2012 IEEE Symposium on Security and Privacy

Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints

Yinglei Wang, Wing-kei Yu, Shuo Wu, Greg Malysa, G. Edward Suh, and Edwin C. Kan
School of Electrical and Computer Engineering
Cornell University
Ithaca, NY, USA
{yw437, wsy5, sw626, gjm76, gs272, eck5}@cornell.edu

Abstract— **We demonstrate that unmodified commercial Flash memory can provide two important security functions: true random number generation and digital fingerprinting. Taking advantage of random telegraph noise (a type of quantum noise source in highly scaled Flash memory cells) enables high quality true random number generation at a rate up to 10Kbits / second. A scheme based on partial programming exploits process variation in threshold voltages to allow quick generation of many unique fingerprints that can be used for identification and authentication. Both schemes require no change to Flash chips or interfaces, and do not require additional hardware.**

Keywords - security; flash memory; true random number generation; hardware fingerprints; device authentication

I. INTRODUCTION

Flash memory has gained a ubiquitous place in the computing landscape today. Virtually all mobile devices such as smartphones and tablets rely on Flash memory as their non-volatile storage. Flash memory is also moving into laptop and desktop computers, intending to replace the mechanical hard drive. Floating-gate non-volatile memory is even more broadly used in electronic applications with a small amount of non-volatile memory. For example, even 8-bit or 16-bit microcontrollers for embedded systems commonly have on-chip EEPROMs to store instructions and data. Many people also carry Flash memory as standalone storage medium as in USB memory sticks and SD cards.

In this paper, we propose to utilize analog behaviors of off-the-shelf Flash memory to enable hardware-based security functions in a wide range of electronic devices without requiring custom hardware. More specifically, we show that a standard Flash memory interface can be used to generate true *random numbers* from quantum and thermal noises and to produce *device fingerprints* based on manufacturing variations. The techniques can be applied to any floating-gate non-volatile memory in general, and does not require any hardware modifications to today's Flash memory chips, allowing them to be widely deployed.

Both hardware random number generators (RNGs) and device fingerprints provide important foundations in building secure systems. For example, true randomness is a critical ingredient in many cryptographic primitives and security protocols; random numbers are often required to generate secret keys or prevent replays in communications. While pseudo-random number generators are often used in today's systems, they cannot provide true randomness if a seed is reused or predictable. As an example, a recent study showed that reuse of virtual machine (VM) snapshots can break the Transport Level Security (TLS) protocol due to predictable random numbers [1]. Given the importance of a good source of randomness, high security systems typically rely on hardware RNGs.

Instead of requiring custom hardware modules for RNGs, we found that analog noise in Flash memory bits can be used to reliably generate true random numbers. An interesting finding is that the standard Flash chip interface can be used to put a memory bit in partially programmed state so that the internal noise can be observed through the digital interface. There exist two sources of true randomness in Flash bits, Random Telegraph Noise (RTN) and thermal noise. While both sources can be leveraged for RNGs, our scheme focuses on RTN, which is quantum noise. Unlike thermal noise, which can be reduced significantly at extremely low temperatures, RTN behavior continues at all temperature ranges. Moreover, the quantum uncertainty nature of RTN provides a better entropy source than system level noises which rely on the difficulty of modeling complex yet deterministic systems. Our algorithm automatically selects bits with RTN behavior and converts RTN into random binary bits.

Experimental results demonstrate that the RTN behavior exists in Flash memory and can be converted into random numbers through the standard Flash interface. The Flash-based RNG is tested using the NIST test suite [2] and is shown to pass all tests successfully. Moreover, we found that the RNG works even at a very low temperature (-80 °C). In fact, the RTN behavior is more visible at low temperatures. On our test platform, the Flash RNG generates about 1K to 10K bits per second. Overall, the experiments show that true random numbers can be generated reliably from off-the-shelf Flash memory chips without requiring custom circuits.

In addition to generating true random numbers, we also found that the standard Flash interface can be used to extract fingerprints (or signatures) that are unique for each Flash chip. For this purpose, our technique exploits inherent random variations during Flash manufacturing processes. More specifically, we show that the distributions of transistor threshold voltages can be measured through the standard Flash interface using incremental partial programming. Experimental results show that these threshold voltage distributions can be used as fingerprints, as they are significantly different from chip to chip, or even from location to location within a chip. The distributions also stay

© 2012, Yinglei Wang. Under license to IEEE.
DOI 10.1109/SP.2012.12

IEEE computer society

relatively stable across temperature ranges and over time. Thanks to the large number of bits (often several gigabits) in modern Flash chips, this technique can generate a large number of independent fingerprints from each chip.

The Flash fingerprints provide an attractive way to identify and/or authenticate hardware devices and generate device-specific keys, especially when no cryptographic module is available or a large number of independent keys are desired. For example, at a hardware component level, the fingerprints can be used to distinguish genuine parts from counterfeit components without requiring cryptography to be added to each component. The fingerprinting technique can also be used for other authentication applications such as turning a Flash device into a two-factor authentication token, or identifying individual nodes in sensor networks.

While the notion of exploiting manufacturing process variations to generate silicon device fingerprints and secret keys is not new and has been extensively studied under the name of Physical Unclonable Functions (PUFs) [3], the Flash-based technique in this paper represents a unique contribution in terms of its practical applicability. Similar to true RNGs, most PUF designs require custom circuits to convert unique analog characteristics into digital bits. On the other hand, our technique can be applied to off-the-shelf Flash without hardware changes. Researchers have recently proposed techniques to exploit existing bi-stable storage elements such as SRAMs [4] or Flash cells [5] to generate device fingerprints. Unfortunately, obtaining fingerprints from bi-stable elements requires a power cycle (power off and power on) of a device for every fingerprint generation. The previous approach to fingerprinting Flash only works for a certain types of Flash chips and takes long time (100 seconds for one fingerprint) because it relies on rare errors called program disturbs. As an example, we did not see any program disturbs in SLC Flash chips that we used in experiments. To the best of our knowledge, the proposed device fingerprinting techniques is the first that is fast (less than 1 second for a 1024-bit fingerprint) and widely applicable without interfering with normal operation or requiring custom hardware.

The following list summarizes the main strengths of the proposed security functions based on Flash memory over existing approaches for hardware random number generators and fingerprints.

- *Widely applicable*: Flash memory already exists in many electronic devices. The proposed techniques can often be implemented as system software or firmware updates without hardware changes.
- *Non-intrusive*: the techniques do not require a reboot and only have minimal interference with normal memory operations. Only a small portion of Flash needs to be used for security functions during security operations. There is minimal wear-out.
- *High security*: the Flash random number generator is based on quantum noise, which exists even at extremely low temperatures. Thanks to the high capacity of today's Flash memory, a very large number of independent signatures can be generated from Flash.

The rest of the paper is organized as follows. Section II provides the basic background on the Flash memory. Based on this understanding, Section III and Section IV explain the new techniques to generate random numbers and device fingerprints through standard Flash interfaces. Then, Section V studies the effectiveness and the security of the proposed methods through experimental results on real Flash chips. Section 0 briefly discusses a few examples of application scenarios. Finally, Section VII discusses related work and Section VIII concludes the paper.

II. Flash Memory Basics

This section provides background material on Flash memory and its operating principles in order to aid understanding of our Flash-based security schemes.

A. Floating Gate Transistors

Flash memory is composed of arrays of floating-gate transistors. A floating-gate transistor is a transistor with two gates, stacked on top of each other. One gate is electrically insulated (floating). Figure 1 shows an example of a floating-gate device. The control gate is on top. An insulated conductor, surrounded by oxide, is between the control gate and the channel. This conductor is the floating gate. Information is stored as the presence or absence of trapped charge on the floating gate. The trapped negative charge reduces the current flowing through the channel when the N-type MOS transistor is on. This current difference is sensed and translated into the appropriate binary value.

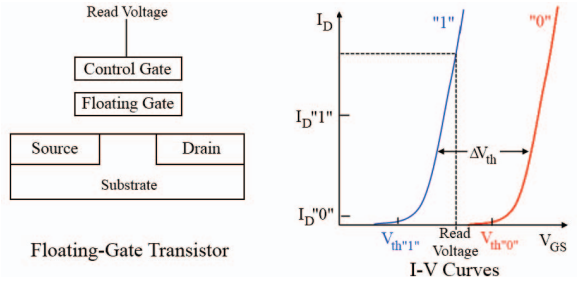

Figure 1. Flash memory cell based on a floating gate transistor.

Flash cells without charge on their floating-gate allow full current flow in the channel and hence are read as a binary "1". The presence of charge on the floating-gate will discourage the presence of current in the channel, making the cell store a "0". Effectively, the charge on the floating-gate increases the threshold voltage (V_{th}) of a transistor. Single-level cells store one bit of information; multi-level cells can store more than one bit by reading and injecting charge to adjust the current flow of the transistor.

Note that the threshold voltage without any charge on the floating-gate is different for each transistor due to variations in manufacturing processes. As a result, the amount of charge that needs to be stored to the floating-gate for a cell to reliably represent a "0" state varies from cell to cell. If the threshold voltage is not shifted sufficiently, a cell can be in an unreliable (partially programmed) state that can be

interpreted as either 1 or 0. In this paper, we exploit the threshold voltage variations and the partially programmed state to extract fingerprints and random numbers.

B. Flash Organization and Operation

At a high-level, Flash memory provides three major operations: *read*, *erase*, and *program* (write). In order to read a bit in a Flash cell, the corresponding transistor is turned on and the amount of current is detected. A write to a Flash cell involves two steps. First, an erase operation pushes charge off the floating-gate by applying a large negative voltage on the control gate. Then, a program (write) operation stores charge on the floating-gate by selectively applying a large positive voltage if the bit needs to be zero.

An important concept in Flash memory operation is that of *pages* and *blocks*. *Pages* are the smallest unit in which data is read or written, and are usually 2KB to 8KB. *Blocks* are the smallest unit for an erase operation and made up of several pages, usually 32 - 128 pages. Note that Flash does not provide bit-level program or erase. To read an address from a Flash chip, the page containing the address is read. To update a value, the block that includes the address must be first erased, then the corresponding page is written with an update and other pages in the block are restored.

III. Random Number Generation

A. Random Telegraph Noise (RTN)

The proposed RNG uses a device effect called Random Telegraph Noise (RTN) as the source of randomness. In general, RTN refers to the alternating capture and emission of carriers at a defect site (trap) of a very small electronic device, which generates discrete variation in the channel current [6]. The capture and emission times are random and exponentially distributed. RTN behavior can be distinguished from other noise using the power spectrum density (PSD), which is flat at low frequencies and $1/f^2$ at high frequencies. In Flash memory, the defects that cause RTN are located in the tunnel-oxide near the substrate. The RTN amplitude is inversely proportional to the gate area and nearly temperature independent. As Flash memory cells shrink, RTN effects become relatively stronger and their impact on the threshold distribution of Flash memory cells, especially for multi-level cells, can be significant. Because RTN can be a major factor in Flash memory reliability, there have been a large number of recent studies on RTN in Flash memory from a reliability perspective [7] [8] [9].

While RTN is a challenge to overcome from the perspective of Flash memory operations, it can be an ideal source of randomness. RTN is caused by the capture and emission of an electron at a single trap, and is a physical phenomenon with random quantum properties. Quantum noise can be seen as the "gold-standard" for random number generation because the output of quantum events cannot be predicted. As Flash memory cells scale to smaller technology nodes, the RTN effect will become stronger. Moreover, RTN behavior will still exist with increasing process variation and at extremely low temperatures.

B. Noise Extraction from Digital Interface

As digital devices, Flash memory is designed to tolerate analog noise; noise should not affect normal memory operations. In order to observe the noise for random number generation, a Flash cell needs to be in an unreliable state between well-defined erase and program states. Interestingly, we found that Flash cells can be put into the in-between state using the standard digital interface. In a high level, the approach first erases a page, issues a program command, and then issues a reset command after an appropriate time period to abort the program. This procedure leaves a page partially programmed so that noise can affect digital outputs. We found that the outcome of continuously reading a partially programmed bit oscillates between 1 and 0 due to noise.

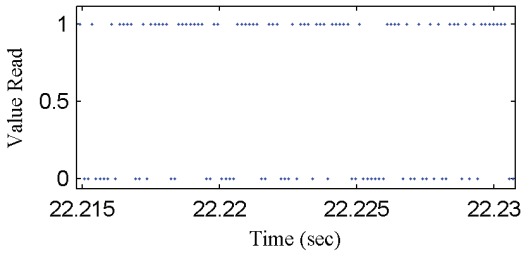

Figure 2. Thermal noise in Flash memory (time domain).

For Flash memory in practice, experiments show that two types of noise coexist: thermal noise and RTN. Thermal noise is white noise that exists in nearly all electronic devices. RTN can be observed only if a surface trap exists, the RTN amplitude is larger than that of thermal noise, and the sampling frequency (speed for continuous reads) is high enough. If any of these three conditions is not satisfied, only thermal noise will be observed as in Figure 2. In the case of thermal noise, a bit oscillates between the two states quickly, and the power spectral density (PSD) indicates white noise.

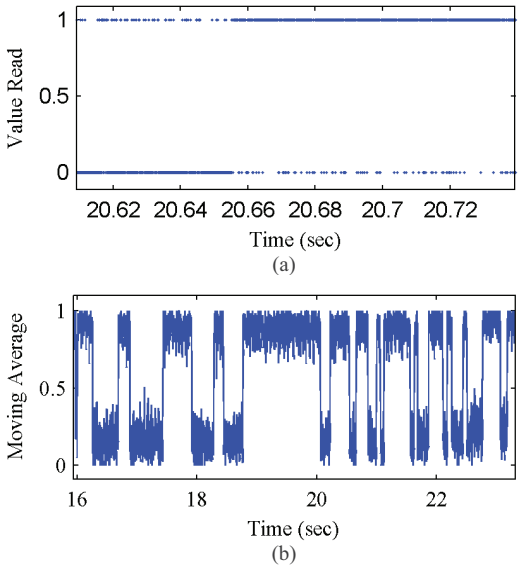

Figure 3. RTN with thermal noise in Flash memory. (a) Time domain. (b) Moving average of 29 points on the time domain.

In the case that the RTN amplitude is comparable to thermal noise, a combination of RTN and thermal noise is observed as shown in Figure 3. This is reflected by the density change of 1s in the continuous reading. A moving average on the time domain helps to visualize the density change. The PSD of the result shows $1/f^2$ spectrum at low frequencies and becomes flat at high frequencies.

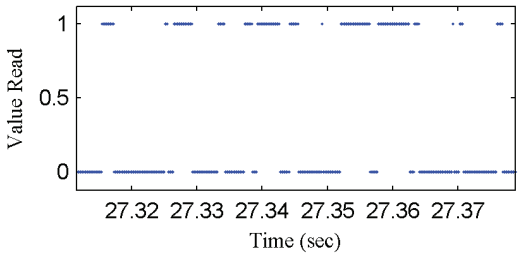

Figure 4. RTN in Flash memory (time domain).

In some cases, the RTN amplitude is very high and dominates thermal noise. As a result, only RTN behaviors are visible through digital interfaces for these bits. As shown in Figure 4, continuous reads show clear clusters of 1s and 0s in the time domain. The power spectral density (PSD) of these bit sequences shows a clear RTN pattern of $1/f^2$.

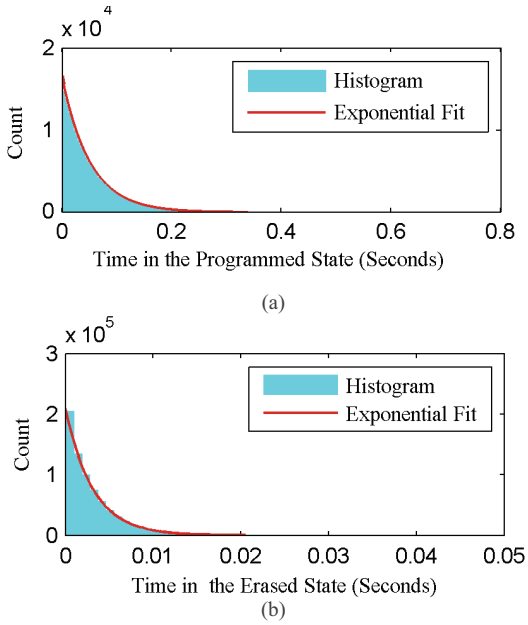

Figure 5. (a) Distribution of time in the programmed state. (b) Distribution of time in the erased state.

For a bit with nearly pure RTN behavior, we further validated that the error pattern corresponds to RTN by plotting the distributions of up and down periods. As shown in Figure 5, both up time and down time nicely fit an exponential distribution as expected. Overall, our experiments show that both RTN and thermal noise exist in Flash memory and can be observed through a digital interface. While both noise types can be used for random number generation, we focus on RTN, which is more robust to temperature changes.

C. Random Number Generation Algorithms

In Flash memory devices, RTN manifests as random switching between the erased state (consecutive 1s) and programmed state (consecutive 0s). At a high-level, our Flash random number generator (RNG) identifies bits with RTN behavior, either pure RTN or RTN combined with thermal noise, and uses a sequence of time in the erased state (called *up-time*) and the time in the programmed state (called *down-time*) from those bits. In order to produce random binary outputs, the RNG converts the up-time and down-time sequence into a binary number sequence, and applies the von Neumann extractor for de-biasing. We found that thermal noise itself is random and does not need to be filtered out.

Algorithm I Overall Flash RNG algorithm

```
Erase a block;

Num = 0;
do {
  Partially program a page for T;
  Num++;

  Read Nbytes in a page N times, and record a
  trace for each bit - trace[bit];
  For each bit in Nbytes, not selected yet
    If (CheckRTN(trace[bit]) == true) {
      Selected[bit] = yes;
      NumProgram[bit] = Num;
    }
  End for
} repeat until most bits are programmed.

ProgramSelectBits(Selected);

Read selected bits M times, and record up-
time and down-time;
For each bit
  ConvertToBinary(rawdata);
End for
```

Algorithm I shows the overall RNG algorithm. To generate random numbers from RTN, the first step is to identify bits with RTN or both RTN and thermal noise. To do this, one block in Flash memory is erased and then multiple incomplete programs with the duration of `T` are applied. After each partial program, a part of the page is continuously read `N` times and the outcome is recorded for each bit. In our experiments, we chose to read the first 80 bits (10 bytes) in a page for 1,000 times. For each bit that has not been selected yet, the algorithm checks if RTN exists using `CheckRTN()` and marks the bit location if there is RTN. As an optimization, the algorithm also records the number of partial programs when a bit is selected. The algorithm repeats the process until all bits are checked for RTN. The second step is to partially program all of the selected bits to an appropriate level so that they will show RTN behavior. Finally, the algorithm reads the selected bits `M` times, records

a sequence of up-time and down-time for each bit, and converts the raw data to a binary sequence.

Algorithm II Determine whether there is RTN in a bit

```
If trace[bit] has over 98% 1/0s
  Return false;
End if

Calculate the power spectrum density (PSD);
Convert PSD to the log scale in both x-y;

If PSD slope is always < T_slope for all high
frequency (> T_freq)
  Return RTN
End if

If PSD slope is < T_slope at least one interval
(Invl) at a high frequency (> T_freq)
  Return RTN-Thermal
End if
```

The function `CheckRTN()` in Algorithm II determines whether there is RTN in a bit based on a trace from `N` reads. The algorithm first filters out bits that almost always (more than 98%) produce one result, either 1 or 0. For the bits with enough noise, the algorithm uses the power spectral density (PSD) to distinguish RTN from thermal noise; PSD for RTN has a form of $1/f^2$ at a high frequency. To check this condition, the algorithm computes the PSD, and converts it to a log-scale in both x and y axes. If the result has a slope less than T_{slope} (we use -1.5, the ideal value is -2) for all frequencies higher than T_{freq} (we use 200Hz), the algorithm categorizes the bit as RTN only. If the PSD has a slope less than T_{slope} for any interval larger than than `Invl` (we use 0.2) at a high frequency, the bit is categorized as a combination of RTN and thermal noise.

Algorithm III Program selected bits to proper levels where RTN could be observed.

```
For each selected bit
  Do (NumProgram[bit]-K) partial programs;

  do {
    Partially program the bit for T;

    Read the bit N times;
    Find Max and Min for moving averages;

    If Max > TMax and Min < TMin
      Break;
    End if
  } repeat up to L times
End for
```

The function `ProgramSelectBits()` in Algorithm III programs selected bits to a proper level where RTN can be observed. Essentially, the algorithm aims to take each bit to the point near where they were identified to have RTN. The number of partial programs that were required to reach this point before were recorded in `NumProgram[Bit]`. For each selected bit, the algorithm first performs partial programs with the duration of `T` based on the number recorded earlier (`NumProgram[Bit]-K`). Then, the algorithm performs up to `L` more partial program operations until a bit shows RTN behavior. The RTN behavior is checked by reading the bit `N` times, and see if the maximum of moving averages is greater than a threshold (`TMax` = 0.7) and the minimum is less than another threshold (`TMin` = 0.3).

Algorithm IV Convert the raw data to binary random sequence.

```
If the bit has both RTN and thermal noise
  For each up/down-time in raw data
    Output = LSB(up/down-time);
  End for
End if

If the bit has only RTN
  do {
    For each up/down-time in raw data
      Output = LSB(up/down-time);
      Shift right up/down-time by one bit;
    End for
  } repeat until all up/down time are zero;
End if

Perform von Neumann de-biasing
```

Finally, the function `ConvertToBinary`() converts the raw data to a binary random sequence. For bits with both RTN and thermal noise, the up-time and down-time tend to be short. So only the LSBs of these numbers are used. Essentially, for every up-time and down-time, the algorithm produces 1 if the time is odd and 0 otherwise. Effectively, this is an even-odd scheme. For bits with perfect RTN behavior, up-time and down-time tend to be longer and we use more LSBs from the recorded up/down-time. In this case, we first produce a bit based on the LSB, then the second LSB, the third LSB, and so on until all extracted bits become 0. Finally, for both methods, we apply the von Neumann de-biasing method. The method takes two bits at a time, throws away both bits if they are identical, and takes the first bit if different. This process is described in Algorithm IV.

The stability of the bits in the partially programmed state is also important. We define the stability as how long a bit stays in the partially programmed state where RTN behavior can be observed. This is determined by the retention time of the Flash memory chip and the amplitude of the RTN compared to the designed noise margin. Assume the amplitude of the RTN is `Ar`, the noise margin of Flash memory is `An`, and the Flash retention time is 10 year, then the stable time for random number generation after partial programming will be roughly `Ts=Ar/An*10` years. This means that after time `Ts`, a bit needs to be reset and reprogrammed. In our experiments, the bit that is shown in Figure 5 was still showing ideal RTN behavior even after 12 hours.

IV. DEVICE FINGERPRINTS

This section describes techniques to generate unique fingerprints from Flash memory devices.

A. Sources of Uniqueness

Flash memory is subject to random process variation like any other semiconductor device. Because Flash is fabricated for maximum density, small variations can be significant. Process variation can cause each bit of a Flash memory to differ from its neighbors. While variation may affect many aspects of Flash cells, our fingerprinting technique exploits threshold voltage variations. Variations in doping, floating gate oxide thickness, and control-gate coupling ratio can cause the threshold voltage of each transistor to vary. Because of this threshold voltage variation, different Flash cells will need different times to be programmed.

B. Extracting Fingerprints

In this paper, we introduce a fingerprinting scheme based on partial programming. We repeatedly partially program a page on a Flash chip. After each partial program, some bits will have been programmed enough to flip their states from 1 to 0. For each bit in the page, we record the order in which the bit flipped. Pseudo-code is provided in Algorithm V. In our experiments, T is chosen to be 29.3us. A short partial program time provide a better resolution to distinguish different bits with the cost of increased fingerprinting time. We do not enforce all bits to be programmed, in order to account for the possibility of faulty bits.

Algorithm V Extract the order in which bits in a page are reach the programmed state.

```
Choose a partial programming time T (below
the rated program time).

Nbits = number of bits in one page
Order = 1;
Initialize BitRank[Nbits] to 0.

do {
    Partially program a page for T;
    For all programmed bits do
        BitRank[programmed bit] = Order;
    End for
    Order = Order + 1;
} repeat until most (99%) bits in the page
are programmed
```

C. Comparing Fingerprints

The fingerprints extracted from the same page on the same chip over time are noisy but highly correlated. To compare fingerprints extracted from the same page/chip and different pages/chips, we use the Pearson correlation coefficient [5], which is defined as

$$P(X,Y) = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

where X is the vector of program orders extracted from one experiment and Y is another vector of program orders extracted from another experiment. μ_X and σ_X are the mean and standard deviation of the X vector. μ_Y and σ_Y are the mean and standard deviation of the Y vector.

In this way, the vector of program orders is treated as a vector of realizations of a random variable. For vectors extracted from the same page, Y=aX+b+noise where a and b are constants and the noise is small. So, X and Y are highly correlated and the correlation coefficient should be close to 1. For vectors extracted from different pages, X and Y should be nearly independent of each other, so the correlation coefficient should be close to zero. From another perspective, if both X[i] and Y[i] are smaller or bigger than their means, $(X[i]-\mu_X)(Y[i]-\mu_Y)$ would be a positive number. If not, it would be a negative number. If X and Y are independent, it is equally likely to be positive and negative so the correlation coefficient would approach 0.

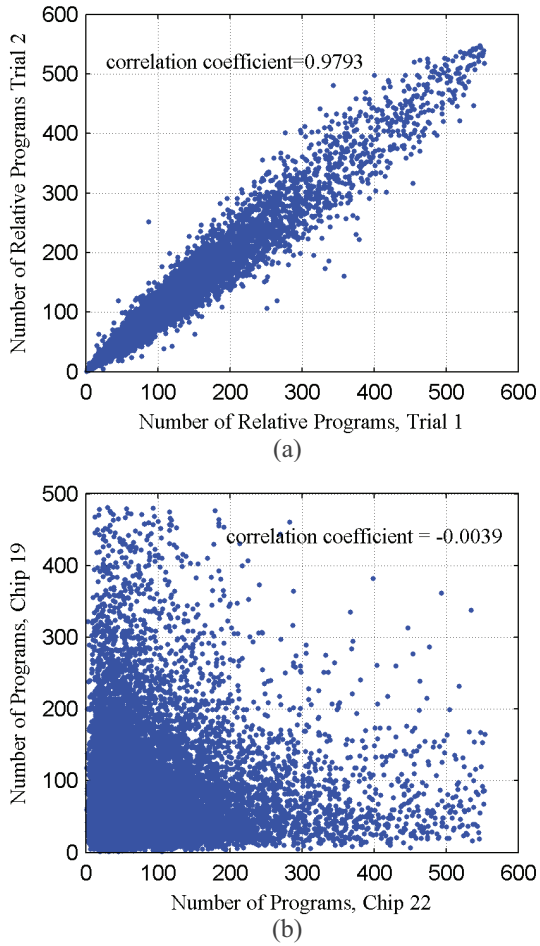

Figure 6. Scatter plot for fingerprints extracted on (a) the same page and (b) different chips.

The scatter plot of X and Y from the same page/chip and from different chips are shown in Figure 6. The figure clearly demonstrates a high correlation between fingerprints

from the same chip over time and a low correlation between fingerprints from different chips. Therefore, this correlation metric can be used to compare fingerprints to determine whether they are from the same page/chip or from different pages/chips.

D. Fingerprints in Binary Numbers

The above fingerprints are in the form of the order in which each bit was programmed. If an application requires a binary number such as in generating cryptographic keys, we need to convert the recorded ordering into a binary number.

There are a couple of ways to generate unique and unpredictable binary numbers from the Flash fingerprints. First, we can use a threshold to convert a fingerprint based on the programming order into a binary number as shown in Algorithm VI. In the algorithm, we produce 1 if the program order is high, or 0 otherwise. This approach produces a 1 bit fingerprint for each Flash bit. Alternatively, we can obtain a similar binary fingerprint directly from Flash memory by partially programming (or erasing) a page and reading bits (1/0) from the Flash.

Algorithm VI Generate a binary signature from the partial programming order information.

```
Pick threshold t = Max(BitRank) / 2
For each bit
    If BitRank[bit] > t
        Output 1
    Else Output 0
End for
```

V. Experimental Results

This section presents evaluation results for the random number generation and fingerprint techniques for Flash memory devices.

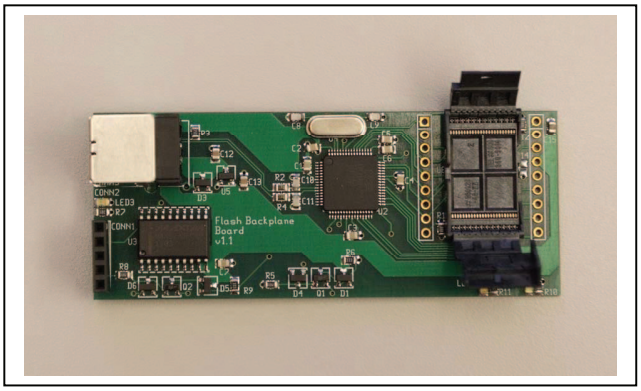

Figure 7. Flash test board.

A. Testbed Device

Our experiments use a custom Flash test board as shown in Figure 7. The board is made entirely with commercial off-the-shelf (COTS) components with a custom PCB board. There is a socket to hold a Flash chip under test, an ARM microprocessor to issue commands and receive data from the Flash chip, and a Maxim MAX-3233 chip to provide a serial (RS-232) interface. USB support is integrated into the ARM microcontroller. We also wrote the code to test the device. The setup represents typical small embedded platforms such as USB flash drives, sensor nodes, etc. This device shows that the techniques can be applied to commercial off-the-shelf devices with no custom integrated circuits (ICs).

The experiments in this paper were performed with four types of Flash memory chips from Numonyx, Micron and Hynix, as shown in 0.

TABLE I. Tested Flash Chips

Manufacturer	Part Number	Capacity	Quantity	Technology
Numonyx	NAND04GW3B2 DN6	4Gbit	3	57nm SLC
Hynix	HY27UF084G2B	4Gbit	10	SLC
Micron	MT29F2G08ABA EAWP-IT:E 4	2Gbit	24	34nm SLC
Micron	MT29F16G08CB ACAWP:C	16Gbit	5	MLC

B. Random Number Generation

The two main metrics for random number generation are randomness and throughput. For security, the RNG must be able to reliably generate true random numbers across a range of environmental conditions over time. For performance, higher throughput will be desirable.

TABLE II. Summary of the NIST test suite

Test Name	Test Description
1 The Frequency (Monobit) Test:	Tests proportion of zeros and ones for the whole sequence.
2 Frequency Test within a Block	Tests the proportions of ones within M-bit Block.
3 The Run Test	Tests the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits
4 Tests for the Longest-Run-of-Ones in a Block	Tests the longest run of ones within M-bit Block and consistency with theory
5 The Binary Matrix Rank Test	Tests rank of disjoint sub-matrices of the entire sequence and independence
6 The Discrete Fourier Transform (Spectral) Test	Tests the peak heights in the Discrete Fourier Transform of the sequence, to detect periodic features that indicates deviation of randomness
7 The Non-overlapping Template Matching Test	Tests the number of occurrences of a pre-specified target strings
8 The Overlapping Template Matching Test	Tests the number of occurrences of a pre-specified target strings. When window found, slide only one bit before the next search
9 Maurer's "Universal Statistics" Test	Tests the number of bits between matching patterns
10 The Linear Complexity Test	Tests the length of a linear feedback shift register, test complexity
11 The Serial Test	Tests the frequency of all possible overlapping m-bit pattern
12 The Approximate Entropy Test	Tests the frequency of all possible overlapping m-bits pattern across the entire sequence
13 The Cumulative Sums (Cusums) Test	Tests maximal excursion from the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence
14 The Random Excursion Test	Tests the number of cycles having exactly K visits in a cumulative sum random walk
15 The Random Excursions Variant Test	Tests the total number of times that a particular state is visited in a cumulative sum random walk

1) Randomness

Historically, three main randomness test suites exist. The first one is from Donald Knuth's book "The Art of computer Programming (1st edition, 1969)" [10] which is the most quoted reference in statistical testing for RNGs in literature. Although it was a standard for many decades, it appears to be outdated in today's view and it allows many "bad" generators to pass the tests. The second one is the "diehard" test suite from Florida State University. The test suite is stringent in the sense that they are difficult to pass. However, the suite has not been maintained in recent years. Therefore, it was not selected as the tests for this study. The third one is developed by National Institute of Standards and Technology (NIST) which is a measurement standard laboratory and a non-regulatory agency of the United States Department of Commerce. The NIST Statistical Test Suite is a package consisting of 15 tests that were developed to test the randomness of arbitrary long binary sequences produced by either hardware or software. The test suite makes use of both existing algorithms from past literatures and newly developed tests. The most updated version, sts-2.1.1, which was released in August 11, 2010, is used in our randomness tests. TABLE II summarizes the 15 NIST tests [2].

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

P-VALUE	PROPORTION	STATISTICAL TEST
0.122325	10/10	Frequency
0.911413	10/10	BlockFrequency
0.534146	10/10	CumulativeSums
0.066882	10/10	CumulativeSums
0.534146	10/10	Runs
0.350485	10/10	LongestRun
0.739918	10/10	Rank
0.739918	10/10	FFT
0.213309	10/10	NonOverlappingTemplate
0.739918	10/10	NonOverlappingTemplate
0.350485	10/10	OverlappingTemplate
0.911413	9/10[1]	Universal
0.534146	10/10	ApproximateEntropy
----	5/5	RandomExcursions
----	5/5	RandomExcursions
----	5/5	RandomExcursionsVariant
----	5/5	RandomExcursionsVariant
0.739918	10/10	Serial
0.350485	10/10	Serial
0.534146	10/10	LinearComplexity

[1]The minimum pass rate for each statistical test is 8 for a sample size of 10 binary sequences, and 4 for a sample size of 5 sequences.

Figure 8. NIST test suite results for bits with RTN and thermal noise.

Figure 8 shows one test result for the even-odd scheme, which only used an LSB from the up-time and down-time, when bits with both RTN and thermal noise are used. 10 sequences generated from multiple bits are tested and each sequence consists of 600,000 bits. Note that some of the results are not shown here due to the space constraint. `NonOverlappingTemplate`, `RandomExcursions` and `RandomExcursionsVariant` have a lot of tests. In the result above, the proportion in the second column shows the proportion of the sequences which passed the test. If the proportion is greater than or equal to the threshold value specified at the bottom of the figure (8 out of 10 or 4 out of 5), then the data is considered random. The P-value in the first column indicates the uniformity of the P-values calculated in each test. If P-value is greater than or equal to 0.0001, the sequences can be considered to be uniformly distributed [2]. The result indicates that the proposed RNG passes all the NIST tests.

We also tested random numbers from one bit with only RTN behavior, using multiple bits from up-time and down-time. In this case, we generated ten 200,000-bit sequences from one bit. The data passed all NIST tests with results that are similar to the above case. For the `Universal` test, which requires a sequence longer than 387,840 bits, we used five 500,000-bit sequences.

2) Performance

The throughput of the proposed RNG varies significantly depending on the switching rate of individual bits, sampling speed and environment conditions. Typically, only a small fraction of bits show pure RTN behavior with minimal thermal noise. TABLE III shows the performance of Flash chips from four manufacturers. The average throughput ranges from 848 bits/second to 3.37 Kbits/second. Note that the fastest switching trap that can be identified is limited by the reading speed in our experiments.

TABLE III. PERFORMANCE OF BITS WITH PURE RTN BEHAVIOR.

Chip	Hynix SLC	Numonyx SLC	Micron SLC	Micron MLC
Reading speed (KHz)	46.51	45.25	43.10	17.78
Number of bits characterized	303	478	1030	134
Number of bits identified	9	16	5	0
Max throughput (bits/sec)	8.03K	5.35K	2.71K	--
Ave. throughput (bits/sec)	3.27K	1.79K	848.29	--
Min throughput (bits/sec)	107.04	34.77	8.14	--

If bits with both RTN and thermal noise are also used, the percentage of bits which can be used for RNG can be much higher. The performance of these bits from the same Flash chips as in the pure RTN case is shown in TABLE IV. The average throughputs are higher because thermal noise is high frequency noise.

TABLE IV. PERFORMANCE OF BITS WITH BOTH RTN AND THERMAL NOISE.

Chip	Hynix SLC	Numonyx SLC	Micron SLC	Micron MLC
Reading speed (KHz)	46.51	45.25	43.10	17.78
Number of bits characterized	303	478	1030	134
Number of bits identified	27	81	58	28
Max throughput (bits/sec)	11.48K	9.68K	10.03K	3.83K
Ave. throughput (bits/sec)	3.28K	3.87K	3.53K	1.26K
Min throughput (bits/sec)	28.39	10.21	8.14	55.12

In our tests, the RNG throughput is largely limited by the timing of the asynchronous interface which is controlled by an ARM microcontroller with CPU frequency of 60MHz and the 8-bit bus for a Flash chip. We believe that the RNG

performance can be much higher if data can be transferred more quickly through the interface. As an example, the average for RTN transition time is reported to range from 1 microsecond to 10 seconds [11]. If a 128 bytes can be read in 6 microseconds which is the ideal random cache read speed for the Micron SLC chips, a RTN bit with 0.1ms average transition time will give approximately 20 Kbits/second throughput. Note that one page could have multiple RTN bits and our algorithm allows using multiple bits in parallel so that the aggregated throughput of an RNG can be much higher. For example, if N bits can be read at a time, in theory, that can increase the throughput by a factor of N.

3) Temperature Variations

For traditional hardware RNGs, low temperatures present a particular challenge because thermal noise, which they typically rely on, can be reduced with the temperature. To study the effectiveness of the Flash-based RNG in low temperatures, we tested the scheme at two low temperature settings: one in a freezer, which is about -5°C, and the other in dry ice, which is about -80°C. The generated random sequences are tested individually as well as combined together with data from experiments at room temperature. All of them passed the NIST test suite without a problem, showing that our technique is effective at low temperatures.

Note that the experiments for temperature variations and aging are performed with a setup where data from Flash memory are transferred from a testbed to a PC through an USB interface. The post processing is performed on the PC. The USB interface limits the Flash read speed to 6.67KHz. As a result, the throughput in this setup is noticeably slower than the results in previous subsections where the entire RNG operation is performed on a microcontroller.

To understand the impact of temperature variations on the Flash-based RNG, we tested the first 80 bits of a page from a Numonyx chip. At room temperature, 62 bits out of the 80 bits showed oscillations between the programmed state and erased state. 14 bits out of the 62 bits were selected by the selection algorithm, which identifies bits with pure RTN or both RTN and thermal components. The throughputs of the 14 bits are shown in Figure 9.

Figure 10 and Figure 11 show the performance of the RNG at -5 °C and -80 °C, respectively. At -5 °C, 79 bits out of 80 bits showed noisy behavior and 20 out of 79 bits were selected by the RNG algorithm as ones with RTN. At -80 °C, 72 bits out of 80 bits showed noise and 28 out of 72 bits were selected as the ones with RTN. On average, we found that per-bit throughput is slightly decreased at low temperatures, most likely because of reduced thermal noise and possibly because of slowed RTN switching. However, the difference is not significant. In fact, a previous study [12] claimed that RTN is temperature independent below 10 Kelvin. Interestingly, we found that the number of bits that are selected by our algorithm as ones with RTN behavior increases at a low temperature. This trend is likely to be because the low temperature decreases thermal noise amplitude while RTN amplitude stays almost the same and the RTN traps slow down so that they become observable at our sampling frequency.

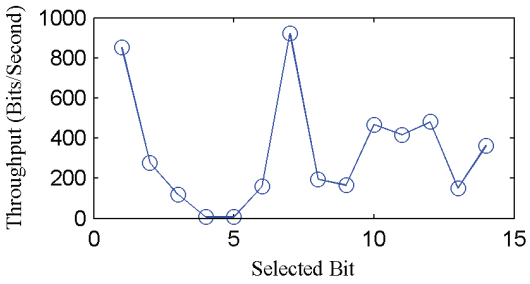

Figure 9. Throughputs under room temperature.

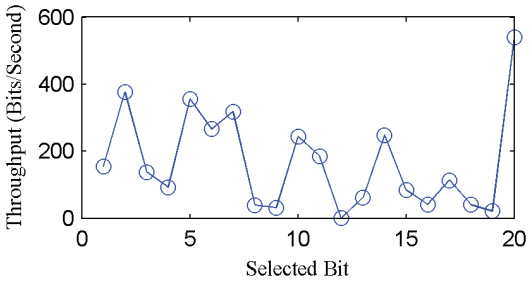

Figure 10. Throughput at -5 °C.

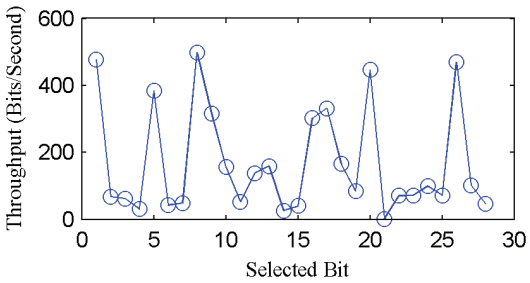

Figure 11. Throughputs at -80 °C.

4) Aging

Flash devices wear-out over time as more program/erase (P/E) operations are performed. A typical SLC Flash chip has a lifetime of 1 million P/E cycles. In the context of RNGs, however, we do not think that wear-outs cause concerns. In fact, aging can create new RTN traps and increase the number of bits with RTN. To check the impact of aging on the RNG, we tested the scheme after 1,000 P/E operations and 10,000 P/E operations as shown in TABLE V. The RNG outputs passed the NIST test suite in both cases and did not show any degradation in performance.

TABLE V. PERFORMANCE SUMMARY OF RTN IN STRESSED PAGES

Stress (P/E)	Bits with noise	Bits selected	Ave. throughput (bits/sec)
1,000	64	9	303.26
10,000	70	15	239.66

The table shows an interesting trend that more bits show RTN behavior after 10,000 P/E cycles. The increase in noisy bits can potentially increase the overall RNG throughput. One possible concern with aging is a decrease in "stable time period" during which each bit shows noisy behavior. In our

experiments, we found that a bit can be used for random number generation for over 12 hours after one programming (Algorithm III). If a bit is completely worn out, charge can leak out more quickly, requiring more frequent calibration. However, given that Flash memory is designed to have a retention time of 10 years within its lifetime, we do not expect the leakage to be a significant problem. We plan to perform larger scale experiments to understand how often a bit needs to be re-programmed for reliable random number generation. In practice, a check can also be added to ensure that a bit oscillates between 1 and 0.

C. Fingerprints

For fingerprinting, we are interested in uniqueness and robustness of fingerprints. The fingerprint should be *unique*, which means that fingerprints from different chips or different locations of the same chip must be significantly different – the correlation coefficient should be low. The fingerprint should also be *robust*, in a sense that fingerprints from a given location of a chip must stay stable over time and even under different environmental conditions – the correlation coefficient should be high.

In the experiments detailed below, we used 24 chips (Micron 34nm SLC), and 24 pages (6 pages in 4 blocks) from each chip. 10 measurements were made from each page. Each page has 16,384 bits.

1) Uniqueness

To test uniqueness, we compared the fingerprint of a page to the fingerprints of the same page on different chips, and recorded their correlation coefficients. A total of 66,240 pairs were compared – (24 chips choose 2) * 24 pages * 10 measurements. The results are shown in Figure 12. The correlation coefficients are very low, with an average of 0.0076. A Gaussian distribution fits the data well, as shown in red.

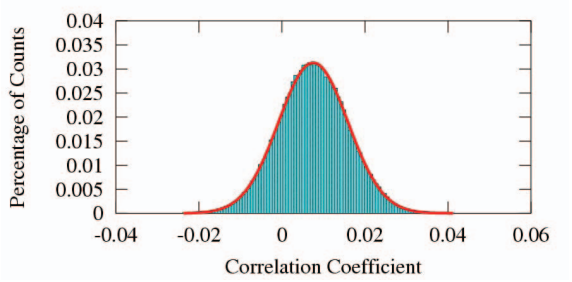

Figure 12. Histogram of correlation coefficients for pages compared to the same page on a different chip (total 66,240 comparisons).

The correlation coefficients are also very low when a page is compared not only to the same page on different chips, but also to different pages on the same and different chips, shown in Figure 13. There are 1,656,000 pairs in comparison – ((24 pages * 24 chips) choose 2) * 10 measurements. This indicates that fingerprints from different parts (pages) of a chip can be considered as two different fingerprints and do not have much correlation. Therefore, the fingerprinting scheme allows the generation of many independent fingerprints from a single chip. The average correlation coefficient in this case is 0.0072.

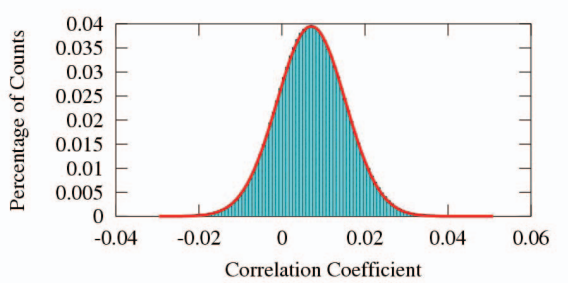

Figure 13. Histogram of correlation coefficients for every page compared to every other page at room temp (total 1,656,000 comparisons).

2) Robustness

To test robustness, we compared each page's measurement to the 9 other measurements of the same page's fingerprint (an intra-chip measurement). The histogram of results for all pages is shown in Figure 14. The correlation coefficient for fingerprints from the same page is very high, with an average of 0.9673. The minimum observed coefficient is 0.9022. The results show that fingerprints from the same page are robust over multiple measurements, and can be easily distinguished from fingerprints of a different chip or page.

To be used in an authentication scheme, one could set a threshold correlation coefficient t. If, when comparing two fingerprints, their correlation coefficient is above t, then the two fingerprints are considered to have come from the same page/chip. If their correlation coefficient is below t, then the fingerprints are assumed to be from different pages/chips.

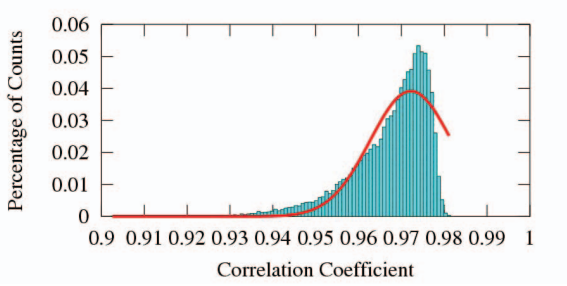

Figure 14. Histogram of correlation coefficients for all intra-chip comparisons (total 25,920 comparisons).

In such a scheme, there is a potential concern for false positives and false negatives. A false negative is defined as comparing fingerprints that are actually from two different pages/chips, but deciding that the fingerprints are from the same page/chip. A false positive occurs when comparing fingerprints from the same page/chip, yet deciding that the fingerprints came from two different pages/chips. The threshold t can be selected to balance false negatives and positives. A high value of t would minimize false negatives, but increase the chance of false positives, and vice versa.

To estimate the chance of false positives and false negatives, we fit normal probability mass distribution functions to the correlation coefficient distribution. A false positive would arise from a comparison of two fingerprints from the same page being below t. The normal distribution fitted to the intra-chip comparison data in Figure 14 has an average $\mu = 0.9722$ and a std. deviation of 0.0095. For a threshold of $t = 0.5$, the normal distribution function estimates the cumulative probability of a pair of fingerprints having a correlation coefficient below 0.5 as 2.62×10^{-539}. At $t = 0.7$, the probability is estimated as 7.43×10^{-181}.

The normal distribution function fitted to the inter-chip comparison data in Figure 13 has a $\mu = 0.0076$ and a std. deviation of 0.0083. The estimated chance of a pair of fingerprints from different chips exceeding $t = 0.5$ is 4.52×10^{-815}. At $t = 0.3$, the probability is estimated as 6.14×10^{-301}.

The tight inter-chip and intra-chip correlations along with low probability estimates for false positives or negatives suggest that the size of fingerprints can possibly be reduced. Instead of using all 16,384 bits in a page, we can generate a fingerprint for a 1024-bit, 512-bit, or even only a 256-bit block. Experiments show that the averages of the observed correlation coefficients remain similar to those when using every bit in a page while the standard deviation increases by a factor of 2-3. However, the worst-case false negative estimates remain low. When using 256 bit fingerprints with the threshold $t = 0.3$, the estimate is 7.91×10^{-7}. Under the same conditions, using 1024 bit fingerprints gives an estimated 3.20×10^{-22} chance of a false negative.

3) Temperature Variations and Aging

To see how robust the fingerprints are across different temperatures. We extracted fingerprints from chips at two other ambient temperatures, 60 °C and -5 °C. We tested a subset of the chips tested at room temperature – 6 pages (3 pages in 2 blocks) in 6 chips.

Of interest is how fingerprints from the same page/chip, but taken at different temperatures, compare. Figure 15 shows the results of the intra-chip comparison between each temperature pair. Correlations remain high for fingerprints from the same page/chip, indicating that fingerprints taken at different temperatures can still be identified as the same. The average correlation coefficient is lower than when compared without a temperature difference, but is still sufficiently high to have very low false positive rates.

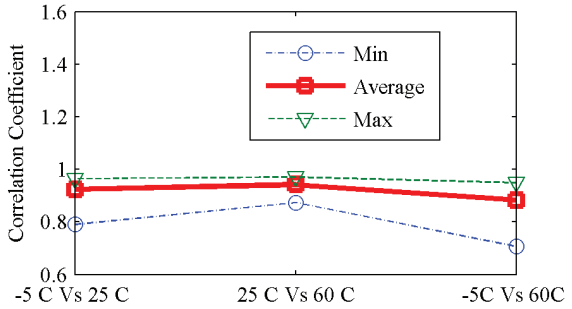

Figure 15. Average, minimum, and maximum correlation coefficients for intra-chip comparisons between different ambient temperatures.

Comparing fingerprints from the same page at the same temperature at -5 °C or 60 °C still yields high correlation coefficients, as expected. Comparisons of fingerprints from different pages/chips at different temperatures give very low correlation coefficients.

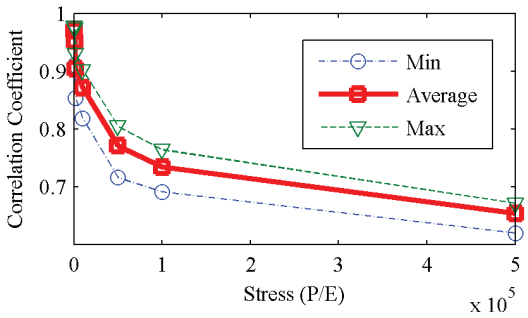

Figure 16. Average, minimum, and maximum correlation coefficients for comparisons between fresh and stressed Flash.

Flash chips have a limited lifetime, wearing out over many program/erase (P/E) cycles. For a page's fingerprint to be useful over time, fingerprints taken later in life should still give high correlation with younger fingerprints. Figure 16 shows the results of comparing fingerprints for the same page/chip taken when a Flash chip is new to fingerprints taken after a different number of P/E cycles. While the average correlation coefficient goes down noticeably, we note that it appears to bend towards an asymptote as the chip wears out. Even after 500,000 P/E cycles, which is beyond the typical lifetime of Flash chips, the average coefficient is still high enough to distinguish fingerprints of the same page/chip from fingerprints acquired from a different page/chip.

However, we found that an extreme wear-out such as 500,000 P/E cycles can raise a non-negligible false positive concern (10^{-4}) for short 256 or 512-bit fingerprints. This result indicates that we need longer fingerprints if they need to be used over a long period of time without a re-calibration.

4) Security

An attacker could attempt to store the fingerprints of a Flash device and replay the fingerprint to convince a verifier that he has the Flash chip in question. If the attacker cannot predict which page(s) or parts of a page (for shorter signatures) will be fingerprinted, he would need to store the fingerprints for every page to ensure success. The Flash chips in our experiments required about 800 partial program cycles per fingerprint. As the fingerprint comprises the order in which the bit was programmed, each bit's ordering could be stored as a 10-bit number. To store an entire chip's fingerprints would require 10x the chip storage.

Acquiring a single fingerprint is relatively fast. Our setup could record an entire page's fingerprint in about 10 seconds. However, there are 131,072 pages on our (relatively small) test chip; characterizing one chip would take about 2 weeks. The characterization time depends on the speed of the Flash interface, and we plan to further investigate the limit on how fast fingerprints can be characterized.

D. Applicability to Multiple Flash Chips

Most of the above experimental results are obtained from the Micron SLC Flash memory. In order to answer the question of whether the proposed techniques are applicable to Flash memory in general, we have repeated both RNG and fingerprinting tests on four types of Flash memory chips in 0, including an MLC chip.

The experiments showed that RNG and fingerprinting both work on all four types of Flash chips, with comparable performance. Detailed results are not included as they do not add new information.

While we found that the proposed algorithm works without any change in most cases, there was one exception where the fingerprinting algorithm needed to be slightly modified in order to compensate for systematic variations for certain manufacturers. For example, for the Hynix and Numonyx chips, we found that bits from the even bytes of a page tend to be programmed quicker than bits from the odd bytes. Similarly, for the MLC chip, bits in a page divide into two groups: a quickly programmed group and a slowly programmed group. To accommodate such systematic behaviors, the fingerprinting algorithm was changed to only compare programming ordering of bits within the same group.

VI. Application Scenarios

This section briefly discusses how the Flash memory based security functions, namely RNGs and device fingerprints, can be used to improve security of electronic devices. We first discuss where the techniques can be deployed and present a few use cases.

A. Applicability

The proposed Flash-based security techniques work with commercial off-the-shelf Flash memory chips using standard interfaces. For example, our prototype design is based on the Open NAND Flash Interface (ONFI) [13], which is used by many major Flash vendors including Intel, Hynix, Micron, and SanDisk. Other Flash vendors such as Samsung and Toshiba also use similar interfaces to their chips.

The proposed techniques can be applied to any Flash or other floating-gate non-volatile memory, as long as one can control read, program (write), and erase operations to specific memory locations (pages and blocks), issue the RESET command and disable internal ECC. Embedded systems typically implement a Flash memory controller in software, exposing the low-level Flash chip interface to a software layer. Our prototype USB board in the evaluation section is an example of such a design. While we did not have a chance to study details, the manual for the TI OMAP processor family [14], which is widely used in mobile phones, indicates that its External Memory Interface (EMI) requires software to control each phase of NAND Flash accesses. In such platforms where Flash accesses are controlled by software, our techniques can be implemented as relatively simple software changes.

For large memory components such as SSDs, the low-level interfaces to Flash memory chips may not be exposed to a system software layer. For example, SSD controllers often implement wear-leveling schemes that move data to a new location on writes. In such devices, the device vendor needs to either expose the Flash interfaces to higher level software or implement the security functions in firmware.

B. Random Number Generation

The Flash-based random number generator (RNG) can either replace or complement software pseudo random number generators in any applications that need sources of randomness. For example, random numbers may be used as nonces in communication protocols to prevent replays or used to generate new cryptographic keys. Effectively, the Flash memory provides the benefits of hardware RNGs for systems without requiring custom RNG circuits. For example, with the proposed technique, low-cost embedded systems such as sensor network nodes can easily generate random numbers from Flash/EEPROM. Similarly, virtual machines on servers can obtain true random numbers even without hardware RNGs.

C. Device Authentication

One application of the Flash device fingerprints is to identify and/or authenticate hardware devices themselves similar to the way that we use biometrics to identify humans.

As an example, let us consider distinguishing genuine Flash memory chips from counterfeits through an untrusted supply chain. Recent articles report multiple incidents of counterfeit Flash devices in practice, such as chips from low-end manufacturers, defective chips, and ones harvested from thrown-away electronics, etc. [5] [15] [16]. The counterfeit chips cause a serious concern for consumers in terms of reliability as well as security; counterfeits may contain malicious functions. Counterfeits also damage the brand name for a manufacturer.

The Flash fingerprints can enable authentication of genuine chips without any additional hardware modifications to today's Flash chips. In a simple protocol, a Flash manufacturer can put an identifier (ID) to a genuine chip (write to a location in Flash memory), generate a fingerprint from the chip, and store the fingerprint in a database along with the ID. To check the authenticity of a Flash chip from a supply chain, a customer can regenerate a fingerprint and query the manufacturer's database to see if it matches the saved fingerprint.

In order to pass the check, a counterfeit chip needs to produce the same fingerprint as a genuine one. Interestingly, unlike simple identifiers and keys stored in memory, device fingerprints based on random manufacturing variations cannot be controlled even when a desired fingerprint is known. For example, even legitimate Flash manufacturers cannot precisely control individual transistor threshold voltages, which we use to generate fingerprints. To produce specific fingerprints, one will need to create a custom chip that stores the fingerprints and emulates Flash responses.

The authentication scheme can be strengthened against emulation attacks by exploiting a large number of bits in

Flash memory. Figure 17 illustrates a modified protocol that utilizes a large number of fingerprints that can be generated from each Flash chip. Here, we consider a Flash chip as a function where a different set of bits that are used to generate a fingerprint is a challenge, and the resulting fingerprint is a response. A device manufacturer, when in possession of a genuine IC, applies randomly chosen challenges to obtain responses. Then, these challenge-response pairs (CRP) are stored in a database for future authentication operations. To check the authenticity of an IC later, a CRP that has been previously recorded but has never been used for a check is selected from the database, and a re-generated response from a device can be checked.

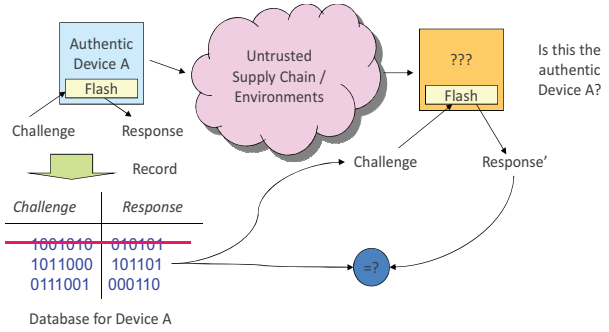

Figure 17. Device authentication through a challenge-response protocol.

Unless an adversary can predict which CRPs will be used for authentication, the adversary needs to measure all (or at least a large fraction) of possible fingerprints from an authentic Flash chip and store them in an emulator. In our prototype board, a generation of all fingerprints from a single page (16K bits) takes about 10 seconds and requires 10 bits of storage for each Flash bit. For a 16Gbit (2 GB) Flash chip, which is a moderate size by today's standards, this implies that fully characterizing the chip will take hundreds of days and 20 GB storage. In the context of counterfeiting, such costs are likely to be high enough to make producing counterfeits economically unattractive.

The security of the authentication scheme based on Flash fingerprints can be further improved if an additional control can be added to the Flash interface. For example, imagine using a USB Flash memory as a two-factor authentication token by updating its firmware to have a challenge-response interface for Flash fingerprints. Given that authentication operations only need to be infrequent, the USB stick can be configured to only allow a query every few seconds. If a fingerprint is based on 1024 Flash bits, fully characterizing an 8 GB USB stick can take tens of years.

D. Cryptographic Keys

In addition to device identification and authentication, the Flash fingerprints can be used as a way to produce many independent secret keys without additional storage. In effect, the proposed Flash fingerprints provide unpredictable and persistent numbers for each device. Previous studies such as fuzzy extractors [17] and Physical Unclonable Functions (PUFs) [3] have shown how symmetric keys (uniformly distributed random numbers) can be obtained from biometric data or IC signatures from manufacturing variations by applying hashing and error correction. The same approach can be applied to Flash fingerprints in order to generate reliable cryptographic keys. A typical Flash with a few GB can potentially produce tens of millions of 128-bit symmetric keys.

VII. Related Work

A. Hardware Random Number Generators

Hardware random number generators generate random numbers from high-entropy sources in the physical world. Theoretically, some random physical processes are completely unpredictable. Therefore, hardware random number generators provide better random numbers in terms of randomness than software based pseudo-random number generators.

Thermal noise and other system level noise are the common entropy sources in recently proposed hardware random number generators. In [18], the phase noise of identical ring oscillators is used as the entropy source. In [19], the differences in path delays are used. In [20] and [21], the metastability of flip-flops or two cross coupled inverters are used. Basically, the entropy source of these RNG designs is thermal noise and circuit operational conditions. These hardware random number generators can usually achieve high throughput because the frequency of the entropy sources is high. One common characteristic of these hardware random generators is that they all need carefully designed circuits where process variations should be minimized so that noises from the entropy source can be dominant. Compared to this, the random number generation in Flash memory cells does not require specially designed circuits and is more immune to process variation. Moreover, our entropy source is based on quantum behavior and theoretically, it should still work under extremely low temperatures where thermal noise or other kinds of noise decrease dramatically.

B. Hardware Fingerprint – Physical Unclonable Funcitons

Instead of conventional authentication based on a secret key and cryptographic computation, researchers have recently proposed to use the inherent variation in physical characteristics of a hardware device for identification and authentication. Process variation in semiconductor foundries is a common source of hardware uniqueness which is out of the control of the designer [22] [23] [24]. A unique fingerprint can be extracted and used to identify the chip, but cannot be used for security applications because it can be simply stored and replayed. We also take advantage of process variation for our fingerprinting scheme.

For security applications, Physical Unclonable Functions (PUFs) have been proposed. A PUF can generate many fingerprints per device by using complex physical systems whose analog characteristics cannot be perfectly replicated. Pappu initially proposed PUFs [25] using light scattering patterns of optically transparent tokens. In silicon, researchers have constructed circuits which, due to random

process variation, emit unique outputs per device. Some silicon PUFs use ring oscillators [26] or race conditions between two identical delay paths [27]. These PUFs are usually implemented as custom circuits on the chip. Recently, PUFs have been implemented without additional circuitry by exploiting metastable elements such as SRAM cells, which have unique value on start-up for each IC instance [28] [4], or in Flash memories [5].

Our authentication scheme requires no new circuitry and can be done with commercially available and ubiquitous Flash chips. Unlike metastable elements, authentication does not require a power cycle. The scheme can generate many fingerprints by using more pages in the Flash chip. Acquiring a fingerprint is also faster and more widely applicable than previous Flash authentication methods.

VIII. Conclusion

In this work, we show that unmodified Flash chips are capable of providing two important security functions: high-quality true random number generation and the provision of many digital fingerprints. Using thermal noise and random telegraph noise, random numbers can be generated at up to 10Kbit per second for each Flash bit and pass all NIST randomness tests. An authentication scheme with fingerprints derived from partial programming of pages on the Flash chip show high robustness and uniqueness. The authentication scheme was tested over 24 pages with 24 different instances of a Flash chip and showed clear separation. A Flash chip can provide many unique fingerprints that remain distinguishable in various temperature and aged conditions. Both random number generation and fingerprint generation require no hardware change to commercial Flash chips. Because Flash chips are ubiquitous, the proposed techniques have a potential to be widely deployed to many existing electronic device though a firmware update or software change.

IX. Acknowledgements

In This work was partially supported by the National Science Foundation grant CNS-0932069, the Air Force Office of Scientific Research grant FA9550-09-1-0131, and an equipment donation from Intel Corporation.

References

[1] S. Yilek and T. Ristenpart, "When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography," in *Proceedings of the 17th Annual Network and Distributed System Security Conference*, 2010.

[2] A. Rukhin, J. Soto and J. Nechvatal, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Crytographic Applications," April 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.

[3] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Proceedings of the 44th Conference on Design Automation,* pp. 9-14, 2007.

[4] P. Koeberl, J. Li, A. Rajan, C. Vishik and W. Wu, "A Practical Device Authentication Scheme Using SRAM PUFs," *Proceedings of the 4th International Conference on Trust and Trustworthy Computing,* pp. 63-77, 2011.

[5] P. Prabhu, A. Akel, L. M. Grupp, W.-K. S. Yu, G. E. Suh, E. Kan and S. Swanson, "Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations," *Proceedings of the 4th International Conference on Trust and Trustworthy Computing,* pp. 1-17, 2011.

[6] M. J. Kirton and M. J. Uren, "Noise in Solid-State Microstructures: A New Perspective on Individual Defects, Interface States and Low-Frequency Noise," *Advances in Physics,* vol. 38, pp. 367-468, 1989.

[7] H. Kurata, K. Otsuga, A. Kotabe, S. Kajiyama, T. Osabe, Y. Sasago, S. Narumi, K. Tokami, S. Kamohara and O. Tsuchiya, "Random Telegraph Signal in Flash Memory: Its Impact on Scaling of Multilevel Flash Memory Beyond the 90-nm Node," *Solid-State Circuits, IEEE Journal of ,* vol. 42, no. 6, pp. 1362 - 1369 , 2007.

[8] C. Compagnoni, M. Ghidotti, A. Lacaita, A. Spinelli and A. Visconti, "Random Telegraph Noise Effect on the Programmed Threshold-Voltage Distribution of Flash Memories," *Electron Device Letters, IEEE ,* vol. 30, no. 9, pp. 984-986, 2009.

[9] S.-M. Joe, J.-H. Yi, S.-K. Park, H. Shin, B.-G. Park, Y. J. Park and J.-H. Lee, "Threshold Voltage Fluctuation by Random Telegraph Noise in Floating Gate nand Flash Memory String," *Electron Devices, IEEE Transactions on ,* vol. 58, no. 1, pp. 67-73, 2011.

[10] D. Knuth, The Art of Computer Programming, Reading: Addison-Wesley, 1968.

[11] T. K. Abe, A. Sugawa and S. Ohmi, "Understanding of Traps Causing Random Telegraph Noise Based on Experimentally Extracted Time Constants and Amplitude," in *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*, Monterey, CA, 2011.

[12] J. H. Scofield, N. Borland and D. M. Fleetwood, "Temperature-independent switching rates for a random telegraph signal in a silicon metal–oxide–semiconductor field-effect transistor at low temperatures," *Applied Physics Letters ,* vol. 76, no. 22, pp. 3248 - 3250 , 2000.

[13] Open NAND Flash Interface, "Open NAND Flash Interface," [Online]. Available: http://onfi.org.

[14] Texas Instruments Incorporated, "OMAP Mobile Processors," [Online]. Available: http://focus.ti.com/general/docs/gencontent.tsp?contentId=46946.

[15] EE Times.com, "U.S.: Fake parts threaten electronic market," 17 February 2010. [Online]. Available: http://www.eetimes.com/electronics-news/4087628/U-S--Fake-parts-threaten-electronic-market.

[16] FrankenFlash Project , "SOSFakeFlash," [Online]. Available: http://sosfakeflash.wordpress.com/.

[17] Y. Dodis, L. Reyzin and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and and Other Noisy Data," *SIAM Journal of Computing,* vol. 38, no. 1, pp. 97-139, 2008.

[18] B. Sunar, W. J. Martin and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," in *IEEE Transactions on Computers*, 2007.

[19] C. W. Odonnell, G. E. Suh and S. Devadas, "PUF-based random number generation," In MIT CSAIL CSG Technical Memo 481, Cambridge, 2004.

[20] M. Majzoobi, F. Koushanfar and S. Devadas, "FPGA-based True Random Number Generation using Circuit Metastability with Adaptive Feedback Control," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2011.

[21] G. Cox, C. Dike and D. J. Johnston, "Intel's Digital Random Number Generator," Hot Chips, 2011.

[22] D. S. Boning and J. E. Chung, "Statistical metrology: Understanding spatial variation in semiconductor manufacturing," in *Proceedings of SPIE 1996 Symposium on Microelectronic Manufacturing*, 1996.

[23] K. A. Bowman, S. G. Duvall and J. D. Meindl, "Impact of die-to-die and within die parameter fluctuations on maximum clock frequency distribution for gigascale integration," *Journal of Solid-State Circuits,* vol. 37, no. 2, pp. 183-190, 2002.

[24] S. R. Nassif, "Modeling and forecasting of manufacturing variations," in *Proceedings of ASP-DAC 2001, Asia and South Pacific Design Automation Conference 2001*, 2001.

[25] R. Pappu, Physical One-Way Functions, PhD Thesis, MIT, 2001.

[26] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the Computer and Communication Security Conference*, New York, 2002.

[27] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication application," in *Proceedings of the Symposium on VLSI Circuits*, 2004.

[28] D. E. Holcomb, W. P. Burleson and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security*, 2007.