

Hiding Information in Flash Memory

Yinglei Wang, Wing-kei Yu, Sarah Q. Xu, Edwin Kan, and G. Edward Suh
School of Electrical and Computer Engineering
Cornell University, Ithaca, NY 14850 USA
 {yw437,wsy5,qx33,eck5,gs272}@cornell.edu

Abstract—This paper introduces a novel information hiding technique for Flash memory. The method hides data within an analog characteristic of Flash, the program time of individual bits. Because the technique uses analog behaviors, normal Flash memory operations are not affected and hidden information is invisible in the data stored in the memory. Even if an attacker checks a Flash chip’s analog characteristics, experimental results indicate that the hidden information is difficult to distinguish from inherent manufacturing variation or normal wear on the device. Moreover, the hidden data can survive erasure of the Flash memory data, and the technique can be used on current Flash chips without hardware changes.

Keywords-flash memory; security; steganography;

I. INTRODUCTION

Flash memory has become a pervasive element in today’s computing landscape. Flash is arguably the most widely used medium for standalone data storage as in USB memory sticks and SD cards. Virtually all mobile devices such as smartphones and tablets rely on Flash memory as their non-volatile storage. Similarly, embedded devices commonly use on-chip Flash memory to store instructions and data. Flash memory is also moving into laptop and desktop computers, replacing mechanical hard drives.

This paper introduces a technique to hide information in analog characteristics of Flash memory in a way that the hidden bits are not visible at all from the viewpoint of normal Flash memory content. More specifically, our technique encodes a hidden bit in the program time of a group of Flash cells; a fast program time encodes bit ‘1’ and a slow program time encodes bit ‘0’. We found that writing 0 into a Flash cell incurs more stress on the cell than writing 1, which in turn results in a larger decrease in the program time of the corresponding cell. While the program time of individual cells cannot be accurately controlled, our experiments demonstrate that bits can be reliably encoded in the program time using many cells collectively.

While a number of steganography techniques have been developed previously, our Flash-based technique provides unique benefits compared to typical digital steganography schemes where information is hidden in another form of digital content such as images and documents. In particular, the hidden information in Flash memory is decoupled from the Flash memory content and instead tied to the physical

object. The following summarizes the main benefits of our scheme compared to digital steganography.

- **Covert:** The proposed technique does not change normal Flash operations or content at all. As a result, inspecting the Flash memory content does not reveal any hidden information. All Flash memory operations can still be performed without any change, even with hidden information. In fact, our experimental results suggest that even analog characteristics of Flash memory such as page program/erase time do not change noticeably.
- **Erase tolerant:** The hidden information in Flash memory remains intact even if the entire Flash memory is erased and programmed with new content. In fact, our experiments show that the hidden information can survive even hundreds of program/erase operations.
- **Copy tolerant:** In typical digital steganography, the cover text with hidden information can be easily copied and stored so that it can be analyzed over time. The hidden information in our technique, however, is tied to physical Flash memory and can only be accessed by measuring the program time of individual memory cells while the Flash memory is in one’s possession. Because modern Flash memory chips often contain tens or hundreds of billions of memory cells, fully characterizing a Flash chip without knowing the location of hidden bits is quite time consuming.

In a sense, the proposed information hiding technique is similar to physical steganography methods where information is hidden in physical objects. For example, people have used secret inks to write messages on blank parts of other messages [4]. However, the proposed technique provides a couple of key benefits over traditional physical steganography methods thanks to being electrical.

- **No hardware modification:** The proposed technique works on unmodified Flash chips using the standard interface. In fact, the technique can be implemented as a software program as long as a low-level Flash interface is exposed.
- **High capacity:** Thanks to the high capacity of Flash memory, our technique provides a fairly high capacity compared to traditional physical steganography techniques. For example, even if we hide one bit for every 512 Flash cells, a 8GB Flash chip can contain 16MB

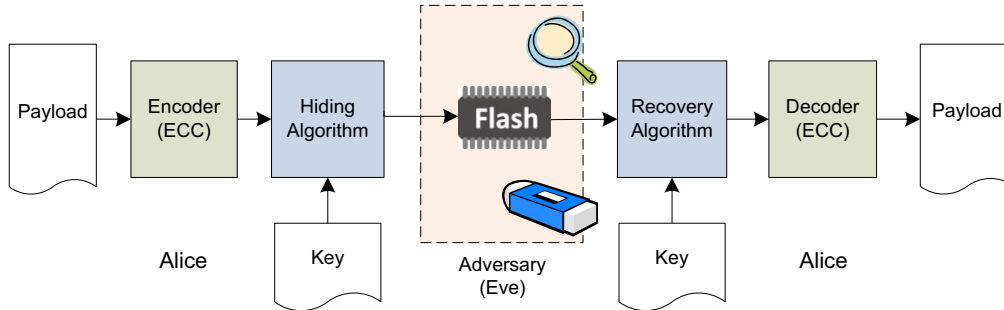


Figure 1. Overview of the information hiding operation.

of hidden information.

Given the ubiquity of Flash memory and the easy applicability of the proposed scheme on commercial Flash chips, we believe that the technique can enable a number of interesting applications. An obvious application of the information hiding in Flash is a secure and covert storage of data. For example, a user can hide sensitive information in the Flash memory of a smartphone with confidence that others cannot retrieve the information even when the phone is lost or stolen. Information hiding provides an additional layer of protection on top of typical encryption by preventing an adversary from reading or even copying the ciphertext.

On the other hand, the capability to covertly communicate may be misused to bypass legitimate access control policies. For example, in the business world, the hidden information in Flash may be misused to export trade secrets. In this sense, this study points out the potential danger.

Another traditional application of information hiding is watermarking. In particular, given that the hidden information is tied to a physical Flash memory chip, the proposed technique can be used to embed watermarking in devices with Flash memory. For example, mobile or embedded devices may be watermarked to help retrieve them when lost or stolen. Similarly, the watermarks can be used to distinguish genuine devices from low-quality counterfeits.

The rest of the paper is organized as follows. Section II discusses the high-level overview of the proposed information hiding approach along with assumptions. Section III provides basic background on the Flash memory. Based on this understanding, Section IV describes the algorithms to hide information in Flash memory and recover it later through standard Flash interfaces. Then, Section V studies the effectiveness and the security of the proposed method through experimental results on real Flash chips. Section VI discusses related work and Section VII concludes the paper.

II. OVERVIEW

A. Threat Model

Figure 1 shows the overview of the information hiding process in Flash memory. In order to hide information in Flash, Alice (left) first adds an error correcting code

(ECC) to her message payload and hides the payload in the analog characteristics in Flash memory. Later, Alice (right) can perform the reverse operations to retrieve the hidden payload by recovering bits from the analog characteristics and correct errors using the ECC. The information hiding and recovery algorithms use a secret key (hiding key) to determine where the hidden bits are stored in Flash memory. As error correcting codes are well studied, this paper focuses on the physical encoding and decoding of information in Flash.

As shown in the figure, an adversary (Eve) gets temporary access to the Flash memory after Alice hides information. We assume that the adversary can inspect and manipulate the memory through its normal interface, but do not consider physical tampering of the memory. In the simple case, the adversary can check normal Flash operations such as program, erase, and read operations. The adversary may also be aware of the information hiding technique and can specifically check analog characteristics of Flash memory that can be observed through the standard interface.

The goal of the adversary may differ depending on the target application. In particular, the adversary may try to

- Detect the existence of hidden information,
- Retrieve the hidden information, or
- Remove the hidden information.

For example, in the traditional steganography context where Alice is trying to establish a covert communication channel, it is important that the adversary cannot easily detect the existence of hidden information. On the other hand, in the context of storing sensitive information, it is more important that the adversary cannot retrieve information without knowing the hiding key. For watermarking, it should be difficult to erase the hidden information.

Given an unlimited amount of time with the Flash chip, an adversary can break the information hiding scheme by trying the retrieval algorithm on all pages with all possible hiding key values because we assume that an adversary knows our hiding algorithm. Therefore, the goal of the hiding technique is to make the detection, retrieval, and removal of hidden information sufficiently time consuming for an attacker.

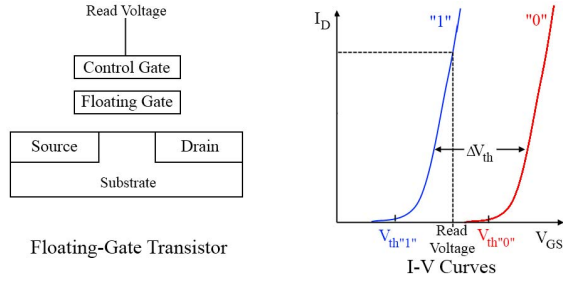


Figure 2. Flash memory cell based on a floating gate transistor.

B. Flash Interface Requirements

The proposed technique is designed to work with 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 (if there is any). For example, our experiments use off-the-shelf Flash chips that use the Open NAND Flash Interface (ONFI) [7], 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. In many embedded and mobile devices, the required interface functions are already exposed to the software layers so that the proposed technique can be simply implemented as a software update.

III. FLASH MEMORY BACKGROUND

This section provides background material on Flash memory and its operating principles to aid understanding of our Flash-based information hiding scheme.

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 2 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.

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 (SLC) store one bit of information per cell by using two threshold voltage levels. Multi-level cells (MLC) store more than one bit by more finely dividing the threshold

voltage levels: for example, four levels can be used to store two bits per cell.

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.

C. Aging

Flash requires high voltages to store and erase information. The voltages involved place great stress on the device oxide; each program operation and each erase operation slightly damages the oxide, wearing out the device. After thousands of program and erase cycles, the oxide could have sustained enough damage to render the bit non-operational, leaving it in a stuck-at state or in a leaky state that cannot reliably hold information over a period of time. Flash is usually guaranteed by the manufacturer up to a certain number of program and erase cycles.

Even before failures, the stress causes the cell’s analog characteristics to change. In particular, the program time that is required to flip a state from ‘1’ to ‘0’ for a cell tends to reduce as the number of program/erase (PE) cycles increases for that cell. We exploit this program time shift in order to hide information.

D. Partial Programming

Our information hiding scheme relies on the measurement of program time, the time it takes to program a Flash cell, at individual cell granularity. However, the standard Flash memory interface requires all bits in a page to be programmed together. Normally, a program operation on a page is held for a long enough time that any cell level variation within a page is overcome. Therefore, the normal program time only reveals how long programming the entire page takes, not how long it takes to program individual bits.

To find the program time on a per-cell basis, we use a technique called “partial programming” [16]. The standard

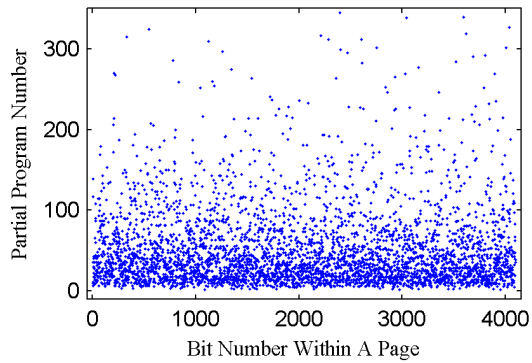


Figure 3. Raw partial program number for each bit in an example page.

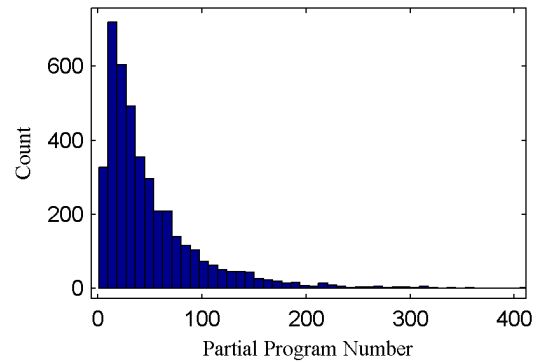


Figure 4. Partial program time distribution for bits in a page.

Flash memory interfaces allow the “partial program” of a cell by aborting a program operation before completion. If the program operation is interrupted, the Flash cell may be in an unreliable state that could be interpreted as 1 or 0. Further “partial programs” will accumulate charge on the floating gate and eventually result in the cell entering a stable programmed state, as if a full program was applied. Effectively, the number of partial program operations to flip a bit from 1 to 0 represents the program time for the bit. In this sense, we use the “partial programming” technique to find program time for individual cells. After a partial program to a page, we read the page and record the state of each bit. When a bit changes to the programmed state (from 1 to 0), we note the number of partial programs required to flip the bit as the bit’s program time.

IV. INFORMATION HIDING ALGORITHM

This section describes the encoding (hiding) and decoding (recovery) algorithms for our information hiding scheme and the rationale for them.

A. Overview

Our scheme hides information in the program time of individual bits of Flash. The program time is the time it takes for a bit to change from the erased state (1) to the programmed state (0). Normally, a Flash memory controller performs a program operation at a page granularity, and the latency of this program operation is determined by the slowest bit in a page to be successfully written. In order to determine the program time for each bit, which we refer to as *per-bit program time*, we use the partial programming technique that is described in the previous section.

Figure 3 shows per-bit program times for a page. The plot shows the number of partial program operations to flip state from 1 to 0 for each bit in a page. Because of process variations, the program time varies widely from bit to bit as shown in the figure. The per-bit program time distribution for the page is shown in Figure 4. The wide distribution and noisy appearance of per-bit program times suggest that small

changes to each bit’s program time would go unnoticed, and could be used to carry a covert payload.

However, in order to hide information using the program time, we need to be able to intentionally change and control each bit’s program time. Interestingly, in this context, previous work has observed that program time tends to decrease as a Flash cell becomes more worn-out [3], [9]. In this work, we also found that how worn-out each bit is can be controlled by selectively stressing a bit. Although one can only program an entire page together, we can stress some bits within a page more than others by controlling the value that we write. During an erase operation, every bit in a page is reset to an erased state (for example, assume that the erased state represents ‘1’). On a program operation, only bits that switch to 0 experience the program stress. When these bits are later erased, they also experience erase stress as they are reverted to the 1 state. Therefore, bits that undergo both switches (1 to 0 and 0 to 1) see the full program and erase stress from one program and erase cycle. However, bits that store 1 will not be switched to the 0 state by a program operation. These bits see much less program and erase stress than their counterparts which are programmed to 0 because their states do not need to change. Therefore, by deciding whether to write a 1 or a 0 to each bit location in a page, we can control which bits are stressed more relative to other bits in the same page.

In theory, if every bit had a similar program time without much variation, we could hide one bit of information in every Flash bit by simply stressing or not stressing the bit so that its program time encodes the hidden bit. However, in practice, the program times of individual bits vary significantly due to manufacturing variations, and intentional stress is often not sufficient to overcome the inherent variations; inherently slow bits will be likely to be still slower than inherently fast bits even after being deliberately stressed. To address this issue, we choose to encode 1 bit of hidden information using many bits in Flash memory. For each bit to hide, we choose a group of Flash bits and program

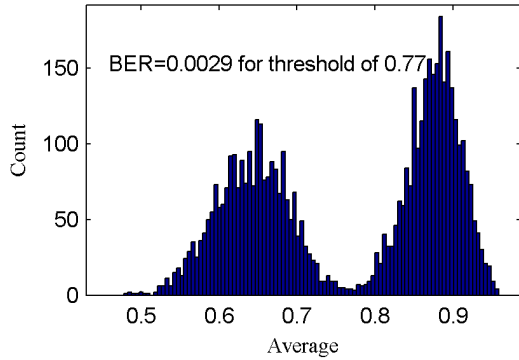


Figure 5. The distribution of the average program time of a group with a correct key.

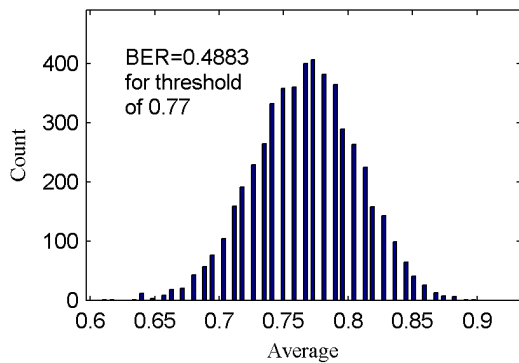


Figure 6. The distribution of the average program time of an incorrect group.

them to the same value, either 1 or 0. Effectively, this process encodes a bit in the collective program time of the group. The averaging effect reduces variations among different groups and allows the hidden bit to be more reliably recovered.

The use of a group also improves the security of the hiding scheme. In our scheme, we use a key (hiding key) to select which Flash bits will be grouped together for each hidden bit. If an attacker does not know the correct key, he or she cannot accurately identify which bits form a group together. Because an incorrect group is likely to contain both more stressed and less stressed bits, the average program time of an incorrect group of bits will not show a clear bias towards either 1 or 0.

For example, Figure 5 shows the distribution of the average program time of a correct group. In the experiment, we randomly selected 5,120 groups, each of which has 128 bits from a page, and hid either 1 or 0. As shown in the figure, there is an obvious gap in the distribution between the fast and slow groups. Therefore, the value of hidden bits can be easily recovered through a simple thresholding.

On the other hand, Figure 6 shows the distribution of the

Algorithm I: Encoding

Part A – Composing the message

```

1 For each selected page in a block
2   Generate the group for each message bit via the page hiding key
3   Assign each group 0 or 1 according to the embedded data
4   For each bit
5     If its group will represent a message "1"
6       Set it to be programmed 0
7     Else
8       Set it to be programmed 1
9   End if
10 End for
11 End for

```

Part B – Writing the message to Flash

```

1 For each selected block
2   For  $i = 1, 2, \dots, N$  ( $N$  is the number of Hiding PE cycles)
3     Erase the block
4     Program every selected page
5   End for
6 End for

```

Figure 7. An algorithm to encode (hide) a payload into Flash memory program time.

average program time when the hiding key is unknown. In this experiment, we used a randomly selected hiding key. As shown in the figure, the average program time of a group shows a normal distribution without any clear separation. This result suggests that it is difficult for an adversary to recover hidden information without correct groupings because each group is likely to have both more and less stressed bits.

B. Hiding Algorithm

Figure 7 describes our methodology for hiding a payload in program time of Flash memory. The algorithm is split into two parts: (A) composing the payload by assigning bits of the message to groups of bits in Flash, and then (B) the actual process of writing the payload to Flash by repeated program and erase stress.

For a given message, we first choose a set of pages and blocks in which to encode the message based on the hiding key and the number bits that need to be hidden. Then, we divide the bits within each page into fixed size groups. Each group is used to store one message bit. The page, block, and group selections are based on the hiding key in a way that cannot be predicted without the key. In our implementation, we used RC4 to choose the Flash bit locations for each message bit.

Then, the algorithm determines which value (0 or 1) needs to be written to each bit location based on the message bit to be encoded. If a group is to store a "1" value, we will program (write a 0) the bits in the group, and the group will experience full program and erase stresses. If a group is to store a "0" value, the bits in the group will be set to 1, and will see less stress.

With the payload mapped to bits in Flash memory, we perform the actual write (program/erase) to Flash (Part B). We decide on a set number of stresses N to exert on the Flash. N is chosen to ensure an acceptable bit error rate without causing excessive stress. Each page is programmed

Algorithm II: Decoding

Part A – Reading the program time from Flash

```

1 For each selected block
2   Erase the block
3   Program every bit in the block to 0
4   Erase the block
5   For each selected page
6     For  $i = 1, 2, \dots, M$ 
7       Partial program the page to 0 (abort a program operation after time  $T$ )
8       Read the page
9       For each bit in the page
10        If the bit changed from 1 to 0
11          Set programtime for this bit to  $i$ 
12        End if
13      End for
14    End for
15  For each bit
16    If the bit did not flip
17      Set its programtime to be  $M + 1$ 
18    End if
19  End for
20 End for
21 Erase the block
22 End for

```

Part B – Extracting the payload message

```

1 For each selected block
2   For each selected page
3     Calculate the median  $X$  of the program times for all the bits
4     For each bit
5       If its programtime  $> (X/2)$ 
6         Set programtime to 1
7       Else
8         Set programtime to 0
9       End if
10    End for
11  Generate the group for each message bit with the page hiding key
12  For each group
13    Calculate the average program time for the group
14    If the average is less than  $Th$ 
15      Recover the message bit: 1
16    Else
17      Recover the message bit: 0
18    End if
19  End for
20 End for
21 End for

```

Figure 8. An algorithm to decode (recover) a payload from Flash memory program time.

N times in order to imprint the payload into the Flash. In our experiments, we found that several hundred to a few thousand PE cycles are sufficient for SLC chips. An even smaller amount of PE cycles are enough for MLC chips.

C. Recovery Algorithm

Figure 8 describes our algorithm to decode a payload hidden by our encoding algorithm in Flash bit program time. Again, the algorithm is divided into two parts: (A) physically reading the per-bit program time from Flash, and (B) recomposing the payload from the program time distribution.

To read the hidden information, we must measure the program times for every bit in the pages containing the hidden bits. To do so, we use the partial programming algorithm described in the previous section. We choose M such that at the end of M partial programs, more than half of the bits, are programmed. The program time of a bit is expressed as the number of partial program cycles needed

to flip the bit from 1 to 0. For the bits that do not flip after the M partial program operations, their program times are set to be a constant above M (i.e. $M + 1$).

To reconstruct the payload from the per-bit program times, we apply two thresholding steps. First, we compute the median program time X across all bits within each page. Then, the program time of each bit within a page is quantized based on the median; if a bit's program time is above half the median program time ($X/2$), then its program time is set to 1; otherwise it is set to 0. ($X/2$) was chosen empirically.

The bits are then divided into the groups specified by the hiding key. Within each group, the average of each individual bit's program times (now consisting of only 1 and 0) is computed, and the second thresholding step is performed. Each bit in the payload is set to 1 if the average program time of the corresponding group is below the threshold Th . Otherwise, the bit is set to 0.

In practice, with sufficient hiding PE cycles, we saw that there exists an obvious gap between the average program times of the more-stressed and less-stressed groups. As a result, it is straightforward to set the threshold Th to distinguish the two types of groups. For each page, we first sort the average program time of each group. Suppose the sequence of sorted program times is $X_0, X_1, X_2, \dots, X_N$. Then we calculate the intervals between the sorted average program times and get $X_1 - X_0, X_2 - X_1, \dots$. Suppose the maximum interval is $X_M - X_L$, then we set the threshold to be in the middle of that interval; $Th = (X_M + X_L)/2$. In this way, we can get a per-page threshold. For the cases with low hiding PE cycles, where there is no clear gap between the two clusters, the threshold is set to be a constant across pages based on the histogram of the average program times from multiple blocks.

For simplicity, we describe and evaluate the algorithm for the case where all bits within a selected page are used to hide bits. In order to make detection more difficult, it is also possible to only use a small subset of bits within a page. We leave this variant for future work.

V. EVALUATION

In this section we evaluate the proposed scheme through experiments on Flash chips. In addition to validating correct operation of the encoding and decoding algorithms, we also study the robustness across various design parameters, performance, detectability, recovery without the hiding key, and erase tolerance.

A. Evaluation Setup

1) *Testbed Device*: Our experiments use a custom Flash test board as shown in Figure 9. The board is made entirely with commercial off-the-shelf (COTS) components with a custom PCB. 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

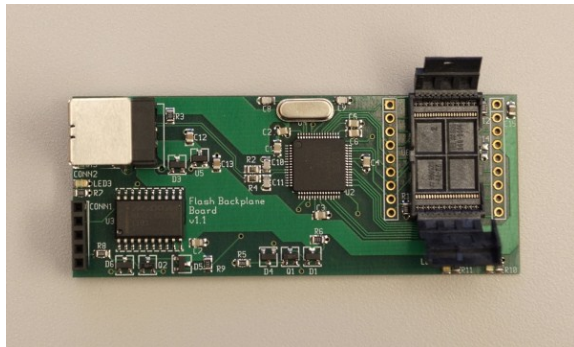


Figure 9. Flash test board.

Manufacturer	Part Number	Size	Qty	Process
Hynix	HY27UF084G2B	4 Gbit	1	5nm class SLC
Micron	MT29F2G08ABA EAWP-IT:E4	2 Gbit	5	34nm SLC
Micron	MT29F4G08ABA DAWP:D	4 Gbit	15	34nm SLC
Micron	MT29F16G08CB ACAWP:C	16 Gbit	1	MLC
Numonyx	NAND04GW 3B2DN6	4 Gbit	1	57nm SLC

Table I
TESTED FLASH CHIPS.

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).

2) *Flash Memory Chips*: The experiments in this paper were performed with five types of Flash memory chips from Numonyx, Micron, and Hynix. Table I shows their details. We primarily performed experiments with Micron 4Gbit chips. Experiments using other models will be marked.

In most experiments, we only used the first 4,096 bits of 16,896-bit pages to avoid performance overheads given the limited amount of memory in the microcontroller. We will refer to the first 4,096 bits as a “page” in the following discussion. For the analyses of per-page read/program time and per-block erase time, we used the entire page.

B. Robustness - Bit Error Rate

In this subsection, we first study whether the proposed scheme can reliably hide and recover bits in the program time characteristics. Here, we use the bit error rate (BER) as the metric for measuring robustness. To measure the BER, we hid a randomly generated message into Flash memory and compared the retrieved message with the original.

In the baseline experiment, we used the first 4,096 bits of a page and divided them into 32 groups (128 bits each) based on a randomly selected hiding key. Then, we selected

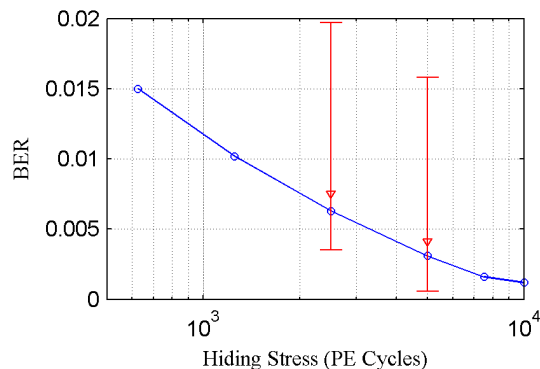


Figure 10. Influence of hiding stress on BER.

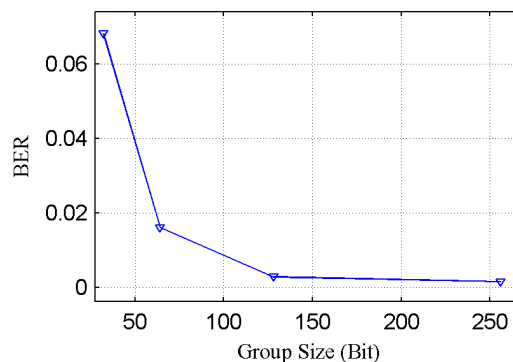


Figure 11. Influence of group size on BER.

multiple pages and blocks across a Flash chip to form 5,120 groups, which represent 5,120 hidden bits, and stored bits using 5,000 program and erase (PE) cycles in the encoding process. In this case, we got a bit error rate (BER) of 0.0029 (0.29%).

Figure 10 shows the BER as a function of hiding stress, which is the number of program/erase (PE) cycles used to stress each group in the hiding process. The blue line shows the average BER using a single Micron 4Gbit chip. For each data point in the figure, the BER is computed over 5,120 bits of hidden information with the group size of 128 bits. For hiding stress levels of 2,500 and 5,000 PE cycles, we also show the statistics across 15 Flash chips; the red triangles show the average BER and the error bars show the maximum and minimum BERs across the 15 chips. We can see that the BER decreases as the hiding stress increases. More stress increases the program time difference between bits hiding 1s and 0s. However, the incremental benefit after 5,000 PE cycles is rather small. Note that the typical lifetime of an SLC Flash chip from the datasheet is 100,000 PE cycles.

There is also a trade-off between the robustness of the scheme and its hiding capacity. When more physical bits are included in a group, the capacity decreases. On the other hand, the statistical variations among groups will decrease as

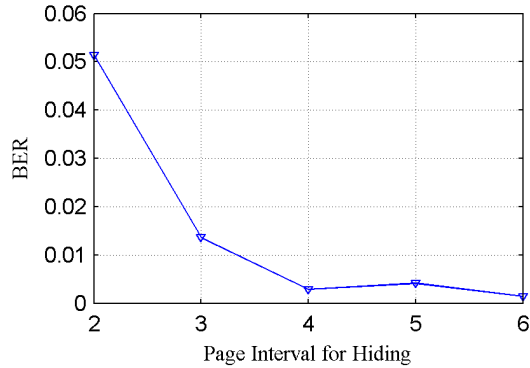


Figure 12. Influence of page interval on BER.

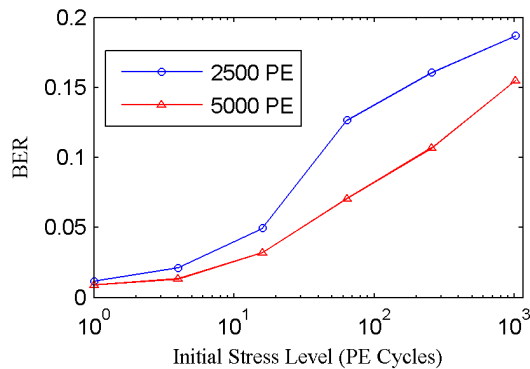


Figure 13. Influence of initial stress level on BER.

the group size increases. Therefore, the BER decreases with an increasing group size, as shown in Figure 11. It is also observed that neighboring pages have a strong influence on each other; stressing one page may also cause some stress in a neighboring page. To solve this problem, only a subset of pages with a specific interval K can be used within a block. If K is 4, then only page 0, page 4, page 8, and so on are used to hide information while the rest is not used. The influence of this page interval on the BER is shown in Figure 12. The experimental results suggest that there is not much benefit to using a group size beyond 128 and a page interval beyond 4 for these chips. Figure 11 and Figure 12 were generated from the 2Gbit Micron chips, but we found that the group size of 128 and page interval of 4 also work well for the 4 Gbit chips.

The effectiveness of the method on moderately used Flash chips is also studied. The influence of the initial stress level before the encoding process on the BER is shown in Figure 13. Here, we aim to simulate the normal usage of the Flash chip. So, in each program operation for the initial stress, random data are programmed. For example, the BER at the initial stress level of 10 PE cycles shows the error rate when bits are hidden after 10 PE cycles of programming random data. It can be observed that as the initial stress

	5,000 Hiding PE	10,000 Hiding PE
BER after zero retention (1 post PE cycle)	0.0029	0.0021
BER after 2-day retention (3 post PE cycles)	0.0141	0.0035
BER after 3-day retention (5 post PE cycles)	0.0187	0.0045
BER after over a month retention (7 post PE cycles)	0.0178	0.0031

Table II
RETENTION CHARACTERISTICS OF THE HIDDEN MESSAGE.

level increases, the BER also increases. However, a higher initial stress level can be tolerated by increasing the stress level in the encoding process. Note that the error rate is still manageable (less than 10-15%) even after hundreds of normal PE cycles.

The retention characteristics of the hiding scheme are shown in Table II. Note that since each decoding performs 2 PE cycles, these retention characteristics include impacts from additional PE cycles in addition to the time between information hiding and retrieval. In the first three rows of Table II, the BER increases as retention time and post-hiding PE cycles increase. In the last row, the BER actually decreases a little compared to the third row. The results suggest that the retention time has little effect on the BER. Intuitively, given that the hiding scheme utilizes cell aging, this result is also supported by the fact that a worn-out Flash memory does not recover greatly even after having been left unattended for a long time.

C. Performance

In our experiments, when a whole page is used for hiding, it takes about 123.6 seconds to perform 5,000 PE cycles of hiding stress on a block, which embeds 2,048 bits of information in the block. The hiding throughput is around 16.6 bits/second. The upper limit of the throughput can also be calculated using the page program time and block erase time given in the Flash memory chip datasheet. The typical page program time is 200 microseconds and the typical block erase time is 700 microseconds. With 2,048 hidden bits in 16 pages of a block, the 5,000 PE cycles will take $(0.2 * 16 + 0.7) * 5,000 / 1,000 = 19.5$ seconds. The throughput will be about 105 bits/second. This is the ideal case which does not include program data transfers and microcontroller overhead. The hiding throughput will also be higher if we use a smaller number of PE cycles for stressing, or if we use smaller groups.

In order to read the hidden information, one needs to obtain per-bit program times using partial programming. The characterization speed depends on the number of partial programs, M , used in the decoding algorithm. For reading hidden bits (decoding), we only need to perform partial programs until more than half of the bits flip. In our experiment, M for decoding is around 30, and it takes around 3.63 seconds to characterize 16 pages, which contain 2,048 hidden bits. Therefore, the read throughput is about

564 bits/second. The read throughput will be higher if the hiding scheme uses a smaller number of Flash bits to encode each hidden bit.

For a detailed analysis to detect hidden bits (see V-D3), one needs to obtain a complete program time distribution with a large M . In our testbed, it takes 612.6 seconds to characterize a block using $M = 1,200$ even if we ignore data transfer from the microcontroller to the host computer and processing time on the host. A 4Gbit Flash memory chip has 4,096 blocks, so obtaining the complete program time distribution of the whole chip will take around 29 days. Higher capacity chips will take even more time to characterize for detection and decoding. For comparison, simply reading the digital content from the 4Gbit Flash chip will take approximately 4 minutes. Therefore, fully characterizing the entire Flash chip without knowing where hidden information is located is quite time consuming.

D. Detectability

The previous subsection shows that the per-bit program time in Flash memory can be controlled sufficiently to reliably store hidden information. Here, we discuss whether an attacker with access to a Flash chip can detect the existence of hidden information. In essence, the question is whether variations in Flash memory characteristics due to information hiding can be distinguished from variations due to normal use.

The proposed information hiding scheme uses per-bit program time, which is not visible from the digital content in a Flash memory device. Also, the hiding operation does not change normal Flash functions; users can still read, erase, and write Flash memory in the expected manner. Therefore, the hidden information cannot be detected from the inspection of digital content. Instead, an attacker needs to rely on checking the analog properties of the Flash memory.

The following list summarizes the steps that an attacker needs to take in order to analyze the analog properties, and in particular, the timing properties, of Flash memory.

- 1) Check for anomalies in timing of normal Flash operations.
- 2) Pick pages/blocks for more detailed analysis.
- 3) Collect per-bit program time for a selected page.
- 4) Analyze the per-bit program time distribution of a page.
- 5) Repeat Steps 2 to 4.

In order to determine whether a Flash chip contains hidden information or not, an attacker can start by checking the timing of normal Flash operations such as per-page program time and per-block erase time, which can easily be obtained from normal operation. If these operations do not show any anomaly – their timing is within the range of timing characteristics for normal use – then the attacker needs to obtain and analyze per-bit program time by picking a page for detailed analysis, collecting per-bit program times

through partial programs, and then running an analysis. If there is no way to identify suspicious pages and blocks from normal operations, in the worst case, the attacker will need to perform the detailed analysis for every single page in Flash memory, which will take a long time.

In the rest of the subsection, we will discuss each step that the attacker needs to take and whether the information that is hidden can be detected in each step.

1) *Anomalies in Normal Flash Operations*: Stressing a Flash chip may affect the analog characteristics of normal memory operations such as page read time, page program time, and block erase time. If these characteristics change significantly due to our scheme, an attacker could use that to detect the existence of hidden information. Therefore, we first study the impact of information hiding and normal Flash use on the page read time, page program time, and the block erase time.

Using the Micron 4Gbit chips, we tested six hiding PE cycle counts (625, 1,250, 2,500, 5,000, 7,500, and 10,000) and five normal PE cycle counts (0, 32, 64, 128, 256) on 4 different chips. On each chip, we used 20 blocks, each containing 64 pages. Because we hide data once every fourth pages, only 16 pages within each block are used to hide information. A normal PE cycle is performed by writing randomly generated data to every page in a block, then erasing that block, simulating wear from normal usage.

To study the impact of information hiding on the page read time, we measured the time to read pages (after performing an erase) when they were fresh as well as after 5,000 hiding PE cycles. The read times were virtually identical before and after the hiding stress, showing that the read time would not be a good indicator for the existence of hidden information.

Figure 14 shows the program times for individual pages in two blocks from one chip, one fresh block and the other with hidden information. As shown in the figure, even though our hiding algorithm only uses every fourth page in a block, there is no visible pattern in per-page program time. The figure also shows that the program time of a page shows distinct values. The distribution between the distinct program times may change as a page wears out with PE cycles. However, we found that the possible program time values for each chip stay the same across the range of stress levels in both normal usage and information hiding cases.

Figure 15 shows the program time distributions across four chips for three different stress levels: fresh, 5,000 hiding PE cycles, and 32 normal PE cycles. The figure again shows that the program time falls into a small set of distinct values even though there are more distinct values across 4 chips. More importantly, pages with and without hidden information share the same set of program time values. Also, unlike per-bit program time, the experimental results show that the page program time does not change significantly with stress, at least for the particular 4Gbit chips that we tested. This is likely due to the fact that the page program

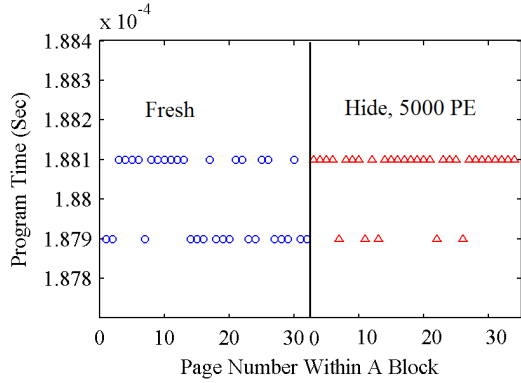


Figure 14. Program time for pages within a block.

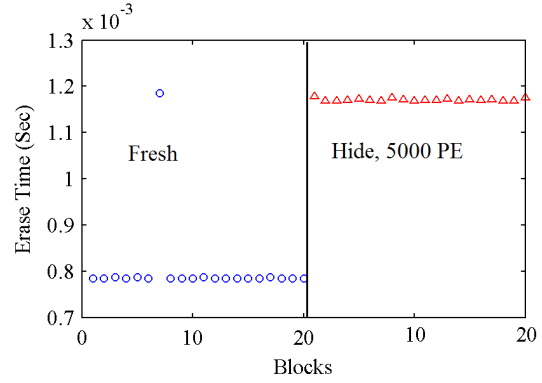


Figure 16. Erase time for 20 blocks within a chip.

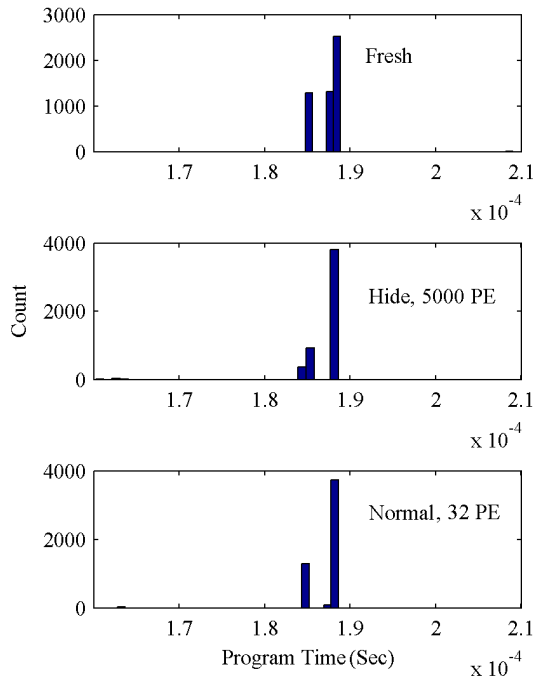


Figure 15. Program time histogram for three stress levels.

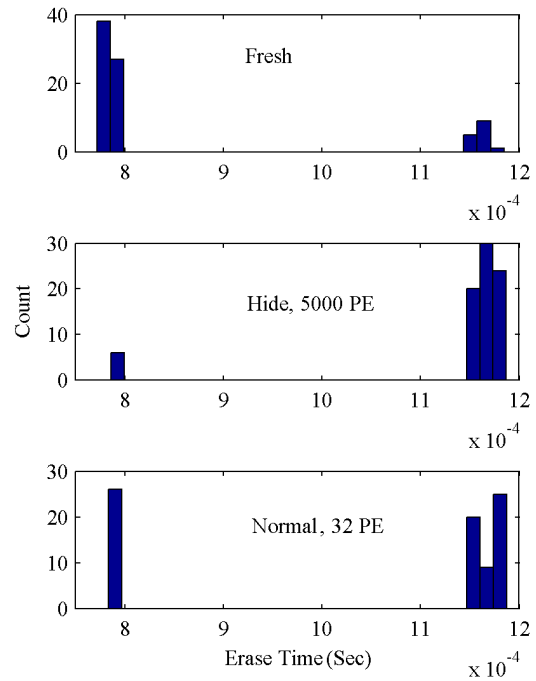


Figure 17. Erase time histogram for three stress levels (across 4 chips).

time is determined by the control circuit based on the slowest bit within a page. Therefore, each page's program time by itself does not show whether the page has hidden information or not.

Figure 16 and Figure 17 illustrate the block erase time distribution within a chip and across 4 chips, respectively. Similar to the program time, the erase time also falls into a few distinct levels, which are common across different stress levels. On the other hand, the figures show that the erase time tends to increase as the stress level increases. As a result, blocks with hiding stress are more likely to have a long erase time compared to fresh block without any stress. In that sense, the erase time may be used to distinguish fresh

pages from blocks with hidden bits. However, because both normal PE cycles and hiding PE cycles increase the erase time, it is unclear how to distinguish blocks with hidden information from blocks with normal PE stress based on the erase time distribution (see Figure 17). We also found that there exist fairly large chip-to-chip variations. For example, some fresh chips may have over 50% of blocks that show a long erase time even without any PE stress.

The experimental results so far show that there is no obvious pattern in program time and erase time distributions to distinguish pages or blocks with hidden information from pages or block with normal PE stress. Yet, it may be possible that there exists a pattern that is difficult to detect in human

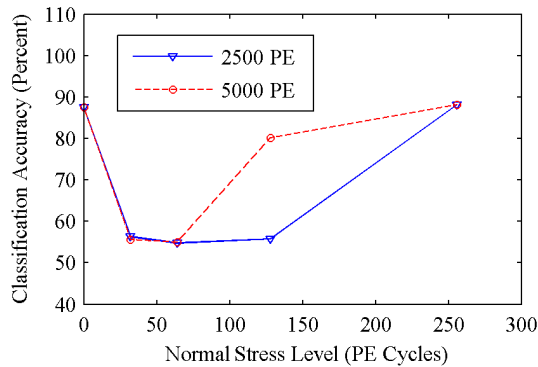


Figure 18. SVM accuracy for detecting hidden information (per-page analysis).

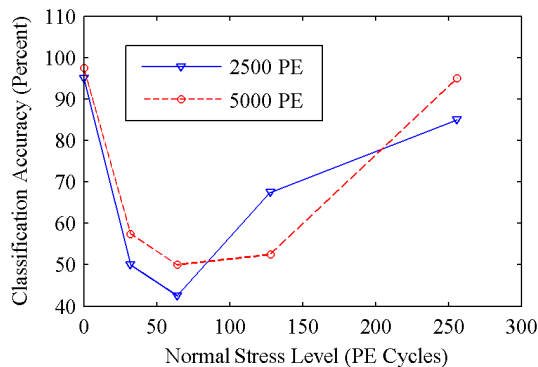


Figure 19. SVM accuracy for detecting hidden information (per-block analysis).

eyes. To further study detectability of hidden information based on normal Flash operations timings, we tried a support vector machine (SVM) to predict whether a page or a block has hidden information. A support vector machine is a machine learning model that is widely used to recognize patterns and classify data sets. We used libsvm, a popular SVM software package [1].

For the SVM experiments, we constructed multiple data sets using pages/blocks with hidden information as well as pages/blocks with normal stress, combining data from one hiding stress level and one normal stress level. We used two hiding stress levels (2,500 and 5,000 PE cycles) and five normal stress levels (0, 32, 64, 128, 256 PE cycles), collected from 4 Flash chips. Then, for each data set, the SVM was trained with data from 3 chips and then tested on data from one remaining chip. This construction represents an idealistic scenario for an attacker. In practice, the attacker will need to consider all possible stress levels for both normal uses as well as hiding, which will add more variations.

Figure 18 shows the prediction accuracy when the SVM is given the program time and erase time for each page

individually without a notion of blocks. The SVM performs relatively well when distinguishing fresh pages and pages with hiding stress. However, the accuracy drops significantly when comparing pages with hiding stress and pages with moderate levels of normal PE stress. In such cases, the accuracy is not much better than random guesses (50%). The SVM performs better again for cases with high normal stress levels beyond 128 PE cycles because the normal stress exceeds the hiding stress level. In essence, the results suggest that the SVM can distinguish pages with different stress levels, but not pages with hiding stress and normal stress.

Figure 19 shows the accuracy of another SVM construction where the input vectors were organized by block. In this design, each input is a vector of program and erase time pairs for each page within a block, allowing the SVM to see a pattern within a block instead of handling each page separately. The goal of this SVM is to identify blocks with hidden information. The accuracy of this SVM was similar to that of the per-page SVM. The SVM could distinguish more stressed blocks from less stressed blocks, but not the hiding stress from the normal stress.

While not shown here, we also tested cases where data from all stress levels were combined together to form a large data set. We found that dealing with multiple stress levels significantly reduces SVM prediction accuracy for both the page-granularity analysis and the block-granularity analysis. The SVM predictions were no better than random guesses.

The experimental results so far show that it is difficult to distinguish pages/blocks with hiding stress from pages/blocks with normal stress even on one particular Flash model (Micron 4Gbit). In practice, an adversary will also need to deal with diversity and variations among multiple Flash manufacturers and models, which will make detecting hidden bits even more difficult.

In fact, we found that analog characteristics of Flash memory varies significantly from model to model. For example, we tested 2Gbit Flash chips from Micron, which have an identical specification with the 4Gbit chips except for the capacity. Surprisingly, the 2Gbit chips, although only a generation apart from the 4Gbit chips, showed a markedly different behavior compared to the 4Gbit chips. For 2Gbit chips, the PE stress had little impact on block erase time while noticeably changing page program time. In essence, the 2Gbit chips showed the opposite type of behavior as the 4Gbit chips where the erase time shows a significant shift. In both cases, we still found that it is difficult to distinguish the impact of hiding stress from that of normal stress.

The significant variations across Flash models imply that an attacker will need to build and train an SVM model for each Flash chip model in order to use the SVM for determining the existence of hidden data on a particular chip. Obviously, this would require a significant investment on the part of the attacker. Even then, as we have shown above, there is no guarantee that an SVM model using normal Flash

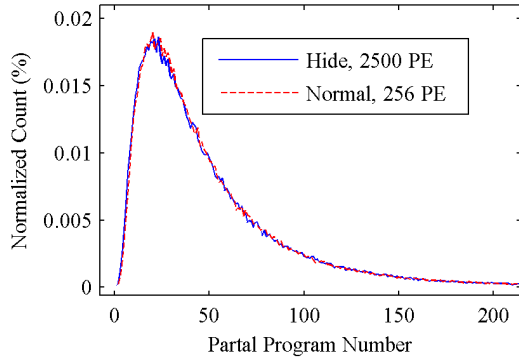


Figure 20. Partial program number distribution curve averaged over 5 blocks.

operations will be able to determine the existence of hidden data with a high probability.

2) *Page Selection and Per-Bit Program Time Collection:*

The study of normal Flash operations shows that an adversary cannot simply determine whether a Flash chip has hidden information or not based on measurements of normal Flash operation times. In essence, the hiding stress cannot be effectively distinguished from normal PE stress. As a result, an attacker needs to perform a more detailed analysis on per-bit program times in an attempt to determine the existence of hidden data, which we will discuss next.

To perform the detailed analysis of each page, the attacker will have to characterize each page. However, characterizing per-bit program time for every page is quite a time-consuming process. As discussed in Section V-C, a 4 Gbit Flash memory chip requires around 29 days to characterize. For larger chips, which are common today, the per-bit characterization will take even longer.

To avoid expensive characterization of every page, an attacker may be able to use normal Flash operation times to select candidate pages for the detailed analysis. For example, for the 4Gbit Micron chips, an attacker may consider blocks with a higher erase time to be more likely to have hidden information. However, the study in the previous subsection suggests that pages and blocks with hiding stress can be hidden by stressing other blocks on the chip with a moderate number of normal PE cycles.

3) *Per-Bit Program Time Analysis:* A more detailed detectability analysis involves analyzing the partial program time distribution for bits within a page. In normal usage, the bits are programmed 0s and 1s randomly over time. In the hiding scheme, some bits are always programmed 0s and others are always programmed 1s. However, the hiding scheme does not cause an obvious bimodal distribution due to large intrinsic variations of bits in a page. Figure 20 shows the partial program time distribution averaged over 5 blocks. It can be seen that they are very similar to each other.

To statistically analyze the distributions, we turned to

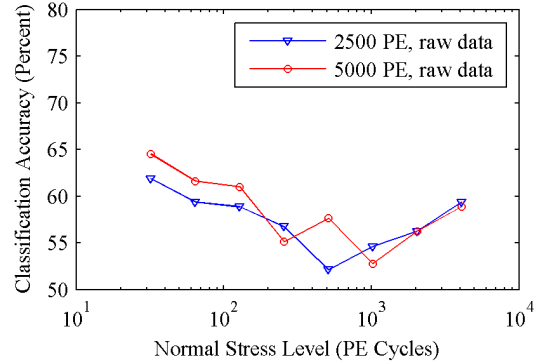


Figure 21. SVM accuracy for detecting pages with hidden information (using raw data).

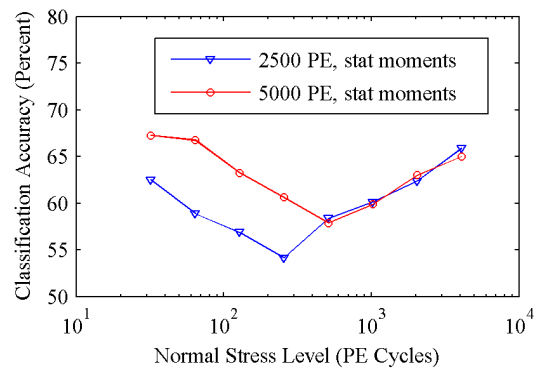


Figure 22. SVM accuracy for detecting pages with hidden information (using statistical moments).

support vector machines again. To train an SVM for the per-bit analysis, we prepared pages across 2 different hiding PE stress levels (2,500 and 5,000) and 8 different normal wear stress levels (32, 64, 128, 256, 512, 1,024, 2,048, and 4,096 PE cycles). We used 5 blocks on each chip, 16 pages per block, for a total of 80 pages per chip, at each stress level; i.e. on one chip, there are 80 pages with a hidden message stressed at 2,500 hiding PE cycles, 80 pages with a hidden message stressed at 5,000 hiding PE cycles, 80 pages without hidden data stressed 32 normal PE cycles, and so on. We characterized pages across 15 different chips. Each page represents a data point in the SVM. The SVM had access to the complete raw data for each page: the vector representing a page and an entry for each bit, with the entry's value as the partial program time.

We then grouped the data from all chips into multiple sets, combining one hiding stress level and one normal stress level. For example, one data set comprises the hidden data with 2,500 hiding PE cycles and the data with 128 normal PE cycles, another data set used 5,000 PE hidden data and 4,096 normal PE cycles, and so on, with a data set for each combination of hiding and normal PE cycles.

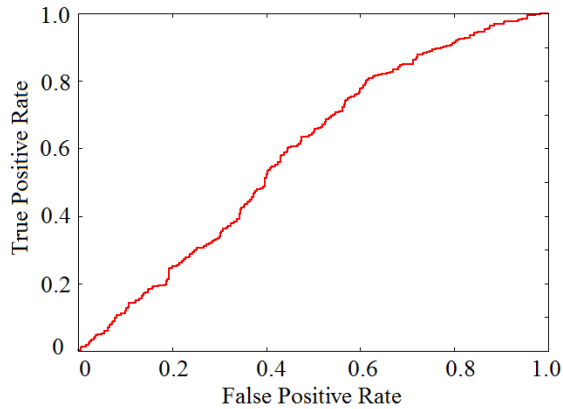


Figure 23. Receiver operating characteristic curve for data set including 2500 hiding PE and 128 normal PE stresses.

For each data set we labeled the hidden pages and non-hidden pages appropriately, trained the SVM with data from chips 1-10, and then used the resulting SVM to predict data from chips 11-15. Overall prediction accuracy of the SVM on test data from chips 11-15 is shown in Figure 21 and Figure 22.

Each data set is represented by a point in Figure 21. Normal PE stress level is shown on the X-axis. The data sets sharing 2,500 hiding PE stress are connected by a solid line; the data sets sharing 5,000 hiding PE stress are connected by a dashed line. Accuracy is shown on the Y-axis.

Overall accuracy is slightly better than random (50%) for all data sets, with increased accuracy near the extremes of normal PE stress cycles. This matches the expectation that a given page with a certain hiding PE stress level looks similar to a page with a certain normal PE stress level. The further the normal PE stress level varies from the matching hidden PE stress level, accuracy should increase.

The data sets in Figure 22 show the SVM accuracy using a different representation for page characteristics. Instead of using the partial program count for every single bit in a page, a page was summarized by several statistical parameters: minimum, maximum, average, variance, skew, and kurtosis. We can see that prediction accuracy is similar to the SVM using the raw bit-level data.

Figure 23 shows a more detailed analysis of the SVM accuracy using the data set for 2,500 hiding and 128 normal stresses levels. The receiver operating characteristic (ROC) curve plots the true positive rate versus the false positive rate, and gives an indication of how accurate the SVM prediction is, for a given false positive rate. The graph shows that the SVM prediction cannot achieve a high true positive rate without incurring a large percentage of false positives.

We also note that detecting hidden information is likely to be even more difficult in practice. For example, the hiding scheme may only use a subset of a page instead of every

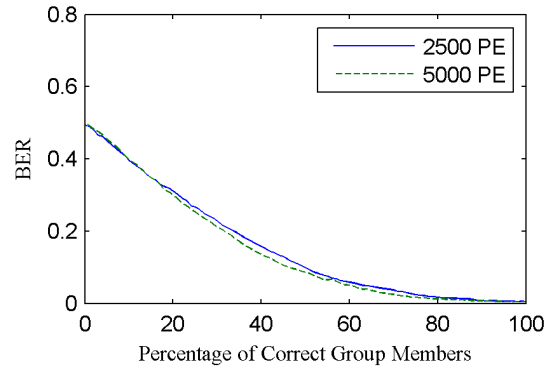


Figure 24. BER as a function of the percentage of correct group members.

bit. Also, a classifier such as an SVM will need to deal with multiple stress levels together. We found that SVM accuracy is lower when a data set contains multiple stress levels.

E. Retrieval without the Hiding Key

Without the hiding key, one can still attempt to extract the hidden information. By estimating (through random guessing if necessary) which bits are grouped together, an attempt at extraction could reveal data if enough of the estimate is correct. Figure 24 shows the bit error rate versus the percentage of correctly guessed group bits.

With a large enough group and page size, it is difficult to correctly guess enough of the group members. For our group size of 128, the probability that 10% (13) of the bits in a randomly selected group of 128 bits belong to the desired group is approximately $\binom{128}{13} * (1/32)^{13}$, or 0.5%. As there are 32 groups of 128 bits in a 4,096 bit page, each bit has a 1/32 chance of being in the desired group. Even at 10%, the bit error rate is approximately 0.4. The chance of guessing 20% of the bits in a randomly selected group drops precipitously; it is 7.3e-11%. In addition, an attacker would have to try several group sizes.

Group size is a security parameter that one can adjust in order to provide greater or lesser protection against brute force group selection.

F. Erase Tolerance

To test the erase tolerance of the scheme, we deliberately stress the chip after hiding information on the chip. For this post-hiding stress, we program every bit of the page to 0, in order to put the maximum stress on the bits. The influence of post-hiding stress on the BER versus the number of PE cycles performed after hiding information is shown in Figure 25. From the figure, we can see that the BER increases as the post PE stress level increases. However, the BER of hidden information is quite reasonable, even after hundreds of post PE cycles. For example, with 5,000 hiding PE cycles, the BER is less than 10% even after 500 post-hiding stress cycles.

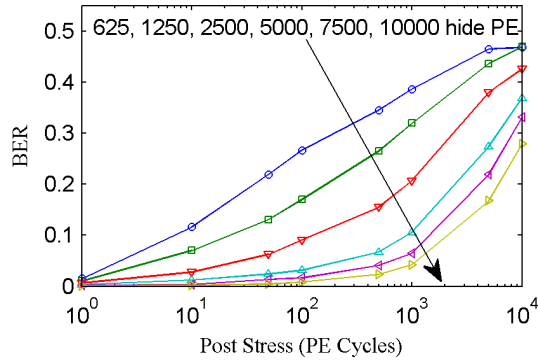


Figure 25. Influence of post hiding PE cycles.

G. Different Flash Models

To ensure that our scheme applies more generally, we tested several different Flash memory models (shown in Table I). On all of the chips, we were able to successfully hide and recover information. We noticed that chips from the same manufacturer tend to perform similarly. For the Micron 2Gbit chips, 5 chips are tested using 10,000 hiding PE stress and 128-bit groups. The mean BER for these five chips is 0.0030. The maximum BER and minimum BER are 0.0041 and 0.0016, respectively. Chips from different manufacturers perform differently. The tested Hynix chip has a similar BER, 0.0021, as the Micron chips in the same experiment. However, for the Hynix chips, page 0 is different from other pages in a block and, in the decoding process, a different threshold Th is needed to convert the average program time into the final binary bit for this page. The tested Numonyx chip has a very large gap for the group averages with the correct hiding key, making its BER 0 in our experiment.

We also included a multi-level cell (MLC) chip in our testing, as these chips are commonly used. MLC chips map multiple bits to each memory cell. As a result, one needs to know the mapping of bits to Flash cells to selectively stress certain cells. For the Micron MLC chip we tested, we only used the upper page in a pair of pages (as specified from the datasheet). We programmed 0 to the bits which we want to stress and 1 to the rest of the bits. Then, we programmed all of the bits to 1. Interestingly, we found that bits within a page split into a fast group and a slow group in this MLC chip, and only the faster programming bits worked for information hiding. The MLC chip required significantly fewer PE cycles to achieve the same level of BER compared to the SLC chips. For example, we used 2,000 PE cycles for our experiments and got a BER of zero – there was a large gap between the more stressed and less stressed groups.

VI. RELATED WORK

This section briefly summarizes prior work in steganography technologies and hardware security functions, and

discusses how they are related to the information hiding technique in this paper.

A. Steganography

With the advent of information technology, digital steganography has become the subject of considerable study.

A large body of work has focused on hiding information within digital files, such as images, videos, audio files, text, and others [2], [10], [8]. These schemes usually hide data in unused meta-data fields, or by exploiting noise in the digital content itself; i.e. altering colors slightly in an image or frequency components in an audio file. In all cases the hidden data is tied to the data in the digital file. A recent proposal [5] takes a different approach: using the fragmentation pattern of digital files in a file system as a covert channel, avoiding tampering with the digital content itself. However, hidden data is still innately tied to the existence of a digital file. Also, modifying hard drive firmware has been investigated as a potential way to hide information [14]. Data is hidden in sectors marked as unusable at the firmware level (instead of the OS or filesystem level), which renders the sectors inaccessible to most software and complicates recovery, as it is difficult to tell legitimately bad sectors from ones used for hiding.

Our proposed scheme for Flash memory shares the concept of exploiting noise to hide data, in the sense that intentionally created biases are hidden in inherent variations in Flash program time. However, unlike the above methods, in which hidden information depends upon plainly visible digital files, our information hiding scheme uses analog properties of Flash. As a result, hidden information is decoupled from the digital content and instead tied to a physical object. The use of physical properties makes detecting, copying, or erasing of hidden information difficult because it requires detailed and time-consuming analog measurements.

Some steganographic techniques hide information where it is not encoded in plainly visible digital files. For example, there exist methods to hide information in the noise of wireless and optical transmissions by modifying the physical layer protocol [6], [15], [11]. Our work presents a new way to hide information in Flash memory. Unlike previous techniques, which often require special tools or modifications to existing protocols, the proposed information hiding technique can be applied to Flash memory chip through a standard interface without any hardware modification.

To make the steganographic functions available in the embedded domain, Stanescu et al. proposed to use an FPGA to efficiently process steganographic algorithms [12]. Our technique gives embedded platforms the ability to hide info within the device at a level not visible to the file system, and requires no additional hardware, as Flash memory is common on embedded platforms.

B. Flash Based Security

We hide a message in the per-bit program times of Flash memory. Given the popularity of Flash memory in computing systems, there have been studies on analog characteristics of Flash memory [3]. While we have gained insight from the previous work, it primarily focuses on using analog variations to build more efficient computing systems rather than enhancing security.

Recently, there have been proposals to use noise and variations in Flash memory for security by generating true random numbers and unique chip fingerprints [9], [16]. We use the partial programming technique that was proposed by the previous study. However, this paper proposes a completely new application of Flash memory in the context of information hiding instead of random number generation and fingerprinting.

C. Physical Unclonable Functions

Physical Unclonable Functions (PUFs) exploit process variation to provide unique fingerprints for logic circuits [13]. Special circuits are built that vary their output depending on the process variation specific to one instance of the chip. This work is related to PUFs in the sense that we exploit physical properties and process variations for security purposes. However, unlike PUFs, our information hiding scheme uses process variations to hide information instead of generating device-specific fingerprints and keys. Also, our information hiding technique can be applied using standard Flash chips and does not require any custom circuitry.

VII. CONCLUSION

In this paper, we demonstrate a technique to hide information using the program time of individual bits in Flash memory. Program time is an analog characteristic of Flash and is not visible from digital content, does not affect normal memory operation, and survives Flash data erasure. Measuring program time can be done over the standard Flash interface (with no hardware modification) via partial programming. Using groups of bits to store one bit of payload allows the technique to effectively hide information robustly with low bit error rates, and makes detection difficult to prove unless one knows the hiding key. Without the key, measuring analog characteristics of the Flash chip reveals nothing that cannot be explained by normal wear or manufacturing variation. We note that retaining a copy of the entire analog characteristics of the Flash memory requires a large amount of time.

VIII. ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation grants CNS-0932069 and CNS-1223955, the Army Research Office grant W911NF-11-1-0082, and an equipment donation from Intel Corporation.

REFERENCES

- [1] C.-C. Chang and C.-J. Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [2] A. Cheddad, J. Condell, K. Curran, and P. M. Kevitt. Digital Image Steganography: Survey and Analysis of Current Methods. *Signal Processing*, 2010.
- [3] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *Proc. of the 42nd International Symposium on Microarchitecture*, 2009.
- [4] N. Johnson and S. Jajodia. Exploring Steganography: Seeing the Unseen. *Computer*, 1998.
- [5] H. Khan, M. Javed, S. A. Khayam, and F. Mirza. Designing a Cluster-Based Covert Channel to Evade Disk Investigation and Forensics. *Computers & Security*, 30(1):35 – 49, 2011.
- [6] A. Mehta, S. Lanzisera, and K. Pister. Steganography in 802.15.4 Wireless Communication. In *Proc. of the 2nd International Symposium on Advanced Networks and Telecommunication Systems*, 2008.
- [7] Open NAND Flash Interface. <http://onfi.org>.
- [8] F. Petitcolas, R. Anderson, and M. Kuhn. Information Hiding - A Survey. *Proc. of the IEEE*, 1999.
- [9] P. Prabhu, A. Akeel, L. Grupp, W. Yu, G. E. Suh, E. Kan, and S. Swanson. Extracting Device Fingerprints from Flash Memory by Exploiting Physical Variations. In *Trust and Trustworthy Computing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011.
- [10] N. Provos and P. Honeyman. Hide and Seek: An Introduction to Steganography. *IEEE Security & Privacy*, 2003.
- [11] P. Prucnal, M. Fok, K. Kravtsov, and Z. Wang. Optical Steganography for Data Hiding in Optical Networks. In *Proc. of the 16th International Conference on Digital Signal Processing*, 2009.
- [12] D. Stanescu, V. Stangaciu, I. Ghergulescu, and M. Stratulat. Steganography on Embedded Devices. In *Proc. of the 5th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2009.
- [13] G. E. Suh and S. Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proc. of the 44th Design Automation Conference*, 2007.
- [14] I. Sutherland, G. Davies, and A. Blyth. Malware and Steganography in Hard Disk Firmware. *Journal in Computer Virology*, 2011.
- [15] K. Szczypiorski and W. Mazurczyk. Steganography in IEEE 802.11 OFDM Symbols. *Security and Communication Networks*, 2011.
- [16] Y. Wang, W. Yu, G. E. Suh, and E. Kan. Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints. In *Proc. of the IEEE Symposium on Security and Privacy*, 2012.