Provably-secure randomness generation from switching probability of magnetic tunnel junctions

Hong Jie Ng,1,* Shuhan Yang,1,* Zhaoyang Yao,1,‡ Hyunsoo Yang,1,† and Charles C.-W. Lim1,‡
1 Department of Electrical & Computer Engineering, National University of Singapore, Singapore

In recent years, true random number generators (TRNGs) based on magnetic tunnelling junction (MTJ) have become increasingly attractive. This is because MTJ-based TRNGs offer some advantages over traditional CMOS-based TRNGs, such as smaller area and simpler structure. However, there has been no work thus far that quantified the quality of the raw output of an MTJ-based TRNG and performed suitable randomness extraction to produce provably-secure random bits, unlike their CMOS-based counterparts. In this work, we implement an MTJ-based TRNG and characterise the entropy of the raw output. Using this information, we perform post-processing to extract a set of random bits which are provably-secure.

I. INTRODUCTION

Random numbers are used in a myriad of applications, from Monte Carlo simulations for scientific research to encryption for data security. Without a doubt, random numbers are widely sought after, and much research has been done to construct random number generators (RNGs) that output high quality random numbers. The quality of random numbers can be characterised by two properties - uniformity and predictability. Indeed, in the ideal scenario, random numbers ought to be uniformly random and unpredictable from any perspective.

There are typically two classes of generators. They are the pseudo random number generator (PRNG) and true random number generator (TRNG). PRNGs make use of an algorithm, together with a short input random seed, to produce a longer sequence of random numbers that are uniformly distributed. However, PRNGs are deterministic algorithms at their core. Thus, any party with knowledge of the seed could predict the output of the PRNG perfectly. It is also possible for an observer to use the first few outputs of a PRNG to gain information about its current state, allowing the observer to accurately predict the subsequent outputs of the PRNG. RandCrack [1], a Python script that was written in 2017, only requires the first 624 numbers of the Mersenne Twister [2] PRNG to be able to predict the following output numbers with high accuracy. In fact, the Mersenne Twister PRNG is widely used in C, Matlab, and Python due to its supposedly strong generator properties.

Briefly speaking, TRNGs utilise the randomness from physical processes (e.g. time taken for atomic decay) to generate random numbers. These processes are inherently random and unpredictable. Hence, they do not exhibit the shortcomings of PRNGs. Over the years, there have been many variations of TRNGs that have been implemented [3–9]. TRNGs based on magnetic tunnelling junction (MTJ) were first proposed in 2014 [10].

* These two authors contributed equally
† eleyang@nus.edu.sg
‡ charles.lim@nus.edu.sg

FIG. 1. Illustration of STT-MTJ random number generation principle. (left) Two stable low energy states; (right) the metastable state.

Compared to their CMOS-based counterparts, some attractive features of MTJ-based TRNGs include a smaller area, a simpler structure, and scalability with silicon large-scale integration technology [11].

An MTJ device consists of two magnetic layers, which are the fixed layer and free layer. Depending on the relative magnetic alignment of the two magnetic layers, the MTJ exhibits two different stable states, namely the parallel (P) state and anti-parallel (AP) state. To switch between these states, we can take advantage of spin transfer torque (STT), which utilizes current to manipulate the direction of the magnetic moment of the free layer [12]. In a memory device, current pulses are applied to set the MTJ in one of the stable states. However, by adjusting the amplitude and width of the current pulses, the MTJ can be perturbed to the metastable state, as shown in Fig. 1. At the metastable state, thermal fluctuations determine the final state of the MTJ and give rise to randomness [10].

A system is considered as provably-secure if the system's security can be formally expressed or bounded using mathematical statements, under some general assumptions about the generator and its physics. This is in contrast to the usual heuristic analysis, which typically considers the bounded adversary model, i.e., computationally bounded adversaries. Indeed, a provably-secure framework is most desirable, especially for important systems such as encryption and random number generation,
where their security has to be lasting and independent of technological advancements. In addition, such a classification provides an operational meaning as well. In the case of RNGs, this definition implies that an adversary will be unable to distinguish the output of the RNG from a truly uniform and random data, except with a vanishingly small probability.

To the best of our knowledge, previous works that have implemented MTJ-based TRNGs did not obtain provably-secure random numbers. One of the works [13] performed entropy estimation on a 1 Mbit dataset, which is too small in size. Thus, they did not perform suitable post-processing to obtain provably secure random numbers. Other works did not characterise the entropy of the raw output [10] or provided only a partial characterisation of the raw output by assuming that the raw bits are independent [14, 15]. In this work, we implement a MTJ-based TRNG setup and characterise the min-entropy of the raw output. Our min-entropy characterisation is carried out with minimal assumptions as we do not assume that the raw output bits are independent. We then perform suitable post-processing in the form of randomness extraction to obtain a set of provably-secure random numbers.

The remaining of this paper is organised as follows. In section II, we provide the implementation details for our work, including the experimental setup of our MTJ-based TRNG and the tuning of parameters. In section III, we explain the idea behind provably-secure randomness extraction, and give a mathematical definition for the ideal output of an RNG. In section IV, we characterise the raw output of our MTJ-based TRNG and choose a suitable function to perform randomness extraction. Lastly, we present the final post-processed results in section V and give a conclusion in section VI.

II. IMPLEMENTATION

In this work, the TRNG is implemented using STT-MTJ. Fig. 2 shows the measured resistance against voltage (R-V) hysteresis curve of the fabricated MTJ device. The tunnelling magneto-resistance ratio (TMR) is \( \sim 200\% \). The writing operation of the STT-MTJ is a stochastic process, which is influenced by thermal fluctuations [16–18]. By carefully tuning the input pulses’ amplitude and width, the switching probability of MTJ can be set to 50\%. This makes it possible to implement TRNG using STT-MTJ. Moreover, because MTJ-based TRNGs can be integrated with CMOS chips in high density [19], it can be a great candidate for future true random number generation devices.

Fig. 3 shows the setup of our MTJ-based TRNG. The setup includes a pulse generator (Keysight 81134A) for generating reset and perturb pulses, a DAQ-mx (NI-USB 6363), and a R=3 kΩ resistor for reading operation. To guarantee the simultaneous start of the pulse generation and the reading operation, a trigger signal from DAQ to the pulse generator is applied. The negative reset pulse from CH1 of the pulse generator and the positive perturb pulse from CH2 are combined in the power combiner. Due to the non-volatility of MTJ devices, a reset step is required before every bit of random number generation. Thus, a random number generation cycle consists of three steps: reset, perturb, and read [19].

For the reading purpose, a DC offset voltage (\( V_{\text{offset}} \)) of 50 mV is superpositioned to the input signal. The MTJ resistance can be expressed as

\[
R_{\text{MTJ}} = \frac{V_{\text{MTJ}}}{V_{\text{offset}} - V_{\text{MTJ}}} R. \tag{1}
\]

where \( R_{\text{MTJ}} \) is the MTJ resistance, \( V_{\text{MTJ}} \) is the voltage drop across the MTJ, \( V_{\text{offset}} \) is the offset voltage applied for reading purpose, \( R \) is the resistor in series to the MTJ. Fig. 4 shows the time sequence of one random bit generation cycle. A negative reset pulse (\( V_{\text{reset}} \)) with an amplitude of -900 mV and width of 1 \( \mu \)s is first applied.
to reset the MTJ to the low resistance parallel state. After the reset pulse, a positive pulse \( V_{\text{perturb}} \) with an amplitude of 652 mV and width of 1 µs is applied to perturb the MTJ to the metastable state. One cycle of random number generation takes 8 µs. The reading operation performed by the DAQ card is conducted at 2 MHz throughout the whole process. Thus, the DAQ card can read out 16 data points for one cycle. To obtain MTJ resistance, only data points corresponding to sequence III should be used. To ensure this, it is required that the data points corresponding to reset and perturb pulses can be clearly identified. Therefore, the pulse width of reset and perturb pulses cannot be too short, otherwise it would be hard to differentiate them. This is why in our case, 1 µs pulse width is chosen. After selecting out the correct sequence of data points, we take the average, and set 4 kΩ as the threshold in order to split the raw data into 2 bins. Finally, binary random numbers are output according to the final state in which the MTJ is in.

III. PROVABLY-SECURE RANDOMNESS EXTRACTION FROM WEAKLY RANDOM SOURCE

Consider an RNG that outputs a binary string. In the ideal scenario, the output has to satisfy 2 conditions, which are 1) all the bits in the string are uniformly distributed, and 2) all the bits are independent from any perspective. An \( m \)-bit string that satisfies both criteria is referred to as the ideal output and denoted as \( U_m \).

For our MTJ-based TRNG, the raw output is most definitely not uniformly distributed. This is because the thermal noise (or any of the other noise sources present) is unlikely to remain constant throughout the entire experiment. Deviations in the thermal noise, no matter how small in magnitude, would skew the distribution of the output bits away from the uniform distribution. In addition, it would be naive to assume that the raw output bits are independent of each other. Imperfections in the experimental setup (e.g. noisy reader) might introduce correlations to the raw output bits. In fact, the raw output data failed the permutation test and Chi-square test outlined in the NIST 800-90B test suite [20], which tests for independence. A sequence under test can be classified as independent and identically distributed (i.i.d.) if they pass these tests.

Clearly, the \( n \)-bit raw output of our MTJ-based TRNG does not satisfy the 2 criteria stated at the beginning of the section. Hence, it is classified as a weakly random source, which we denote as \( X \). We thus have to utilise functions known as randomness extractors (with the help of a seed, \( S \)) to extract an \( m \)-bit almost ideal output, \( Z \), from the raw output of our MTJ-based TRNG. The process can be written as \( Z = \operatorname{Ext}(X, S) \). We use the notion of statistical distance to determine how close \( Z \) is to the ideal output \( U_m \).

The statistical distance between two random variables, \( Y \) and \( Y' \), is defined as

\[
\triangle(Y, Y') := \frac{1}{2} \sum_{y \in \mathcal{Y}} \left| P_Y(y) - P_{Y'}(y) \right|,
\]

where \( \mathcal{Y} \) is the range of values that \( Y \) and \( Y' \) can take. If \( Y \) and \( Y' \) are identically distributed, then their statistical distance is 0. On the other hand, if they are perfectly distinguishable, then their statistical distance is 1.

Intuitively, \( m < n \), because it is not possible to generate randomness from nothing. This also means that for the extraction to be possible, \( X \) must contain at least \( m \)-bits of randomness. To characterise the amount of extractable random bits that is present in a weakly random source, we use the notion of min-entropy. The min-entropy of \( X \) is defined as

\[
H_{\text{min}}(X) := \min_{x \in \mathcal{X}} \left[ -\log_2 P_X(x) \right],
\]

where, \( \mathcal{X} \) is the range of values that \( X \) can take. Notice that we use min-entropy instead of the Shannon entropy to characterise the extractable randomness. To explain why, let us consider an extractor that outputs an ideal \( Z \), i.e. \( P_Z(z) = 2^{-m} \) for all \( z \in \mathcal{Z} \), where \( \mathcal{Z} \) is the range of values that \( Z \) can take. Any \( z \in \mathcal{Z} \) thus has to satisfy

\[
P_Z(z) = \sum_{x: \operatorname{Ext}(X) = z} P_X(x) = 2^{-m}.
\]

Clearly, the implication is that \( P_X(x) \leq 2^{-m} \) for all \( x \in \mathcal{Z} \). Therefore, to extract \( m \) random bits from \( X \), it is necessary that \( H_{\text{min}}(X) \geq m \).

Now that the preliminaries have been introduced, we provide a formal definition for randomness extraction.

**Definition 1 (Strong seeded extractors [21]).** The function \( \operatorname{Ext} : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^m \) is a \((k, \varepsilon)\)-strong extractor if for any source \( X \) with \( H_{\text{min}}(X) \geq k \) and seed \( S = U_r \), its output \( Z = \operatorname{Ext}(X, S) \) satisfies

\[
\triangle(ZU_r, U_mU_r) \leq \varepsilon,
\]

where \( YY' \) represents the concatenation of the strings \( Y \) and \( Y' \).
The implication of a strong extractor is that the output $Z$ remains uniformly random even in the presence of seed $S$, i.e., $Z$ and $S$ are independent of each other. Operationally, this means that we are able to concatenate the seed used to the output of the strong extractor, without any increase in the security parameter. This is important because in Toeplitz hashing, the seed used is longer than the output length. Thus, if we do not use a strong extractor, there will be no net randomness extracted since we consume more randomness than we extract. We follow up by providing the definition of universal hashing.

**Definition 2 (Universal hash functions [22]).** A family of hash functions $\mathcal{F} = \{ f : \{0,1\}^n \rightarrow \{0,1\}^m \}$ with $n \geq m$ is called universal if for $f$ randomly chosen from $\mathcal{F}$,

$$\Pr[f(x) = f(x')] \leq 2^{-m}, \ \forall \ x \neq x' \in \{0,1\}^n.$$  

In our application, we use the random seed $S$ to choose a function from the family of universal hash functions. We denote the chosen function as $f_s$. This also means that the family of universal hash functions has size $|\mathcal{F}| = 2^n$. Importantly, the work in [23] showed that universal hash functions can be used to construct strong seeded extractors. We provide the definition below.

**Definition 3 (Leftover hash lemma [23]).** Let $X$ be a min-entropy source with $H_{\min}(X) \geq k$ and $\mathcal{F} = \{ f_s : \{0,1\}^n \rightarrow \{0,1\}^m \}$ be a universal hashing family of size $2^n$ with $m = \lceil k - 2\log(1/\varepsilon) + 2 \rceil$. Furthermore, let $\text{Ext}(X, s) = f_s(X)$. Then, the extractor is a $(k, \varepsilon)$-strong extractor with seed length $r$ and output length $m$.

Hence, using a family of universal hash functions to perform randomness extraction allows us to obtain an extracted output that is provably-secure, i.e., we are able to bound how close our extracted output is to the ideal output. Another advantage is that such a security definition also adheres to the universal composability framework proposed in [24, 25]. The implication is that if we use the extracted output in other protocols, the security parameter of that protocol will increase by $\varepsilon$.

**IV. ENTROPY ESTIMATION AND EXTRACTION**

We shall denote the $n$-bit raw binary random numbers that our MTJ-TRNG produces as $X$. We can classify $X$ as uniformly random if $H_{\min}(X) = n$. However, as mentioned in section III, more often than not, our MTJ-TRNG produces a string that is not uniformly random, i.e., $H_{\min}(X) < n$. Thus, we have to lower-bound the min-entropy of $X$ using techniques like the NIST 800-90B entropy estimation suite [20] to certify that $H_{\min}(X) \geq k$. Using this estimate of $k$, we are able to quantify the amount of extractable randomness in $X$ and extract an $m$-bit output string $Z$, such that $m \leq k$. The string produced is $\varepsilon$-close to the ideal uniform $m$-bit distribution, $U_m$, i.e., $\Delta(Z, U_m) \leq \varepsilon$.

The results that we obtain from NIST 800-90B indicate that the min-entropy per bit of $X$ is 0.778584. It should be noted that we do not assume independence of the raw bits when performing min-entropy estimation. We collected $n = 217,504,350$ bits, which translates to $k = 169,345,406$ bits. We set our security parameter to be $\varepsilon = 10^{-10}$, and employ the leftover hash lemma [23] to obtain $m = \lceil k - 2\log(1/\varepsilon) + 2 \rceil = 169,345,260$ bits.

The extraction is carried out via Toeplitz hashing, which belongs to a family of universal hash functions [26]. Toeplitz hashing utilises a Toeplitz matrix, $T$. The Toeplitz matrix is an $m \times n$ diagonal-constant matrix, and it is constructed by filling up the first column and first row of the matrix with a uniform seed, denoted by $S$. Thus, the seed length required for Toeplitz hashing is $n + m - 1$ bits. $Z$ is obtained by performing $Z = T \cdot X$, which is expressed as

$$\begin{bmatrix}
    s_n & s_{n-1} & \cdots & s_2 & s_1 \\
    s_{n+1} & s_n & \cdots & s_3 & s_2 \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    s_{n+m-1} & s_{n+m-2} & \cdots & s_{m+1} & s_m \\
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_{n-1} \\
    x_n \\
\end{bmatrix}
= \begin{bmatrix}
    z_1 \\
    z_2 \\
    \vdots \\
    z_{n-1} \\
    z_n \\
\end{bmatrix}.$$

The computational complexity of Toeplitz hashing is $O(n^2)$ due to the need to perform matrix-vector multiplication. However, this is undesirable, as $n$ needs to be a large number for us to be able to accurately characterise the min-entropy of the raw data. To improve the computational complexity of Toeplitz hashing, we utilise the fast Fourier transform (FFT) algorithm. The FFT algorithm speeds up matrix-vector multiplications when the matrix used is a circulant matrix. In this case, Toeplitz matrix $T$ is not a circulant matrix, but it can easily be converted into one by padding $T$ with additional rows and columns, and filling up these rows and columns with 0s and 1s accordingly to obtain $T_{\text{circ}}$. As $T_{\text{circ}}$ has a larger number of columns than $T$, we have to pad $X$ with 0s to get $X_{\text{pad}}$. The main process of Toeplitz hashing accelerated by FFT can then be expressed as

$$Z_{\text{pad}} = IFFT(FFT(X_{\text{pad}}) \cdot FFT(T_{\text{circ}})) = T_{\text{circ}} \cdot X_{\text{pad}},$$

where $FFT$ and $IFFT$ are the FFT algorithm and its inverse respectively, $T_{\text{circ}}$ is the first column of $T_{\text{circ}}$, and $Z_{\text{pad}}$ is the extended output. The original output $Z$ can be recovered by keeping the first $m$ bits of $Z_{\text{pad}}$. The computational complexity of Toeplitz hashing with FFT is $O(n \log n)$, which is an exponential improvement from the initial computational complexity.

**V. RESULTS**

We performed Toeplitz hashing on a personal computer with an Intel Xeon E5-2697 CPU and 128 GB of random access memory (RAM). The total time taken for the extraction was 72.285 seconds, which translates to a throughput speed of 2.3427 Mbits/s.
Similarly, when testing the raw output, we substitute \( \alpha \) that test and is therefore not uniformly random. Sim-
low 0.9670, we conclude that our extracted output fails
returned value of proportion for a specific test is be-
0.9670. The interpretation of this value is that if the

When testing the extracted output, we substitute
\( \alpha \)
and returnOr 2 values, the P-value and proportion.

The proportion parameter is the proportion of se-
tquences that pass the test at the selected significance
level. The range of acceptable proportions is determined
using the confidence interval. It is calculated as

\[
\hat{p} \pm 3 \sqrt{\frac{\hat{p}(1-\hat{p})}{k}},
\]

where \( \hat{p} = 1 - \alpha \) and \( k \) is the number of sequences. When testing the extracted output, we substitute \( \alpha = 0.01 \) and \( k = 169 \) to obtain a confidence interval of 0.9670. The interpretation of this value is that if the returned value of proportion for a specific test is below 0.9670, we conclude that our extracted output fails that test and is therefore not uniformly random. Similarly, when testing the raw output, we substitute \( \alpha = 0.01 \) and \( k = 217 \) to obtain a confidence interval of 0.9697.

The P-value parameter for a specific test, denoted as
\( P_{\text{value}} \), is the result of the Goodness-of-Fit Distribu-
tional Test on the P-values obtained for that statistical
test. The Goodness-of-Fit Distributional Test tests if the
distribution of P-values obtained follows a uniform distri-
bution. If the returned \( P_{\text{value}} < 0.0001 \), then the P-
values failed the uniformity test, and thus, the sequence
under test is considered to be non-uniformly distributed
as well.

VI. DISCUSSION

In summary, we implemented an MTJ-based TRNG,
estimated the min-entropy of the raw output bits, and
performed suitable post-processing on the raw output to
obtain a set of provably-secure random numbers. To the
best of our knowledge, this is the first-ever implementa-
tion of provably-secure random number generation using
an MTJ device. The extracted random numbers pass
all the tests in the NIST STS. Our implementation also
has minimal assumptions as well, as we do not even as-
sume that our raw data are independent or identically
distributed when performing entropy estimation.

| Statistical test            | Raw data P-value | Raw data Proportion | Raw data Result | Extracted data P-value | Extracted data Proportion | Extracted data Result |
|-----------------------------|------------------|---------------------|----------------|------------------------|--------------------------|-----------------------|
| Frequency                   | 0.0000           | 0.1901              | Failure        | 0.2315                 | 1.0000                   | Success               |
| Block frequency             | 0.0000           | 0.8651              | Failure        | 0.9233                 | 0.9882                   | Success               |
| Cumulative sums             | 0.0000           | 0.1839              | Failure        | 0.4201                 | 1.0000                   | Success               |
| Runs                        | -                | -                   | -              | 0.8606                 | 0.9941                   | Success               |
| Longest run                 | 0.0000           | 0.1797              | Failure        | 0.6436                 | 1.0000                   | Success               |
| Rank                        | 0.1806           | 1.0000              | Success        | 0.4547                 | 0.9822                   | Success               |
| FFT                         | 0.0000           | 0.5253              | Failure        | 0.4666                 | 1.0000                   | Success               |
| Non-overlapping template    | 0.0000           | 0.0000              | Failure        | 0.0001                 | 0.9704                   | Success               |
| Overlapping template        | 0.0000           | 0.0335              | Failure        | 0.0024                 | 0.9941                   | Success               |
| Universal                   | 0.0000           | 0.0000              | Failure        | 0.6825                 | 0.9882                   | Success               |
| Approximate entropy         | -                | -                   | -              | 0.5659                 | 0.9941                   | Success               |
| Random excursions           | 0.0003           | 0.8182              | Failure        | 0.2307                 | 0.9703                   | Success               |
| Random excursions variant   | 0.0909           | 0.9091              | Success        | 0.0780                 | 0.9604                   | Success               |
| Serial                      | 0.0000           | 0.0000              | Failure        | 0.1597                 | 0.9763                   | Success               |
| Linear complexity           | 0.3781           | 0.9862              | Success        | 0.3979                 | 0.9882                   | Success               |

TABLE I. Results of the NIST 800-22 statistical test suite for the raw and post-processed data. The minimum values for each
test are tabulated and rounded down to 4 decimal places. To pass an individual statistical test, the post-processed data needs
to obtain 1) P-value \( \geq 0.0001 \); and 2) proportion \( \geq 0.9603 \). Similarly, the raw data needs to obtain 1) P-value \( \geq 0.0001 \); and 2) proportion \( \geq 0.9697 \), except for random excursion variant test, which requires proportion
\( \geq 0.9603 \).
Our work could have been improved by performing an in-depth characterisation of the MTJ device to model it using mathematical statements. By doing so, we will be able to obtain an accurate value of min-entropy, instead of having to rely on min-entropy estimation tests. We could also consider the side-information that is held by an adversary, Eve. Performing min-entropy estimation on the raw data conditioned on Eve’s side-information would allow us to obtain a final extracted output that is uniformly random, even from Eve’s perspective.

ACKNOWLEDGEMENT

We thank Chao Wang for useful inputs and discussions. We acknowledge funding support from the National Research Foundation of Singapore (NRF) Fellowship grant (NRFF11-2019-0001) and NRF Quantum Engineering Programme 1.0 grant (QEP-P2).

[1] Maxim, RandCrack – Python random module cracker / predictor (2017).
[2] M. Matsumoto and T. Nishimura, ACM Transactions on Modeling and Computer Simulation 8, 3 (1998).
[3] C. Petrie and J. Connelly, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 47, 615 (2000).
[4] R. Brederlow, R. Prakash, C. Paulus, and R. Thewes, in 2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers (2006) pp. 1666–1675, ISSN: 2376-8606.
[5] N. Liu, N. Pinckney, S. Hanson, D. Sylvester, and D. Blaauw, in 2011 Symposium on VLSI Circuits - Digest of Technical Papers (2011) pp. 216–217, iSSN: 2158-5636.
[6] Q. Tang, B. Kim, Y. Luo, K. K. Parki, and C. H. Kim, in Proceedings of the IEEE 2014 Custom Integrated Circuits Conference (2014) pp. 1–4. iSSN: 2152-3630.
[7] S. K. Mathew, D. Johnston, S. Satpathy, V. Suresh, P. Newman, M. A. Anders, H. Kaul, A. Agarwal, S. K. Hsu, G. Chen, and R. K. Krishnamurthy, IEEE Journal of Solid-State Circuits 51, 1695 (2016), conference Name: IEEE Journal of Solid-State Circuits.
[8] M. Kim, U. Ha, K. J. Lee, Y. Lee, and H.-J. Yoo, IEEE Journal of Solid-State Circuits 52, 1953 (2017), conference Name: IEEE Journal of Solid-State Circuits.
[9] S. T. Chandraselkaran, V. E. G. Karnam, and A. Sanyal, IEEE Solid-State Circuits Letters 3, 190 (2020).
[10] Won Ho Choi, Yang Lv, Jongyeon Kim, A. Deshpande, Gyuseong Kang, Jian-Ping Wang, and C. H. Kim, in 2014 IEEE International Electron Devices Meeting (2014) pp. 12.5.1–12.5.4.
[11] Z. Fu, Y. Tang, X. Zhao, K. Lu, Y. Dong, A. Shukla, Z. Zhu, and Y. Yang, Frontiers in Physics 9, 172 (2021).
[12] A. Brataas, A. D. Kent, and H. Ohno, Nature materials 11, 372 (2012).
[13] K. Yang, Q. Dong, Z. Wang, Y.-C. Shih, Y.-D. Chih, J. Chang, D. Blaauw, and D. Sylvester, in 2018 IEEE Symposium on VLSI Circuits (IEEE, Honolulu, HI, 2018) pp. 171–172.
[14] E. I. Vatajelu and G. Di Natale, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27, 491 (2019).
[15] B. Perach and s. kvatinsky, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27, 2473 (2019).
[16] J. Slonczewski, Journal of Magnetism and Magnetic Materials 159, L1 (1996).
[17] L. Berger, Phys. Rev. B 54, 9353 (1996).
[18] T. Seki, A. Fukushima, H. Kubota, K. Yakushiji, S. Yuasa, and K. Ando, Applied Physics Letters 99, 112504 (2011), https://doi.org/10.1063/1.3637545.
[19] A. Fukushima, T. Seki, K. Yakushiji, H. Kubota, H. Imamura, S. Yuasa, and K. Ando, Applied Physics Express 7, 083001 (2014).
[20] M. S. Taran, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, NIST 800-90B Recommendation for the entropy sources used for random bit generation, Tech. Rep. NIST SP 800-90b (National Institute of Standards and Technology, 2018).
[21] N. Nisan and D. Zuckerman, Journal of Computer and System Sciences 52, 43 (1996).
[22] J. Carter and M. N. Wegman, Journal of Computer and System Sciences 18, 143 (1979).
[23] R. Impagliazzo, L. A. Levin, and M. Luby, in Proceedings of the twenty-first annual ACM symposium on Theory of computing (Association for Computing Machinery, 1989) pp. 12–24.
[24] R. Canetti, in Proceedings 42nd IEEE Symposium on Foundations of Computer Science (2001) pp. 136–145, iSSN: 1552-5244.
[25] B. Fitzmann and M. Waidner, in Proceedings of the 7th ACM conference on Computer and communications security - CCS ’00 (ACM Press, Athens, Greece, 2000) pp. 245–254.
[26] H. Krawczyk, Advances in Cryptology-CRYPTO’94 839, 129 (1994).
[27] A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks, A. Heckert, and J. Dray, NIST 800-22 A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Tech. Rep. (National Institute of Standards and Technology, 2010).