (X, Y^*, R_c) should be encoded by the same ECC engine as used in Figs. 24 and 25 in order to obtain values (X_e, Y_e^*, R_{ce}).
6. The values (X_e, Y_e^*, R_{ce}) are to be sent via an untrusted channel to the untrusted party. Thus, the code word is changed by adding extra hash value R_{ce}.

The enhanced encoder requires two strong and stable PUFs and one multi-input XOR gate in addition to the ECC engine previously used.

A modified decoder is shown in Fig. 27.

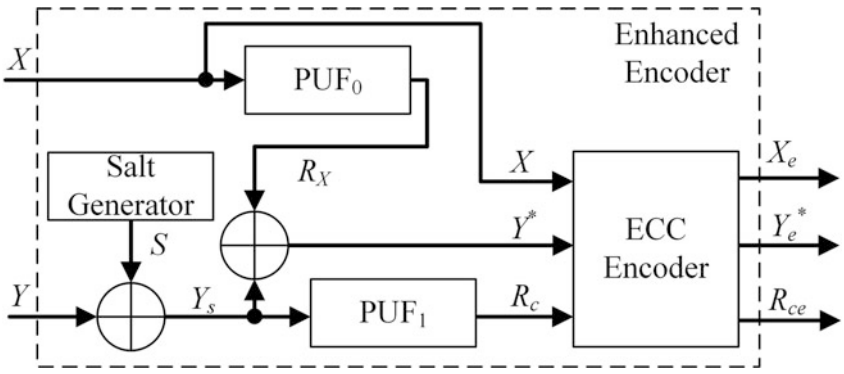

Fig. 26 Block diagram of enhanced encoder

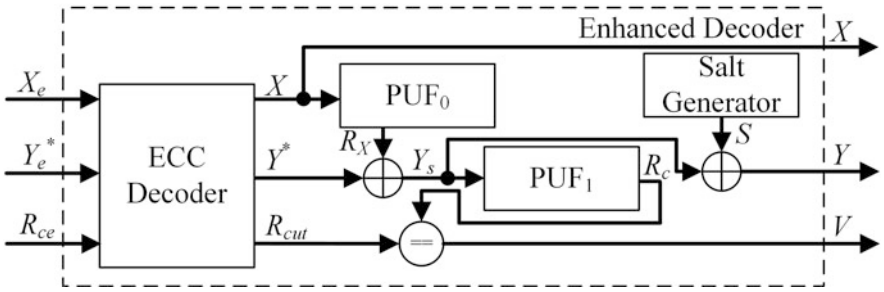

Fig. 27 Block diagram of enhanced decoder

Similarly to the encoding process, enhanced decoder utilizes two additional PUF circuits and XOR gate. To compare received hash value R_{cut} with genuine R_c value, an additional comparator is used:

1. Decode the received (X_e, Y_e^*, R_{ce}) values to get values of (X, Y^*, R_c) using the same decoding ECC engine as previously used.
2. Obtain hash value $R_X = \mathrm{PUF}_0(X)$.
3. Decrypt the value of Y_s by XORing the value of Y^* with hash value R_X ($Y_s = Y^*$ XOR R_X).
4. Deobfuscate the value of Y_s into a value of Y by XORing with salt value S.
5. Obtain hash value $R_c = \mathrm{PUF}_1(Y_s)$.
6. Compare the received hash value under test (R_{cut}) with the value of R_c. As a result, flag value V is generated ($V =$ "1" if received pair (X, Y) has been generated ($R_c = R_{cut}$) by this device and $V =$ "0" otherwise ($R_c \neq R_{cut}$)).

The code word transmitted via an untrusted channel should be transformed from (X_e, Y_e) to (X_e, Y_e^*, R_{ce}).

The changes in the communication process are shown in Fig. 28.

Modified communication protocol also includes pool of shared resources which consists of cryptographic primitives utilized by both enhanced encoder and decoder. Since PUF_0, PUF_1, and salt generator are the same, the keys are consistent for encoding and decoding processes.

As shown in Fig. 28, the attribute value (X) can be accessible by the untrusted party, because it is encoded only by the ECC engine. This does not give an advantage to the attacker as only the knowledge of the pair (X, Y) gives the possibility to observe the data stored on the device side.

Salt generator should change the value of S from time to time, e.g., based on timer (e.g., every 10 min), the number of exchanged pairs (X, Y), etc. It is used to prevent the attacker from taking advantage of functional dependency between X and Y. If X and Y do not depend on each other, this block can be omitted.

Furthermore, an attacker will not be able to modify the message as it is impossible to create a copy of PUFs to reproduce both encryption and encoding. Even if an attacker modifies the data, this fact will be detected by a decoding scheme

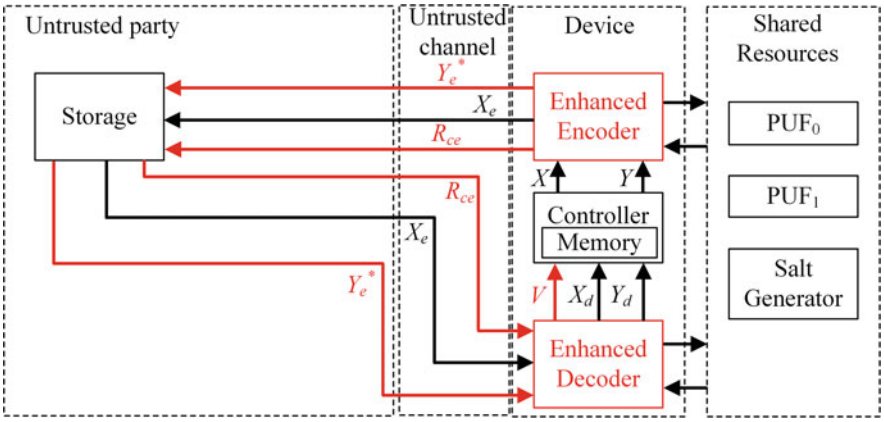

Fig. 28 Changed structure of data transmitting

based on the unique value of (Y_e^*, R_{ce}). The proposed approach also protects against errors caused by the noise on an untrusted channel and the untrusted party side.

5.3 *Example of Usage in Mobile NAND Flash Devices*

The proposed algorithm can be efficiently utilized in Host-aware Performance Booster (HPB) feature widely used in mobile flash devices [39] which is considered the same as Host Memory Buffer (HMB) used in SSD drives [40]. The block diagram of the proposed HPB algorithm enhancement is shown in Fig. 33.

Host stores HPB entries in the following format: LBA_e, PPN_e^*, R_{ce}. LBA_e is a logical block address encoded by ECC engine, i.e., it can be used by the host as a plaintext. PPN_e^* is a physical page number encrypted by enhanced encoder as shown in Fig. 26. In this case, LBA_e corresponds to X_e and PPN_e^* to Y_e^*. R_{ce} is a hash value of the PPN value.

The operation of a proposed modification of HPB algorithm can be described as follows:

1. A pair (LBA, PPN) is created by controller and stored in NAND as L2P table.
2. In order to use host memory as an external cache, NAND I/F (Interface) sends the pair (LBA, PPN) to enhanced encoder which encodes it into a triplet (LBA_e, PPN_e^*, R_{ce}) according to the encoding algorithm shown in Fig. 26.
3. If host decides to use this HPB entry (LBA_e, PPN_e^*, R_{ce}), it sends it back to the device.
4. Device controller decodes HPB entry (LBA_e, PPN_e^*, R_{ce}) into LBA, PPN_{HPB} (decoded PPN which could have been modified by host). The decoding scheme is shown in Fig. 27.

5. Enhanced decoder generates a value of V (validity of received HPB entry, $V =$ "1" when the entry is valid and $V =$ "0" otherwise). LBA is also checked in Dirty Map in order to ensure that (LBA, PPN) pair was not invalidated. Dirty Bitmap returns a validity value VD ($VD =$ "1" when pair (LBA, PPN) is not invalidated and $VD =$ "0" otherwise).
6. If both V and VD values are equal to "1", NAND I/F uses the received value and fetches the data by PPN_{HPB} address. Otherwise, it has to search the LBA and fetch the corresponding PPN from L2P table in NAND.
7. The proposed encoding and decoding algorithm provides a way to guarantee that a pair (LBA, PPN) is created by a unique NAND flash memory device as it utilizes PUF which is irreproducible by an attacker even if he knows the exact design of the encryption algorithm.

5.4 Conclusion

The encoding and decoding algorithm is proposed for attribute-value data which is transmitted via an untrusted channel. The algorithm utilizes strong and stable PUFs to prove that the attribute-value pair received from the untrusted party was generated by an authentic device. Furthermore, the algorithm is also used as an error detection method which can detect errors caused by the noise in the channel. The algorithm appends an initial code word with an additional hash value which proves the authenticity of the sent pair.

The advantages of the algorithm are listed below:

- Protection of the transmitted data from modifications by an attacker if the channel is untrusted.
- Detection of the errors caused by both noise in the channel (if ECC engine cannot correct all errors) and an attacker.
- Less additional hardware is required for the algorithm implementation compared to the encryption engines.

The proposed method can also be used as a part of HPB (HMB) algorithms in order to protect HPB entries and detect errors caused by the channel or injected by the attacker. Thus, this algorithm can be used simultaneously for error detection and security in NAND flash devices [48].

6 Conclusion

This chapter presents research results of SK hynix memory solutions Eastern Europe in area of physical security for NAND flash memory devices. Compact multimode PUF has been developed in order to be used as an entropy source with an identification feature. The proposed TRNG can be used within the existing NAND

flash controller in order to be utilized by security protocols. Randomness also has been extracted directly from NAND flash memory by using read operations without ECC protection [41]. NAND flash memory is also a source of unique identification of the device which can generate more than 500 IDs utilizing only 1 block of memory of 2 MB. Classical PUF designs have been used in order to improve key-value pair transmission in HPB and HMB protocols [42]. Also scrambling engine has been enhanced in order to provide more secure and reliable way of randomizing data before sending it to the NAND memory cells [43].

Proposed solutions show the high potential of using NAND flash memory as an entropy source for cryptography and statistical simulation applications. Also classical PUF designs improve the security and reliability of data storage and transmitting protocols. Thus, presented PUF-based security solutions can be implemented in the areas with strict security and safety requirements (e.g., medical devices [44], avionics [45], critical firmware [46], etc.).

Appendix

See Figs. 29, 30, 31, 32, and 33.

Fig. 29 Heatmap of flipping bits within a single TLC page after $R = 1000$ reads

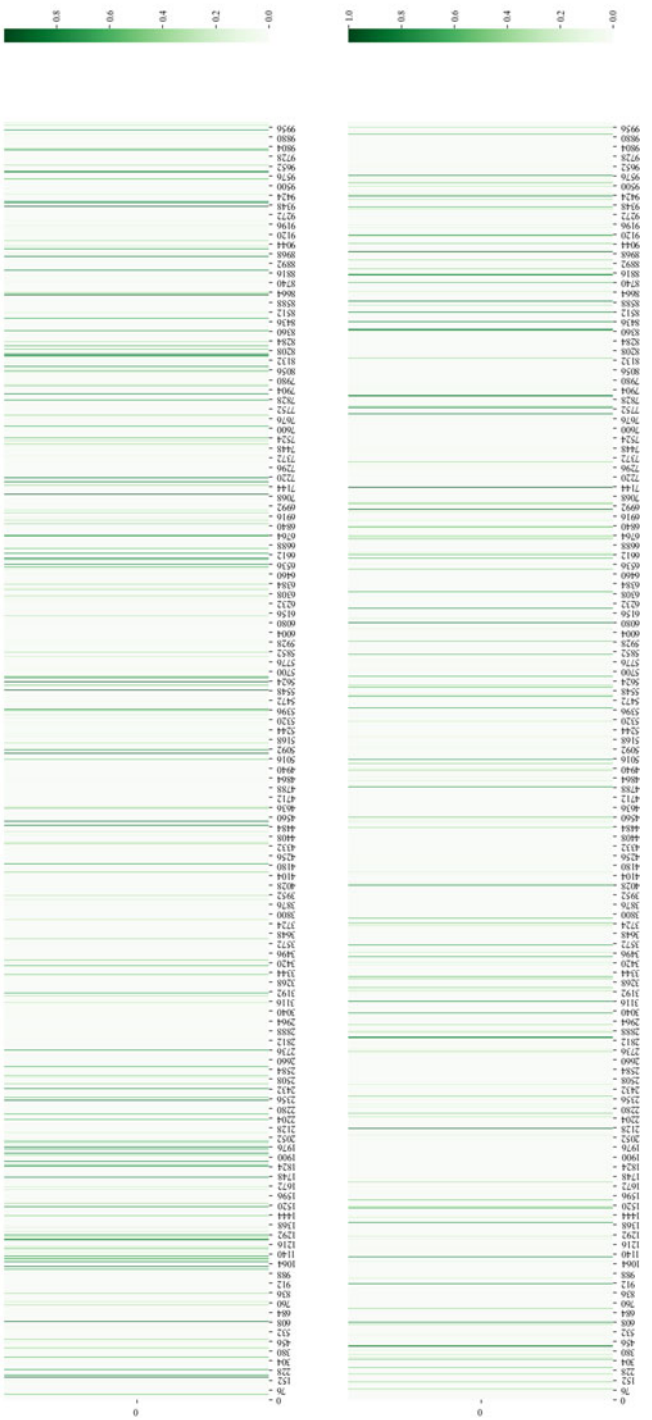

Fig. 30 Ψ scores for the pages with the same address and within the same block in different samples

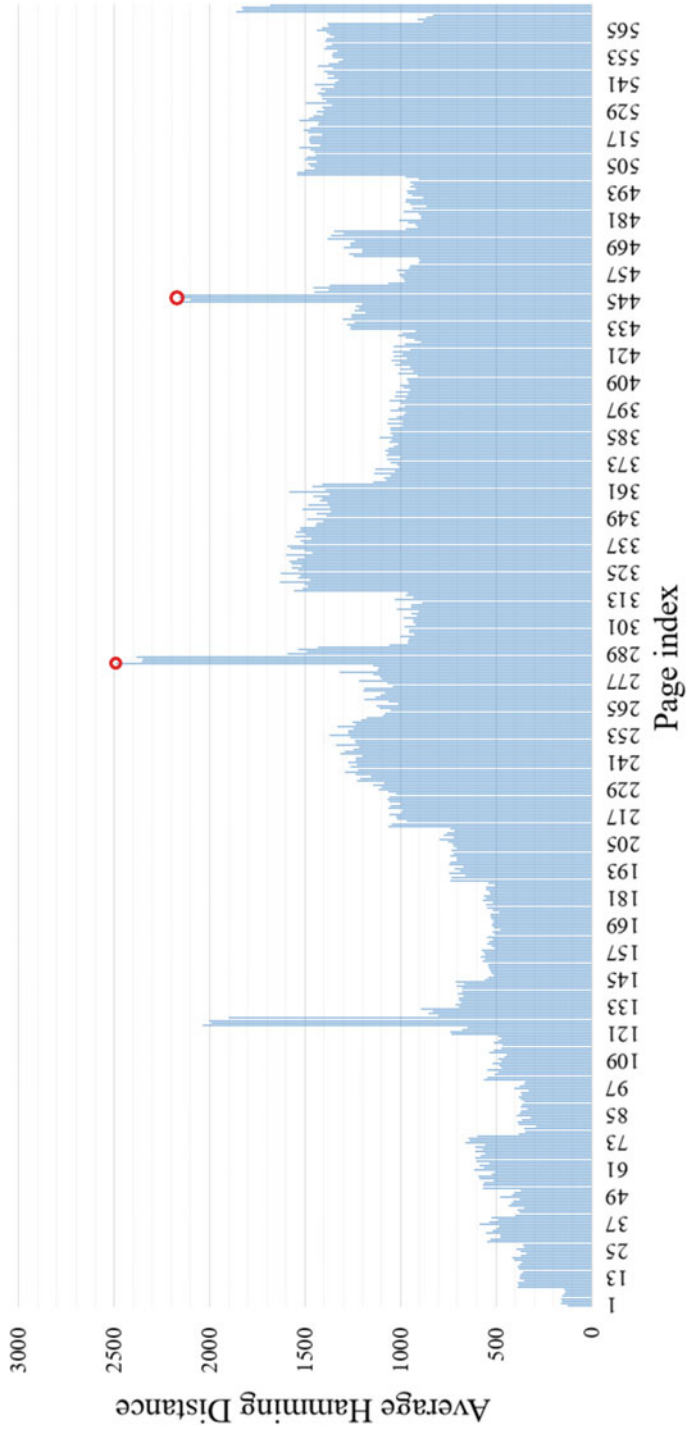

Fig. 31 Average hamming distances between reads for different pages within a block

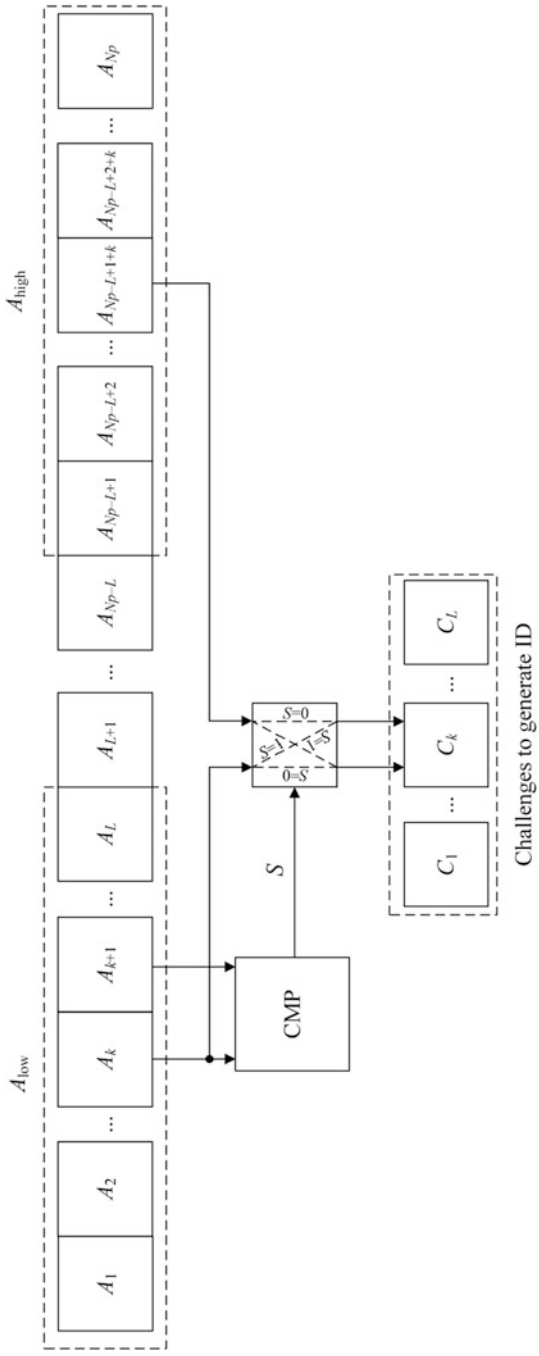

Fig. 32 Block diagram of challenges forming algorithm for ID generation

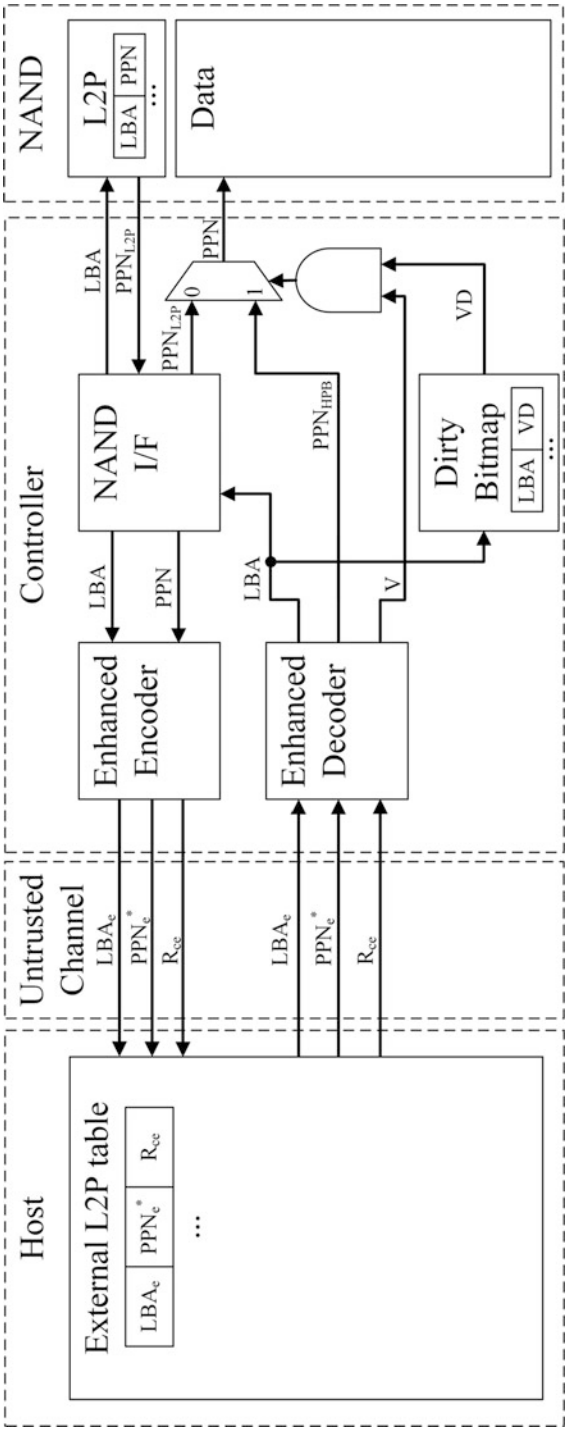

Fig. 33 Block diagram for HPB algorithm utilizing proposed encoding and decoding method

References

1. Trusted Computing Group: TCG Storage Workgroup: Storage Certification Program. In: Trusted Computing Group (2021) https://trustedcomputinggroup.org/wp-content/uploads/Storage_Certification_Program_Rev_1_33-Published-Copy.pdf. Cited 14 Oct 2021
2. Zalivaka, S.S., et al. (2016). Design and implementation of high-quality physical Unclonable functions for hardware-oriented cryptography. In Chang, C.-H., Potkonjak, M. (eds.) Secure System Design and Trustable Computing, pp. 39–81. Springer, New York
3. Samsung Introduces Exynos i T100 for Secure and Reliable IoT Devices with Short-Range Connectivity. In: Samsung (2019) . https://news.samsung.com/my/samsung-introduces-exynos-i-t100-for-secure-and-reliable-iot-devices-with-short-range-connectivity. Cited 14 Oct 2021
4. Lu, T., Kenny, R., Atsatt, S.: Secure device manager for Intel Stratix 10 devices provides FPGA and SoC security. In: Intel (2018). https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01252-secure-device-manager-for-fpga-soc-security.pdf. Cited 14 Oct 2021
5. Toshiba Develops A New PUF Technology for Solid-State Authentication of IoT Equipment. In: Istanbulpost (2018). https://www.istanbulpost.com.tr/toshiba-develops-a-new-puf-technology-for-solid-state-authentication-of-iot-equipment/. Cited 14 Oct 2021
6. Chang, C.H., Zheng, Y., Zhang, L.: A retrospective and a look forward: fifteen years of physical unclonable function advancement. IEEE Circ. and Syst. Mag. **17**(3), 32–62 (2017)
7. Lee, J., et al: A technique to build a secret key in integrated circuits for identification and authentication applications. Int. Symp. VLSI Circ. (VLSI'04), pp. 176–179 (2004)
8. Sehwag, V., Saha, T.: TV-PUF: a fast lightweight analog physical unclonable function. Int. Symp. Nanoel. Inf. Syst. (iNIS'16), pp. 182–186 (2016)
9. Gassend, B., et al: Silicon physical random functions. In: ACM Conf. Comput. and Comm. Secur. (CCS'02), pp. 148–160 (2002)
10. Cao, Y., et al.: CMOS image sensor based physical unclonable function for coherent sensor-level authentication. IEEE Trans. Circuits Syst. I Regul. Pap. **62**(11), 2629–2640 (2015)
11. Holcomb, D.E., Burleson, W.P., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Int. Conf. RFID Secur. (RFID'07), pp. 1–2 (2007)
12. Tehranipoor, F., et al: DRAM-based intrinsic physically unclonable functions for system-level security and authentication. IEEE Trans. Very Large Scale Integr. Circ. **25**(3), 1085–1097 (2017)
13. Kumar, S.S., et al: Extended abstract: the butterfly PUF protecting IP on every FPGA. In: IEEE Int. Worksh. on Hardw.-Orient. Secur. and Trust. (HOST'08), pp. 67–70 (2008)
14. Yamamoto, D., et al: Uniqueness enhancement of PUF responses based on the locations of random outputting RS latches. In: Int. Worksh. Crypt. Hardw. and Emb. Syst. 2011 (CHES'11), pp. 390–406
15. Jia, S., et al: Extracting Robust Keys from NAND Flash Physical Unclonable Functions. Int. Conf. on Inf. Secur. (ISC'15), pp. 437–454 (2015)
16. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of true random numbers. ieee T. Comp. 58(9), 1198–1210 (2009). https://doi.org/10.1109/TC.2008.212
17. Suh, G.E., Devadas, S: Physical unclonable functions for device authentication and secret key generation. In: ACM/IEEE Des. Autom. Conf. (DAC'07), pp. 9–14 (2007)
18. Kacprzak, T.: Analysis of oscillatory metastable operation of an RS flipflop. IEEE J. Solid State Cir. **23**(1), 260–266 (1988)
19. Digilent: Nexys 4 FPGA board reference manual. In: Digilent Inc. (2016). https://digilent.com/reference/_media/reference/programmable-logic/nexys-4/nexys4_rm.pdf. Cited 02 Nov 2021
20. Barker, E., Kelsey, J.: Recommendation for random bit generator (RBG) constructions. In: National Institute of Standards and Technology (NIST) (2016). https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf. Cited 14 Oct 2021

21. Papandreou, N., et al.: Open block characterization and read voltage calibration of 3D QLC NAND flash. In: IEEE Int. Rel. Phys. Symp. (IRPS'20), pp. 1–6 (2020)
22. Papandreou, N., et al.: Reliability of 3D NAND flash memory with a focus on read voltage calibration from a system aspect. IEEE Non-Vol. Mem. Tech. Symp. (NVMTS'19), pp. 1–4 (2019)
23. Ruhrmair, U., Solter, J., Sehnke, F.: On the foundations of physical unclonable functions. In: Cryptology ePrint Archive (2009). https://eprint.iacr.org/2009/277.pdf. Cited 21 Feb 2022
24. Vijayakumar, A., Patil, V.C., Kundu, S.: On testing physically unclonable functions for uniqueness. In: IEEE Int. Symp. on Qual. El. Des. (ISQED'16), pp. 244–249 (2016)
25. Cai, Y., et al.: Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. Proc. IEEE **105**(9), 1666–1704 (2017)
26. Hori, Y., et al.: Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs. In: Int. Conf. Reconf. Comp. FPGA (ReConFig'10), pp. 298–303 (2010)
27. Micheloni, R., Crippa, L., Marelli, A.: Inside NAND Flash Memories, 582 p. Springer, New York (2010)
28. Cha, Y., Kang, S.: Data randomization scheme for endurance enhancement and interference mitigation of multilevel flash memory devices. ETRI J. **35**(1), 166–169 (2013)
29. Cai, Y., et al.: Vulnerabilities in MLC NAND flash memory programming: experimental analysis, exploits, and mitigation techniques. In: IEEE Int. Symp. on High-Perf. Comp. Arch. (HPCA'17), pp. 49–60 (2017)
30. Van Zandwijk, J.P.: A mathematical approach to NAND flash-memory descrambling and decoding. Digital Invest. **12**, 41–52 (2015). https://doi.org/10.1016/j.diin.2015.01.003
31. Heninger, N.: Cold-boot attacks. In: Encyclopedia of Cryptography and Security (2011). https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_124. Cited 21 Feb 2022
32. Maes, R., Van Herrewege, A., Verbauwhede, I.: PUFKY: A fully functional PUF-based cryptographic key generator. In: Crypt. Hardw. and Emb. Syst. (CHES'12), pp. 302–319 (2012)
33. Zalivaka, S..S., Ivaniuk, A.A., Chang, C.H.: Reliable and modeling attack resistant authentication of arbiter PUF in FPGA implementation with trinary quadruple response. IEEE Trans. Inf. Forens. Secur. **14**(4), 1109–1123 (2019)
34. Tomlinson, M., et al.: Error-Correction Coding and Decoding, 522 p. Springer, New York (2017)
35. Xilinx: Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics. In: Xilinx Inc. (2021). https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf. Cited 11 Oct 2021
36. Amazon: What is a key-value database? In: Amazon Inc. (2019). https://aws.amazon.com/nosql/key-value/. Cited 13 Oct 2021
37. Blahut, R.E.: Cryptography and Secure Communication, 587 p. Cambridge University Press, Cambridge (2014)
38. Zalivaka, S..S., et al.: Multi-valued arbiters for quality enhancement of PUF responses on FPGA implementation. In: Asia and South Pacific Des. Autom. Conf. (ASP-DAC'19), pp. 533–538 (2019)
39. Jeong, W., et al.: Improving flash storage performance by caching address mapping table in host memory. In: USENIX Worksh. on Hot Topics in Stor. and File Syst. (HotStorage'17), pp. 19–24 (2017)
40. Dorgelo, J.: Host memory buffer (HMB) based SSD system. In: Proc. Flash Memory Summit (FMS'15) (2015). https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2015/20150813_FJ31_Chen_Dorgello.pdf. Cited 13 Oct 2021
41. Zalivaka, S.S., Ivaniuk, A.A.: Raw read based physically unclonable function for flash memory. US patent application (US20210055912A1). https://patents.google.com/patent/US20210055912A1 Cited 13 Oct 2021
42. Zalivaka, S.S., Ivaniuk, A.A.: Encoder and decoder using physically unclonable functions. US patent (US11394529B2). https://patents.google.com/patent/US11394529B2 Cited 05 Oct 2022

43. Zalivaka, S.S., Ivaniuk, A.A.: Data scramblers with enhanced physical security. US patent application (US20210326490A1). https://patents.google.com/patent/US20210326490A1 Cited 03 Nov 2021
44. Rodriguez, C.A.: Safeguard smart medical devices for enhanced patient safety. In: Maxim Integrated (2020). https://www.maximintegrated.com/en/design/blog/safeguard-smart-medical-devices-for-enhanced-patient-safety.html. Cited 22 Feb 2022
45. O'Neill, K., et al.: Protecting flight critical systems against security threats in commercial air transportation. In: Dig. Avion. Syst. Conf. (DASC'16), pp. 1–7 (2016)
46. Protection against Reverse-Engineering, Counterfeiting/Cloning and Overbuilding. In: Intrinsic ID (2021). https://www.intrinsic-id.com/firmware-ip-protection/. Cited 22 Feb 2022
47. McIntyre, D.: Annual flash controller update. In: Proc. Flash Memory Summit 2019 (FMS'19). https://www.flashmemorysummit.com/Proceedings2019/08-06-Tuesday/20190806_CTRL-102A-1_McIntyre.pdf. Cited 11 Oct 2021
48. Tyson, M.: Researchers find "pattern of critical issues" in SSD encryption. In: HEXUS.net (2018). https://hexus.net/tech/news/storage/123986-researchers-find-pattern-critical-issues-ssd-encryption/. Cited 11 Oct 2021
49. Pickering, P.: NAND rises to the occasion in data-heavy IoT applications. In: Electronic Design (2021). https://www.electronicdesign.com/technologies/iot/article/21807634/nand-rises-to-the-occasion-in-dataheavy-iot-applications. Cited 11 Oct 2021
50. Rajendiran, K.: Using PUFs for random number generation. In: Intrinsic ID (2021). https://semiwiki.com/ip/intrinsic-id/303704-using-pufs-for-random-number-generation/. Cited 21 Feb 2022
51. Neustadter, D.: True random number generators for heightened security in any SoC. In: Synopsys Inc (2021). https://www.synopsys.com/designware-ip/technical-bulletin/true-random-number-generator-security-2019q3.html. Cited 21 Feb 2022
52. Intrinsic ID: Zign RNG. In: Intrinsic ID (2021). https://www.intrinsic-id.com/products/zign-rng/. Cited 21 Feb 2022