Stash in a Flash

Aviad Zuck[1], Yue Li[2], Jehoshua Bruck[2], Donald E. Porter[3], and Dan Tsafrir[1,4]
[1]Technion–Israel Institute of Technology [2]California Institute of Technology
[3]University of North Carolina at Chapel Hill [4]VMware Research Group
{aviadzuc,dan}@cs.technion.ac.il, {yli,bruck}@caltech.edu, porter@cs.unc.edu

Abstract

Encryption is a useful tool to protect data confidentiality. Yet it is still challenging to hide the very presence of encrypted, secret data from a powerful adversary. This paper presents a new technique to hide data in flash by manipulating the voltage level of pseudo-randomly-selected flash cells to encode two bits (rather than one) in the cell. In this model, we have one "public" bit interpreted using an SLC-style encoding, and extract a private bit using an MLC-style encoding. The locations of cells that encode hidden data is based on a secret key known only to the hiding user.

Intuitively, this technique requires that the voltage level in a cell encoding data must be (1) not statistically distinguishable from a cell only storing public data, and (2) the user must be able to reliably read the hidden data from this cell. Our key insight is that there is a wide enough variation in the range of voltage levels in a typical flash device to obscure the presence of fine-grained changes to a small fraction of the cells, and that the variation is wide enough to support reliably re-reading hidden data. We demonstrate that our hidden data and underlying voltage manipulations go undetected by support vector machine based supervised learning which performs similarly to a random guess. The error rates of our scheme are low enough that the data is recoverable months after being stored. Compared to prior work, our technique provides 24x and 50x higher encoding and decoding throughput and doubles the capacity, while being 37x more power efficient.

1 Introduction

The ability to successfully hide data is becoming increasingly important for modern computer users, who often store private and sensitive data on their personal devices. These devices are often stolen or misplaced, jeopardizing confidentiality of sensitive data [1–5]. Although encryption can hide data contents, encryption alone cannot hide the presence of encrypted data. Over time, flaws in encryption techniques can be discovered. Moreover, law enforcement agencies, intelligence agencies, and other potent adversaries are increasingly capable of forcing users to submit the decryption keys or passphrases for their devices [6–9]. Thus, for highly-sensitive data, there is value in hiding the very *presence* of the data.

Commercial forces also drive the need to hide small amounts of data within larger data sets. Economic espionage [10] is forcing companies to find ways to protect and safely circulate sensitive data. Hidden data can also be used to identify copyright infringement, using techniques such as digital watermarking [11]. Hardware validation and fingerprinting is also gaining traction as manufacturers seek cheap and efficient ways to validate products and authenticate their components so they cannot be copied and faked [12, 13]. Thus, both privacy and commercial concerns drive the need for additional data hiding tools, both for users and corporations.

This paper presents a new approach for hiding sensitive data within a larger data set on a NAND flash device. This larger data set can be public, or encrypted with a standard encrypted storage system, like Bitlocker [14] or FileVault [15]; we refer to this larger set as public data for brevity. Within this public data set, our technique encodes hidden data using small manipulations of voltage levels in a subset of the flash cells storing public data.

This paper focuses on NAND flash memory, for both practical and technical reasons. On the practical side, flash is ubiquitous in embedded systems, mobile phones, USB thumb drives, and in solid-state disks (SSDs) on personal laptops—precisely the type of devices that are most likely to be lost, stolen, or confiscated. SSDs are also significant in data centers and servers, which could also be the subject of search or seizure.

From a technical perspective, flash is well-suited for data hiding because it offers high-density, fast random access, and non-volatile storage, but with an abundance of internal randomness [16] that is typically masked by on-device firmware. Internally, flash stores data by electrically charging arrays of floating gate transistors/memory cells to a predefined voltage. To read the data back, the stored voltage levels are coarsely discretized into a one or a zero. This discretization process is noisy—the voltage levels across cells in the device vary widely. Even within one device, the charge levels in flash cells have a high variance, attributable to the inherent noisiness of the programming process, variations created in the manufacturing process, and voltage interference inherent to flash cell transistor technology (see §4). Because the flash programming process is impre-

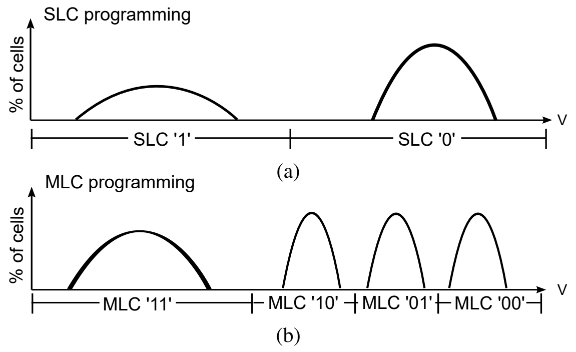


Figure 1: Typical voltage level distributions of cells in SLC (a), and MLC (b) flash memories. Leftmost curves are for programmed cells in the erased state, which are negatively charged. MLC distributions are typically narrower.

cise, flash manufacturers face a trade-off between programming time and storage density, as well as reduced lifetime [17, 18]. The opportunity we see is that there is enough natural variation to hide data in a typical flash array without leaving telltale statistical anomalies— even with an attacker powerful enough to measure the voltage level or other physical characteristics of each cell and run data analysis tools on the voltage level distributions.

The trade-off between write time and precision in flash encoding is well-known, and we leverage this in our design. By taking multiple fine-grain charging and sensing steps, one can more precisely and gradually increase the voltage to a desired level [19]. Single-level cell (SLC) flash can store only one bit selected from one of two voltage levels, whereas multi-level cell (MLC) flash uses four voltage levels and can store two bits, three-level cell (TLC) flash uses eight voltage levels, etc. [20]. Fig. 1 illustrates typical cell voltage distributions for SLC, and MLC. Devices commonly transition cells between SLC and MLC/TLC mode dynamically [21–30]. In other words, the number of bits stored in any given cell can be changed dynamically within a wider range than is commonly used—the only differences are that writing more bits is slower and one needs to know how to interpret the voltage levels of the cell when it is re-read.

In this work we store hidden data by transparently increasing the densities of select flash cells, but without creating a detectable deviation in the overall cell voltage distribution. In our model, a user can access hidden data according to normal methods; the user can hide data with a secret key, that selects certain cells to program with finer-grained variation in the voltage level. Thus, an important part of this work is measuring the expected variance in a faster and coarser charging process (e.g., SLC), and then ensuring that the result of a finer-grained charging process is within this distribution.

Our data hiding scheme, called VolTage-HIde (VT-HI), selects a small number of cells to store an extra bit, from a a larger field of cells not storing hidden data. VT-HI uses a slower charging process to more precisely charge selected cells to a voltage range that represents the logical state of a public and a hidden bit (e.g., converting from SLC to MLC). The cells not selected for hiding data are programmed using standard, widely-available programming operations to store normal data. Public data in VT-HI is assumed to be encrypted with one key, and a second key is used to locate and decrypt hidden data.

The closest related work to ours (Program-Time-HIde, or PT-HI), hides data in flash memory by encoding hidden bits using the different programming times of groups of cells. VT-HI, on the other hand, directly stores data in flash cells, by mimicking the incremental storage technique internally employed by flash vendors. Our straightforward approach has several advantages:

- Encoding is 24x faster in VT-HI and 37x more energy-efficient.
- Decoding of hidden data requires a single, non-destructive, read operation. This makes the decoding process 50x faster. Hidden data can also be read multiple times, while maintaining the integrity of public data.
- Copying hidden data without knowledge of the relevant secret key is impossible, while erasing hidden data (e.g., when in fear of device confiscation) is almost instantaneous.
- The generic nature of VT-HI makes it applicable to multiple chip models from different vendors.

VT-HI is feasible in existing flash-based devices without any hardware modifications, although firmware support would be helpful. For current devices, we approximate the required firmware support on real devices using a sequence of partial programming (PP) [16] operations, where a normal program operation is aborted midway. Using this method, the level of additional charge stored in a cell is roughly correlated with the relative time that the program operation is executed before being aborted. We note that PP steps require only standard flash interface commands [31] (i.e., PROGRAM and RESET).

Hidden data is read using a vendor-specific command that shifts the reference threshold voltage for reading. This command is used in modern flash chips by all vendors to measure voltage distributions and to improve retention [32–35]. Storing and reading public data in VT-HI requires only standard flash operations (e.g., PROGRAM and READ) in order to read data in coarse-grain voltage ranges. Notably, over time flash technology increasingly supports reading in ever finer granularities (e.g., up to four bits per cell [36, 37]).

We evaluate the effectiveness of VT-HI by measuring several issues:

1. **Does VT-HI detectably perturb the voltage levels on the device?** Using the methodology in prior data hiding work [38], we find that, under the most favorable circumstances, a Support-Vector Machine (SVM) can only achieve 50–53% accuracy, or *roughly equivalent to random*.
2. **Does VT-HI encode data faster than the current state of the art technique?** VT-HI is 24x faster and 37x more energy-efficient than PT-HI, the closest related work.
3. **Does VT-HI induce faster wear on the device?** Yes, writing hidden data amplifies writes to hidden cells by a factor of ten; this is an order-of-magnitude reduction compared to the state of the art (PT-HI requires 625). This also only applies to the small fraction of cells storing hidden data.
4. **What is the capacity of VT-HI?** Our implementation uses about 0.02% of the bits to hide data on unmodified devices; with firmware support, this could be increased to 0.2%, or double the capacity of the current state-of-the-art.

In total, these results indicate that the naturally-occurring variability in a flash device creates enough noise to form a useful substrate for data hiding techniques. As part of a larger steganographic system or watermarking system, VT-HI has the particular advantage of creating a variable number of bits; a long-standing challenge for data hiding systems is that the number of bits on a device or in a file is a zero-sum game. Moreover, although the building blocks for VT-HI are not exported to users by most flash vendors, this paper makes the case that VT-HI would be feasible in current flash controllers or firmware.

2 Related Work

Exploiting the Noisiness of Flash to Hide Data. The closest related work to ours is PT-HI [38], which creates a covert channel from the programming time of flash cells. PT-HI applies several hundreds-to-thousands of normal programming cycles to groups of cells, which in turn lengthens the programming time of some cells. Hidden data is encoded based on which cells are slower or faster to program. In other words, the technique creates subtle yet hard-to-detect variations in programming times of each group. A particular advantage of this design, not present in our proposed design, is that these variations persist even if co-located public data persists.

A particular disadvantage of PT-HI is performance: both writing hidden data and reading it requires between dozens, up to hundreds, of programming steps. Decoding in PT-HI is not only time consuming but also a destructive process that destroys any public data stored on the device, and reduces the device's overall lifetime. In addition, the error rate of the hidden payload significantly increases after only a few hundred public data Program/Erase Cycles (PEC), severely limiting the number of times a user can store hidden as well as public data on the device. When combined, these limitations potentially disqualify PT-HI as a building block for a long-lived, steganographic SSD.

Low-level variation in flash has also been used to create a unique fingerprint of flash-based devices [16, 39]. Such fingerprints can be used to authenticate a device's origin. Others suggested to use flash for approximate storage [40].

Hiding Information through Steganography. Our work continues the theme of past research in the field of steganography. Embedding hidden data unto digital objects such as image, audio, and video files is typically achieved by applying small unnoticeable distortions [41–43], abusing existing transmission protocols [44, 45], or in a visible transmission channel [46–49]. A common theme is using inherent noisiness to disguise data hidden within the noise. These solutions often face challenges with mutable data, as data like photographs are typically not expected to change.

Steganographic file systems [50–55] hide data in locations known only to the user, using a hash function on a file name and password. Plausible deniability solutions masquerade hidden data as random content visibly stored alongside regular content [56, 57].

A key limitation of many steganographic file systems is that the total number of bits is fixed. Any bits that are not available to the file system are potential telltale signs of hidden data, and require alternative explanations, like free space, that can fail to hold up if an attacker takes multiple snapshots of the device. Flash firmware can thwart such traditional solutions by leaving multiple copies of data on the device. Several works proposed to solve these and other problems on flash-based devices by openly inserting random-content, undecryptable blocks to the system as part of the system's normal operation [58–60]. However, such solutions still give away the steganographic nature of the system, which may void any claim for the user's innocence for some potent attackers (e.g., intelligence officer in an authoritative regime).

Thus, an advantage of our proposed solution (VT-HI) and PT-HI as building block for a steganographic solution is that they can create hidden bits of storage that do not necessarily reveal the presence of hidden data on the device. In our proposed work (VT-HI) in particular, changes to cells that store both hidden and public data can be excused as routine firmware maintenance (§9.2).

3 NAND Flash Background

NAND flash memories store data using floating gate (FG) cells [61]. Flash packages are divided into blocks,

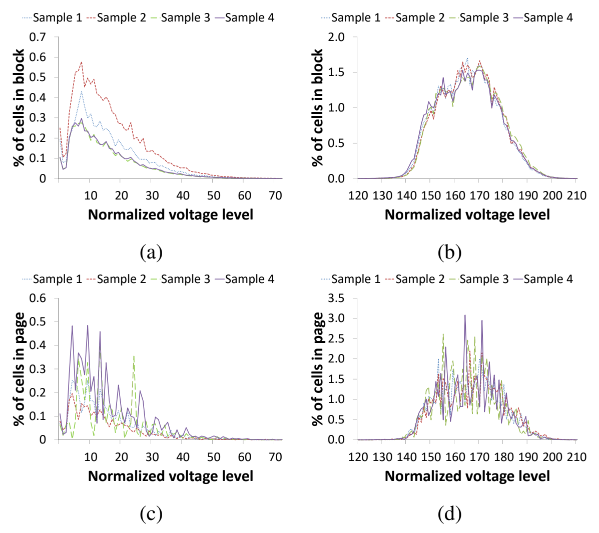


Figure 2: Voltage level distributions of charged cells in four sample 1x-nm MLC chips of the same model. Distributions exhibit significant noisiness at the block level for both non-programmed (a) and programmed (b) cells. (c) and (d) show the distributions at the page level, which exhibit even greater noisiness.

typically 256–2048 KB in size. Blocks are further divided into pages, typically 4–16 KB in size. Pages are stored on physical wordlines, which are serially connected FG cells. When data is written, the cells are electrically charged using small incremental charging steps to a predefined voltage, which traps electrons in the floating gate. The logical value of a cell is read by comparing its voltage to predefined reference threshold voltages placed between relevant voltage intervals. When the flash memory is in MLC/TLC mode, the same cell stores several logical bits by comparing to multiple, smaller voltage intervals. In such cases, several logical pages are stored in a single physical wordline.

An important constraint of flash memories is the lack of support for in-place updates. Once a cell is charged, its level of voltage can only be increased [62, 63]. Voltage is only lowered with an erase operation, which is applied at the granularity of a block (256–2048 KB). Blocks in modern MLC chips can typically endure up to 3K Program/Erase Cycles (PEC). Thus, most SSD vendors include a flash translation layer (FTL), which dynamically remaps logical addresses onto different physical pages [61]; this indirection facilitates rewriting data onto new blocks, garbage collecting old versions of data, and migrating "cold" data onto new blocks for erasure and wear leveling.

4 Flash Variability

The basis for this work is that variability in voltage level distributions of flash cells can be used to hide data. This section gives the reader a sense of the typical range and sources of variation, using measurements from a sample flash chip. The next section explains how we leverage this variability for data hiding.

The inherent variability of flash manifests in three ways relevant to our goals, described and characterized in prior work [16, 35, 64–66]. First, there is significant noise in the programming process. Second, the variability in the chip manufacturing process creates noticeable, naturally-occurring differences in the cell voltage distributions from different NAND flash samples, even from the same vendor, batch and chip model. Finally, there are significant variations in the Bit Error Rate (BER) of different hardware units. VT-HI leverages this inherent noisiness of the charge levels in flash cells, by applying tiny manipulations within the margin of naturally-occurring variations.

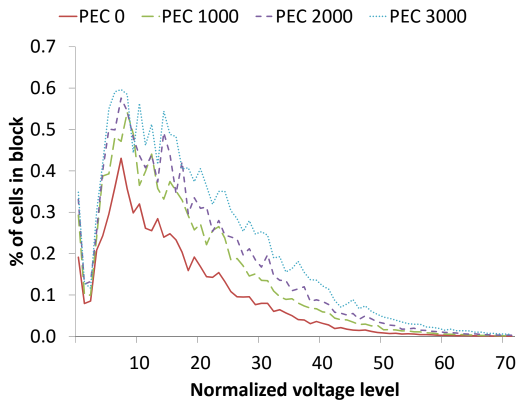
We measure the range of these variations in a representative 1x-nm NAND flash memory model from a major vendor (not listed because of an NDA, see §6.2 for details) , using the following procedure. First, we programmed pseudorandom data to select blocks from four flash chip samples from the same model, and measured the cell voltage distributions for each sample [35,67,68]. On each run, a new random data pattern was used. We repeated this process for 0 to 3000 PEC.

Figure 2 shows some[1] of the voltage distributions of the non-programmed/erased cell state and the full distribution of a programmed state (used to represent data bits "1" and "0", respectively) measured from four blocks (Figures 2a and 2b) and four pages (Figures 2c and 2d), each from a different sample that carries the same number of PEC. We note that 99.99% of cells are concentrated between levels [0, 70] and [120, 210], for non-programmed and programmed cells respectively. Notably, these are essentially SLC distributions. For more fine-grain distributions, such as MLC, TLC, and QLC, the voltage ranges are narrower [17, 32].

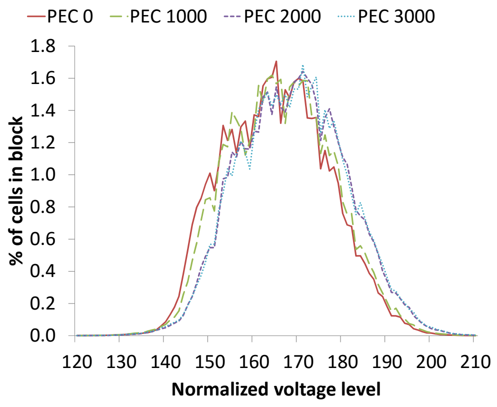
Figures 2a and 2c demonstrate a known phenomenon where non-programmed cells become partially charged due to interference from programming nearby cells [69].

In Figure 2, the long tails and general width of these curves indicate a wide range of valid voltage levels, and the nonsmoothness of the voltage distributions indicates that a uniformly random bit pattern does not generate uniform distributions of voltage levels. At the page-level the variability is even greater, due to disturbances from neighboring pages, and from having a smaller sample relative to blocks. Furthermore, there are noticeable variations in the distributions of different samples. Note

[1]The current NAND flash interface only allows measurement of positive voltage (V) in discrete normalized units (0-255 in this model), indicating that the programming process is noisy. Therefore, the distributions of erased cells that have negative voltage were not measured.



(a)



(b)

Figure 3: Voltage level distributions tend to shift to the right over the lifetime of cells. The figures show distributions for (a) non-programmed and (b) programmed cells with increasing PEC.

that our measurements were taken from blocks in different physical areas of the same chip.

Figure 3 illustrates variation in voltage levels due to aging. The figure shows the block-level voltage distributions in a flash sample after different numbers of program/erase cycles (PEC). As cells with higher PEC are more easily overprogrammed, their voltage distributions tend to have higher means compared to those of cells with lower PEC.

Finally, we measured variation in BER across hardware units in the same package, normalized to the same PEC count. Commensurate with the other results, and prior studies [65, 66], variations in BER of programmed data in flash exist regardless of PEC (as well as an expected increase in BER as PEC increases).

The measurements in this section establish the range of expected voltage levels in a flash device that is programmed with encrypted data, which should roughly appear as a uniformly-random bit pattern. VT-HI stores hidden data with a special, additional flash programming pass. If the overall voltage distribution stays indistinguishable from measurements on the same chip, there will be no telltale anomalies on the device that would indicate additional data is hidden in those cells. This section indicates that there is a wide berth for reliably hiding data within flash voltage levels.

5 Hiding Data

In this section, we describe how users utilize VT-HI to hide data, the relevant threat model and specific VT-HI techniques for a user to hide data on a flash chip; the data flow of VT-HI is illustrated in Figure 4. .

5.1 Usage Overview

Given a flash device, we model the problem as two users, normal user (NU) and hiding user (HU). These can also

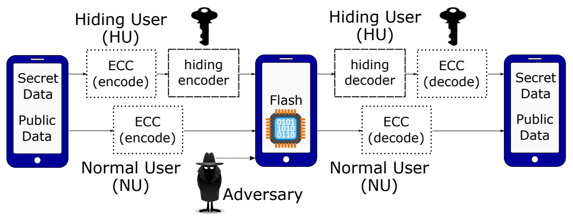


Figure 4: Flow of hiding data on flash in a mobile device.

be thought of as two "modes" or "roles" for the same human user, such as writing to a day planner in normal mode (as NU), but editing sensitive data in hidden mode (as HU). The NU wants to store her public data in flash memory. The HU wants to hide her data inside the data of the NU on the same device, and provides a private, secret, key to VT-HI, which determines the locations in the normal data device where the HU's data will be hidden as extra hidden bits in the chosen cells.

The NU need not be aware of any private keys to correctly read her data. With the secret key, HU's data can also be located and read, without altering the state of public data. Special care must be taken to avoid destroying HU data when the public NU data containing it is migrated or invalidated. The HU must either re-embed the hidden data in a new location (e.g., a page containing newly written NU data), before the old NU page containing it is permanently erased, or apply redundancy a scheme (e.g., parity encoding) to provide some protection for hidden data.

5.2 Threat Model

We assume an adversary who does not know the secret key used to select cells containing hidden data, but has access to the flash device and the capabilities to write and read flash as well as to probe the voltage levels of every cell. We assume that the adversary only gains access

Algorithm 1: Encoding algorithm for VT-HI. The main loop is repeated *m* times.

```
1 VT-HI ($Page, Key, P, H, Vth$);
  Input: Flash page number, secret key, two sets of
         bits to store, and a threshold voltage. $P$ is
         public data and $H$ is hidden data.
2 Use $PRNG(Key, Page)$ to select $|H|$
    non-programmed public bit offsets to store
    hidden bits;
3 Program $P$ to $Page$;
4 Encrypt $H$ using $Key$ and apply ECC;
5 repeat
6     Read cell voltage levels in $Page$;
7     Partial program all hidden “0” bits with
        $Voltage < Vth$;
8 until all hidden “0” bits have $Voltage \geq Vth$;
```

to the device after the hidden payload was stored (see Figure 4). We further assume that the device is spyware-free and that the adversary cannot compare snapshots of the device state over time (we discuss multiple snapshot adversaries in §9). Probing cell voltage levels is widely supported by modern NAND flash memories [35,67,68], and was used as a tool for NAND characterization in this work. An adversary who suspects that the user is hiding data with our technique, can try to detect the existence of such data, as indicated by unexpected charge distributions in a subset of cells. We assume that the VT-HI capability is either added and removed at will by the user or is omnipresent (see §9), and therefore does not raise suspicion in itself.

However, even with perfect knowledge of charge distributions and the exact configuration parameters of VT-HI (e.g., hidden bits per page), there will be no telltale aberrations in the voltage levels indicating the presence of hidden data. In other words, judging by state of the art indicators [38] (see §7) we show it is equally plausible that a given device does or does not hold hidden data.

Finally, we assume that flash block wear in the device is not entirely equal, as is the case in many flash wear leveling policies [70–72].

5.3 Hiding Techniques

We now describe the data hiding algorithm in detail.

Normal data and hidden data are stored by two separate programming passes. The normal data is first programmed into a flash page, using standard flash operations. The hidden data will be programmed to the same pages in a second programming pass. First, a subset of the cells in a given page are selected to store hidden data, then a second encoding pass is done to store the hidden bits. Algorithm 1, as well as the following text, describe our encoding process.

Hidden cell selection. To select cells to store hidden data, we use a pseudo-random number generator (PRNG), such as SHA-256, that produces a set of random numbers based on a key—in our case, a key known only to the HU. We note that the HU does not explicitly persist the location of cells containing hidden data, but rather uses a deterministic PRNG function to calculate the map during boot time. In order to ensure an equal distribution of bit values, VT-HI encrypts hidden data, not unlike standard SSD controller data scrambling [32].

We only select non-programmed (i.e., “1”) bits from the public data in a page to store hidden data. We remind the reader that flash cells typically use low voltage levels to store a “1”, and raise the voltage to store a “0”. We found that it is easier to reliably make small adjustments to the voltage levels of non-programmed cells than programmed cells; we believe that a flash vendor could use either type of cell in a production prototype.

In selecting a cell, the PRNG gives a page-dependent offset, such as the 3rd non-programmed bit in a specific flash page (e.g., by combining the secret key with the page number). This bit is then selected to be programmed with hidden data.

In order to store Error Correcting Codes (ECC) to tolerate bit errors, we select more cells for hidden data than the bits we wish to write.

We note that this technique spreads wear from extra programming evenly across cells over time, as which physical cell is programmed or not will vary over time, as will the output of the PRNG. We further assume public data is encrypted and bit values will be uniformly distributed. In practice, one could adopt more general wear-leveling techniques for hidden data if needed.

Storing Hidden Data. Figure 5 illustrates voltage-level encoding for hidden data in VT-HI. It starts by showing the voltage level distributions for a non-programmed (“1”) cell: any voltage level less than about 127 is considered a public “1”. Anything higher is a “0”. We hide data by selecting a cut-off for hidden values of about 34, which is where most public voltages naturally occur.

To program a hidden “0”, one must use a series of up to *m* partial programming (PP) steps, until the voltage is comfortably above the hidden data threshold. This PP process programs hidden data cells in an intermediate voltage level by iteratively reading and minutely incrementing the voltage level until the target threshold is reached. As with MLC or TLC flash, writes with this iterative technique are slower, but more precise.

A hidden “1” is not programmed. In the small chance that a cell should store a “1” but happens to be above the threshold, we treat this as a bit error and rely on ECC to

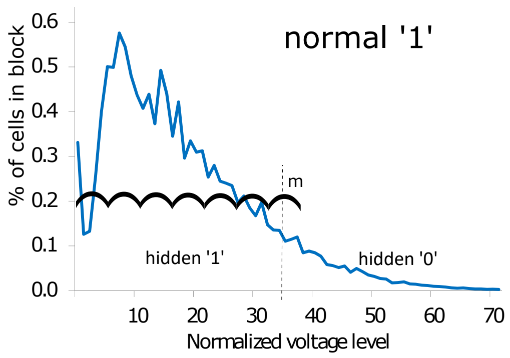


Figure 5: VT-HI hides data in the voltage level distribution of non-programmed cells, which store a normal '1'.

recover the data.

An important property of this design is that public data can be read with no awareness of hidden data or private key. This is because cells that store hidden data stay within the expected voltage levels for the public bit. To read hidden bits, the HU uses her key to calculate the indices of cells holding hidden bits, and reads them using the reference threshold voltage, which is placed in the middle of the two voltage intervals of the hidden bit states.

6 Implementation

In this section we describe the implementation of VT-HI on real hardware. We describe the hardware platform used in our experiments, and determine the configuration parameters for our hiding technique. Finally, we explain how the capacity of VT-HI can be extended through vendor support.

6.1 Experimental Hardware

Any implementation of our technique involves chip-specific configurations. To test them we used the same 1x-nm planar MLC flash used in Section §4. Each flash package has 8GB total storage capacity and contains 2048 blocks. A block consists of 128 lower pages and 128 upper pages with page size of 18048 bytes. The samples have a specified lifetime of 3000 PEC. Read, write (program), and erase latencies are 90 us, 1200 us, and 5 ms, respectively; the energy required for each operation is 50 uJ, 68 uJ, and 190 uJ, respectively.

The flash packages were operated using a commercial NAND flash tester [73]. Voltage level characterization of cells as well as the hiding algorithm were implemented as host software on a PC, which communicates with the tester via a USB interface. Throughout this section our calculations do not take into account data transfer and hardware overheads, which would be considerably lower on a production deployment. The specific voltage threshold (level 34) used for implementing our technique was determined empirically to tolerate the overshooting/underprogramming errors caused by the imprecise PP operation. We also verified that the total number of cells in the range is larger than the total number of hidden bits.

6.2 Vendor Support

Flash vendors are notoriously secretive about the internals of their devices. In order to collect the data presented in this paper, some co-authors of this paper signed a non-disclosure agreement (NDA) with a flash vendor. The NDA prohibits disclosing which vendor or the specific chips. In exchange we were given enough information to use a non-public command on the chip to measure voltage levels of cells, as well as issue partial programming (PP) commands to specific cells. To the best of our knowledge, the operations we use are generally implemented on any flash device, but the particular command encoding details vary from chip to chip, and are not made public. The NDA does not prohibit release of this data.

In principle, our prototype represents the most that a user could accomplish via reverse engineering a flash device, or using a flash device that openly published all available commands. Our results indicate the feasibility of the idea, with no changes to the flash controller. In the rest of this subsection, we explain how a few simple changes to a flash controller or the FTL firmware would improve the results we report.

First and foremost, PP is less precise than a program command issued by the controller. This is also the reason we select only non-programmed cells to store hidden data; PP is too coarse to reliably make fine-grained changes to programmed cells. We believe that an in-controller implementation of voltage hiding could likely program hidden data in fewer programming steps, saving energy and wear on the device, and opening up data hiding in both programmed and non-programmed cells.

Another feature not available to us was the ability to dynamically adjust voltage thresholds and targets [21–26]. The ability to control voltage targets and the width of voltage intervals might improve our hiding technique since narrower voltage intervals have been shown to easily fit into wider programming intervals [74] (e.g., TLC in MLC). This feature is generally available to the controller internally.

A limitation resulting from the lack of a more precise programming mechanism and the inability to adjust target voltage levels is that we found it difficult to reliably hide data in MLC or TLC modes using partial programming. We expect that a flash controller can extend our ideas to MLC or TLC, but the PP command on our test device was too coarse for this experiment to correctly store hidden data, and tended to disrupt public bits. Recall that a goal of our design is that one can read public data without any awareness of private, hidden data. Our measurements indicate that, with more precise programming steps and/or the ability to adjust voltage thresholds slightly, our approach should extend to MLC or TLC. We note that existing flash page architectures regularly use a second fine-grained programming pass that does not significantly add interference to flash cells, and is this less detrimental to the bit rate as PP steps [32, 69]

6.3 Determining Capacity

In this subsection, we explore the potential capacity of our suggested hiding scheme, i.e., how many hidden bits we can store using VT-HI. This is a function of several concerns: over-provisioning bits to correct for errors (i.e., ECC), ensuring that the overall distribution of voltage values is not significantly perturbed (ensuring hidden data remains hidden), and minimizing the risk of inducing errors on neighboring cells or pages.

In order to keep the space overhead for ECC low, when determining configuration parameters for VT-HI we attempt to minimize the standard metric of bit-error rate (BER). We measure BER by encoding a hidden message in multiple blocks, physically located in different areas of the same chip. The message contains random content, both to emulate an encrypted hidden message, and to ensure that the charge levels in cells storing hidden bits have no anomalous effect on the overall distribution. After decoding, the message is compared with the original to determine the resulting error rate.

To find the optimal method and parameter values that minimize the hidden data BER (i.e., improve the effective data capacity) without compromising security, we systematically investigated each possible combination of three key parameters: number of partial programming steps, number of hidden bits per page, and page interval. The number of steps can be taken as a rough upper bound on write performance—fewer steps means faster hidden data writes, but more steps may be required to ensure a target voltage is reached. In setting hidden bits per page, intuitively, adding more hidden bits will push the overall distribution of voltage levels higher. Page interval is the physical distance between two cells storing hidden data; when a cell in one page is partially programmed, it may cause interference on neighboring pages. Intuitively, partially programming too many adjacent cells can cause bit flips on nearby public cells; thus, we measure this risk as a function of average physical distance of hidden bits. Our experiments were performed on a fresh chip, to avoid any interference that might stem from previous write patterns and wear. For each combination of parameters we encoded hidden data in five different blocks, and measured the average hidden data BER after each PP step.

Figure 6 shows that after roughly ten PP steps the BER converges to less than 1%. This trend holds regardless of the number of hidden bits or the page interval.

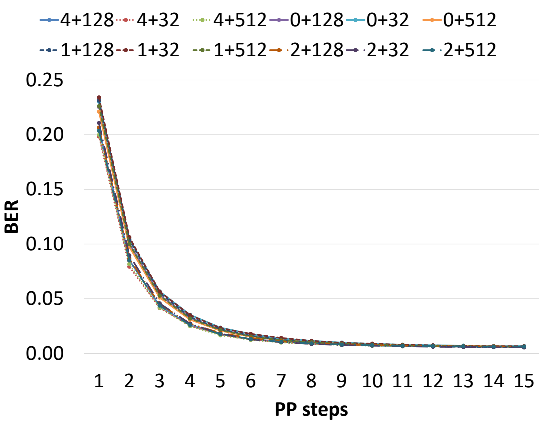


Figure 6: Hidden BER rates for VT-HI for the first fifteen steps in multiple combinations of page intervals and number of hidden bits.

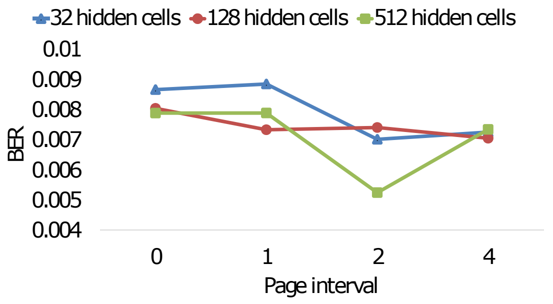


Figure 7: Hidden BER rates for VT-HI with ten PP steps. The illustrated irregularity demonstrates the effects of BER variance and program interference.

Figure 7 shows the sensitivity of BER for hidden data as a function of the number of hidden cells, using 10 PP steps to program the hidden data. Overall, the variation in bit error rate is small and generally insensitive to the number of hidden cells. There is some irregularity that is within the bounds of naturally occurring variance [32, 65, 66]. We do notice a small trend toward lower bit rates; because we only select unprogrammed cells for hiding data, any interference can flip cells that are slightly under-programmed (just short of the target threshold) to being just above the target threshold.

In this experiment, we selected 512 as an upper bound for the number of hidden bits. We measured a range of voltages for chips programmed with random data, and found that one could reliably get a minimum of 700 cells in the non-programmed state that are normally charged above our data hiding threshold. In other words, hiding more bits per page than 512 will likely leave telltale changes to the distribution of voltages.

Figure 7 indicates that, for any number of hidden bits in a page that satisfy our other constraints, the BER is small. The implication is that a small number of error-correcting hidden bits (e.g., 5%) will suffice.

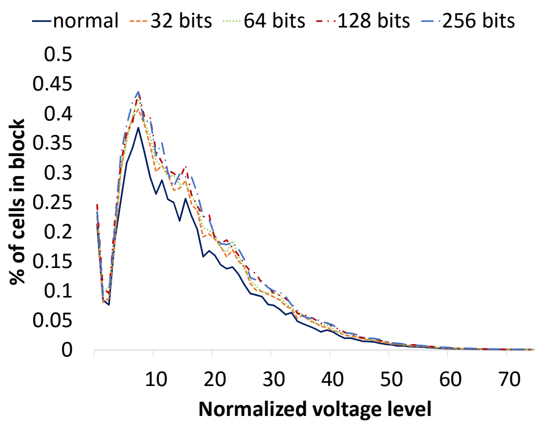


Figure 8: Average voltage level distributions for blocks after applying VT-HI. Hiding more bits creates a more noticeable shift to the right for non-programmed cells.

Figure 8 shows that hiding data using VT-HI creates only a tiny shift to the right for non-programmed cells. which can be attributed to normal voltage level distribution variability and errors, as well as small read disturbs and retention effects [32, 75]. Although we might be able to store 512 bits per page, we conservatively chose to hide 256 bits per page. In Section 7 we further explore the security of VT-HI using this configuration.

Finally, we measured the impact of page intervals on the BER for public data. Using no physical space between pages storing hidden data increased the public BER by 20%. At one physical page interval, the interference is reduced to a more acceptable 10% [64–66, 76]. Thus, subsequent experiments use a page interval of one.

7 Detectability

The primary criterion for evaluating VT-HI's success is whether an attacker with a full and detailed voltage-level analysis of the entire chip can infer whether some pages store hidden data from changes in voltage level distributions. In this section, we show that flash pages in VT-HI, with and without hidden data, cannot be distinguished.

Figure 9 illustrates the difficulty of detecting the existence of data hidden in VT-HI. The figure shows voltage level distributions from three blocks from different chips, first when they are normally programmed and then after applying VT-HI to hide data. The human eye has difficulty distinguishing which distributions come from blocks with hidden data.

SVM Analysis. Rather than rely on the human eye, we follow prior work [38] and instead use supervised machine learning to determine whether there are any detectable anomalies in the data. We use a support-vector machine (SVM) to predict whether pages and blocks contain hidden data. If VT-HI left aberrations in voltage levels that correlate with the presence of hidden data, an SVM would be able to identify these pages with better than 50% accuracy (i.e., better than flipping a coin). Our hypothesis is that the changes induced by data hiding are within normal noise.

To demonstrate this, we obtained data from three different hardware units of varying ages. We first measured the voltage level distributions and BER of three flash chips. For normal data characteristics of flash blocks and pages, we used normal programming for program/erase counts (PEC) ranging from 0 to 3000. We then hid data using VT-HI with the configuration parameters determined is §6 (threshold level 34, one page interval, 256 bits per page, ten PP steps) on all chips for blocks that were cycled to 0, 1000, and 2000 PEC.

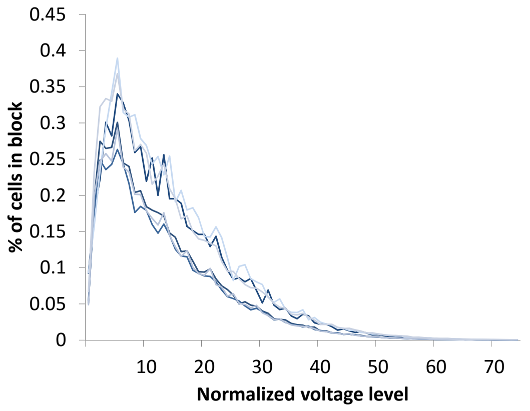
We created a training set for the SVM using datasets from two chips, and then we attempt to classify data from a third chip. For the training, we collected the voltage levels for all cells in the block with both normal and hidden data. We found that the flash chip data representativeness converged after analyzing 31 blocks. The classifier used optimal parameters obtained using grid search, and performed three-fold cross-validation for all three chips. As Wang et al. note [38], this is an unrealistically generous setup for the attacker. In reality, the attacker has to obtain knowledge of all possible PEC levels of the chip for both normal and hidden cases, and for multiple sample chips of the same vendor and model which would probably reduce the prediction accuracy.

In analyzing the voltage data we collected, the wear or number of program/erase cycles (PEC) had a first-order effect on the voltage levels.

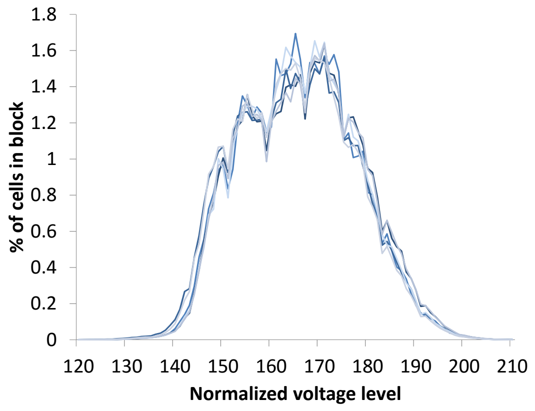
This sensitivity to PEC is illustrated in Figure 10, which presents SVM accuracy for samples at PEC of 0, 1000, and 2000. The x-axis is the PEC of normal data. For each line, there is a range of a few hundred P/E cycles where the accuracy of the SVM is at 50% (or random). For example, consider the PEC 2000 line. At x-axis of PEC 2000 (comparing to the same wear without hidden data), the SVM does not do better than random (50%); for a few hundred cycles on either side of this point, the accuracy is still effectively 50%. At extremely about 1000 PEC, and as PEC increases the classifier's accuracy increases. Thus, we expect that, as long as the wear on the device is uniform within several hundred PEC, an SVM would not be able to reliably classify which blocks have hidden data and which do not. A similar experiment at the page-level shows similar results

We also note that this experiment deliberately places VT-HI at a disadvantage, by training the SVM on the exact chip that was storing the hidden data. We repeated our tests on a data set that includes all of the chips (from the same vendor) and all PEC levels, and this decreased the SVM accuracy to 50% in all cases.

Finally, one might be concerned that an attacker could draw inferences from changes in characteristics of pub-



(a)



(b)

Figure 9: Voltage level distribution in blocks from different chips with normal distributions (light) and after applying VT-HI (dark). Results show (a) non-programmed and (b) programmed cells.

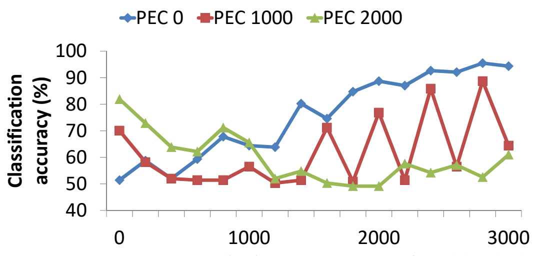


Figure 10: SVM prediction accuracy for block-level voltage distribution data for hidden blocks in three different PEC levels. Normal PEC (X axis) represents public data blocks in different PEC levels. 50% accuracy (dotted line) is equivalent to a random guess.

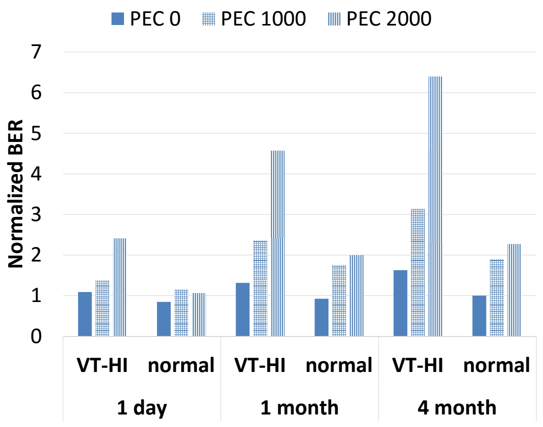


Figure 11: Normalized retention rate (versus "zero" time since programming) for data stored using VT-HI and normal data.

lic data, such as BER, mean voltage, and its standard deviation. Therefore, we performed another SVM analysis to classify blocks with and without hidden data according to these characteristics. Our results indicate that these analyses are also unsuccessful in classifying hidden data.

8 Performance and Applicability

In this section we evaluate and analyze the performance of VT-HI in terms of reliability, throughput, capacity, and energy. For each of these factors, we compare the performance of VT-HI to the most similar prior research paper, PT-HI, as summarized in Table 1. We also demonstrate that with proper configuration and vendor support we can increase the hidden data storage capacity of VT-HI by an order of magnitude. Finally, we verify the applicability of VT-HI on a chip from a second major vendor.

Reliability. As flash devices move toward smaller feature sizes, data errors will increase [77], increasing the importance of error-correcting codes and other countermeasures. Charges stored in flash memory cells gradually leak away over time, causing cell voltage to shift towards lower values. Bit errors accumulate in data as cell voltages shift across the predetermined reference threshold voltage. Here we characterize the reliability of bits hidden with VT-HI.

First, we measured the error rate for VT-HI in blocks with varying PEC levels, and find that BER is low and not affected by wear. We cycled blocks in three different chips to four distinct PEC levels. Next, we hid data using VT-HI, and measured the hidden data BER. Our results show that the BER is not affected by the age of the cells storing hidden data. For example, for PEC 0 the BER was 0.013. For other PEC the BER was roughly 0.011.

We also emulated data retention over longer periods by baking the flash chips in an oven, which accelerates the rate of charge leakage from the floating gates. Three data retention periods were used in our evaluation: 1 day, 1 month, and 4 months. The latter two periods were emulated by baking the flash [78]. Before retention, pseudorandom normal data and hidden data were first input and stored in flash. BERs and voltage distributions were then measured after retention using the previously saved input data. We normalized the BER after the relevant retention period to the BER measured immediately after the data was stored ("zero" time). We compared the rate of changes in the BER to the equivalent rate for public data on our test chip.

The results shown in Figure 11 indicate that retention time has no significant effect on the BER of hidden data for fresh cells (PEC 0). However, the hidden data error rate does increase on older cells, at a higher rate than for public data

For example, for 2000 PEC, the BER after zero time is 0.0099, and rises to 0.063 (6.3x) after four months, while for normal data the BER rises from 0.00003 to 0.000075 (2.3x). The reason for this reduced retention is that cells with higher PEC accumulate trapped charge and become more sensitive to leakage [77]. Moreover, hidden data BER degrades faster than public data BER. The reason is that the programming technique available to VT-HI (PP steps) is not accurate enough to ensure a large buffer zone around the threshold voltage level. Such zones are typically employed to minimize BER in degraded cells in existing flash package programming schemes (see X axis in Figures 2a and 2b), which are also used for storing public data in VT-HI.

These results indicate that additional redundancy would be prudent when hiding data in older cells. Rewriting (refreshing) hidden data every several months, even only after the device reaches 1K PEC, can also significantly improve retention [79]. Finally, to provide additional protection against data loss (e.g., due to bad blocks) data can be further encoded using RAID-like schemes, similarly to normal data [80].

Throughput. Here, we calculate the expected read and write throughput from VT-HI and our closest competitor, PT-HI, using reasonable parameters from current flash chips. We find that VT-HI can deliver an order of magnitude better throughput for hidden data than the best possible configuration for PT-HI.

Under VT-HI's optimal configuration (256 hidden bits per page and 4 logical page intervals), we can estimate the time it would take to encode hidden data in a block: $(600+90)\cdot 10\cdot 64/1,000,000 = 0.44s$ with a PP time of 600 us and read time of 90 us for 10 PP and read steps, and 64 pages per block. Assuming 15,593 hidden data bits per block, this translates to a throughput of 35Kb/s. This figure takes into account a 0.5% hidden BER, which, after applying standard ECC codes, translates to 243.6 bits of data per page (i.e., $\approx$13 parity bits).

We repeated this calculation for PT-HI, assuming its optimal setup with a negligible hidden data error rate. We use the optimal configuration in [38] of 625 per-page PP steps and a 4-page interval for hiding data, which

Method	*Relia bility*	*Perf.*	*Power*	*Public data integrity*	*Repeated Reads*	*Capacity*
PT-HI [38]	±	-	-	+	-	±
VT-HI	+	±	±	-	+	±

Table 1: Our contribution compared to Wang et al. [38].

translates to 72Kb of hidden bits per block. The page program latency used is 1.2 ms and block erase latency is 5 ms. In this setup, the time it would take to write hidden data is $(1.2\cdot 64+5)\cdot 625/1,000 = 51.1s$ per block. Therefore, even for this ideal setup, the optimal throughput for PT-HI is only 1.4Kb/s. We note that PT-HI's performance dramatically deteriorates in setups where the device has undergone even a few hundred PEC due to its increasing BER. We also note that PT-HI wears out the device much faster than VT-HI since it requires 60x more programming steps in order to encode data.

Decoding hidden data that was encoded using VT-HI in a page requires only a single read operation (following a voltage reference shift command). This translates to $90\cdot 64\cdot 1/1,000,000 = 0.006s$ for decoding the data hidden in a block, and a throughput of 2.7 Mb/s. For PT-HI, 30 PP and read operations are required to decode data from a page. This translates to $(600+90)\cdot 64\cdot 30/1000000 = 1.32s$ for decoding the hidden data in a block, and a throughput of 54 kb/s.

Improved Capacity. As we explain in §6.2, our prototype can only reliably encode data in non-programmed cells, which we found to keep us to under 700 bits per page to avoid telltale disruptions in the distribution of voltages. We conjecture that, with controller-internal programming tools, we could apply the same basic idea to a larger number of cells, which should potentially increase hidden data capacity.

In this section, we evaluate the impact on the risk of detection when more non-programmed cells are used. We repeat the SVM analysis in §7. We emulate finer-grained programming by using a single PP step ($m = 1$) instead of ten, and increase the hidden bits per page by a factor of ten. We then adjust the hidden data voltage threshold to level 15 to keep the voltage levels of cells with hidden data within the expected distribution. We kept the page interval the same (1 physical page).

Figure 12 shows the SVM accuracy results for our simulated higher-capacity configuration, on block-level data. Similarly to the results for hiding 256 bits, the results are highly sensitive to PEC. If we only consider ranges where the hidden and non-hidden blocks have PEC within a few hundred cycles of each other, the accuracy is generally low (50–60%), but slightly higher than the other experiment. Some of the increased accuracy is attributable to the lack of precision in PP, especially when only a single step is used.

Hidden data BER for data in the enhanced VT-HI configuration was only 2%. After applying standard ECC

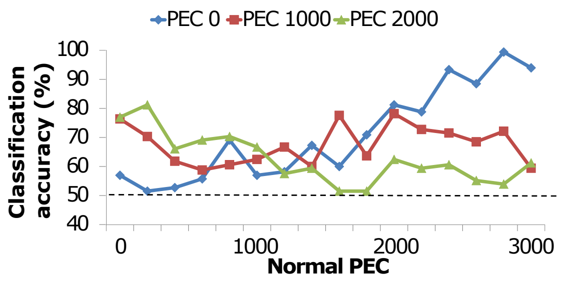


Figure 12: SVM prediction accuracy for an enhanced VT-HI configuration that hides 9x more data. Normal PEC (X axis) represents public data blocks in different PEC levels. 50% accuracy (dotted line) is equivalent to a random guess.

codes this translates to 2197 of data bits per page (14% are used for ECC). After accounting for ECC to mask the increased BER, this represents a 9x increase in usable hidden data capacity (and twice as much as PT-HI).

Energy. For our chip, we estimated the energy required for various data encoding operations using VT-HI and PT-HI (again, in an ideal setup). These include read, program and erase operations, as well as partial programming. We then used these estimated values to calculate the amount of energy required for writing a bit of hidden data. The results show that for VT-HI the energy required for hiding data is 1.1 mJ per page, as opposed to 43 mJ for PT-HI. This data indicates that, if an adversary read two snapshots of the device energy usage statistics, effectively there would not be a telltale difference for VT-HI and a system without hidden data. For instance, the energy overhead of our PP-based is less impactful than, say, extra reads from the device. With an in-controller VT-HI implementation we expect energy overheads could be reduced further.

Applicability. Finally, to verify that our method also applies to other flash chip models, we tested it on a 1x-nm 16GB MLC chip model from a different major vendor (also under a similar NDA). The flash package contains 2096 blocks, with page size of 18256 bytes. We tested our method on a fresh chip (PEC 0) and hid a 256 bit payload in relevant pages (taking into account architecture-specific page intervals). The resulting BER was 1%, similar to the one in the first model.

9 Discussion

This section discusses various applications for which VT-HI could be a useful building block.

9.1 Authentication and Provenance

One property of our approach is that erasing a block of public data on the flash device (thereby de-charging the cells) also erases any hidden payload in the cells. This property does not imply that a user cannot modify normal data; such modifications simply require the user to repeat the hiding process with the same hidden data on newly written normal data.

Many applications require some form of *proof to the trustworthiness and provenance* of their data. A number of systems find ways to embed a signature or metadata in the data file itself. VT-HI could be incorporated into these systems to embed metadata in the physical pages storing this data; only a trusted application can rewrite a file and embed hidden metadata in the device. For example, flash chip steganography enables counterfeit detection by watermarking original parts [38]. Archival storage systems authenticate the identity of data objects [81]. Embedded watermarks in storage media identify ownership of digital objects to prevent copyright infringements [11]. Secure file systems persist the keys required for accessing data to their storage media [59].

9.2 Steganography

VT-HI can also be used as a building block for implementing a steganographic system [51, 57–60]. Implementing a complete steganographic system is beyond the scope of this paper, but, in the interest of brevity, we discuss the main challenges of such a solution.

Basic Design. A VT-HI-capable system would include a publicly visible, encrypted volume, within which a user can store a hidden, encrypted data volume. To access the hidden volume, a user would input the secret key at mount time. Data can then be read and written from this volume using standard block-level operations.

The security of the hidden volume stems from the security of VT-HI. An attacker that inspects the device once, including all low-level characteristics, will not be able to differentiate flash pages that contain hidden data from those that do not, without the secret key.

Hiding VT-HI. The presence of a VT-HI-capable SSD may still raise the suspicion of an adversary that data is hidden. This problem is common to many existing systems [60], and can be mitigated in several ways. First, we can further assume that firmware update capability is available to the user via secure channels, so the VT-HI capability can be loaded whenever the user accesses hidden data and then immediately removed. Alternatively, the VT-HI capability can be included by default as an extension of open-source SSD firmware [82], allowing users to configure the firmware at will to operate with and without hiding capabilities.

Metadata Persistence and Security. VT-HI relies on configuration metadata, such as m, Vth, and the number of bits per page, which must be persisted and recovered on bootstrap. Because the metadata is small, the metadata could be included in the hidden key. Alternatively, the metadata can be encrypted and stored persistently in predetermined locations on flash, or, similarly to the hiding firmware itself, saved and reloaded from an external source. From a security standpoint, the metadata configuration values for a specific chip model may be known to a diligent adversary. However, even with full knowledge of the configuration metadata, without the secret key the adversary is still unaware of the location of cells containing hidden bits and cannot recover them.

Other metadata persistence issues, such as recovering the hidden volume LBA for every set of pages, may require sacrificing some hidden capacity or more sophisticated mapping data structures and algorithms, which we leave as future work.

Multiple-Snapshot Adversary. A stricter threat model involves an adversary capable of comparing multiple snapshots of the device taken over time. In this case, storing hidden data while leaving the public data unchanged leaves telltale signs of voltage manipulations that prevent users from plausibly denying the existence of hidden data. To mitigate, the hiding firmware can piggyback either public data writes (similar to [58]). Alternatively, the hiding firmware can utilize wear-leveling and other SSD-internal activities [61, 79], to create the requisite cover traffic. A trade-off here is that firmware-internal bookkeeping which operates without the private key for too long will eventually damage hidden data by causing internal data movements that copy data without also copying the hidden payload. We note however that hidden data overwrites when operating the system without the hidden key is an inherent limitation of almost all existing steganographic systems [60].

Capacity. The current implementation of VT-HI can only hide a few hundred bits per flash page. We believe that many privacy-concerned users will find the strong deniability offered by VT-HI as a reasonable tradeoff for reduced capacity. Also, in §6.2 we explain how vendor support may significantly alleviate this limitation (e.g., hide data as TLC in MLC cells).

10 Conclusions

In this work we present a new method for hiding data in flash using the inherent variability in voltage level distributions of flash cells. This variation occurs naturally on flash chips, even from the same vendor and model. We manipulate the voltage levels in cells to hide data within normal voltage intervals. Our manipulations hide an additional hidden bit in cells that already store a public bit by mimicking common methods to increase flash densities. Without the hiding key, an attacker cannot detect cells with hidden data even using favorable supervised learning. In comparison with the state of the art, our method achieves respectively 24x and 50x improvement for encoding and decoding throughput of hidden data, and is 37x more power efficient. Our technique is applicable to multiple chip models, allows users to store data even on flash cells that endured significant wear, and imposes significantly less wear while doubling total hidden capacity compared with prior work.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their insightful comments on earlier drafts of the work. This research was supported by Grant 2014621 from the United States-Israel Binational Science Foundation (BSF), by Grant CNS-1526707 from the United States National Science Foundation (NSF), and VMware. This work was done in part while Porter was at Stony Brook University.

References

[1] Kingston. Nearly half of organizations have lost sensitive or confidential information on USB drives in just the past two years. http://www.kingston.com/en/company/press/article/2661, 2011.

[2] Independent. BBC's panorama team loses confidential information relating to a secret british army unit. http://www.independent.co.uk/news/uk/home-news/exclusive-bbcs-panorama-team-loses-confidential-information-relating-to-a-secret-british-army-unit-9580340.html, 2014.

[3] WCSH6. USB drive containing personal information of 950 jetport workers, missing. http://www.wcsh6.com/news/local/portland/usb-drive-containing-personal-information-of-950-jetport-workers-missing/251514955, 2016.

[4] Computer World. NASA breach update: Stolen laptop had data on 10,000 users. http://www.computerworld.com/article/2493084/security0/nasa-breach-update--stolen-laptop-had-data-on-10-000-users.html, 2012.

[5] BBC News. Blackmail fear over lost raf data. http://news.bbc.co.uk/2/hi/uk/8066586.stm, 2009.

[6] The Register. Youth jailed for not handing over encryption password. http://www.theregister.co.uk/2010/10/06/jail_password_ripa/, 2010.

[7] Wikipedia. Key disclosure law. http://en.wikipedia.org/wiki/Key_disclosure_law.

[8] Denver Post. Password case reframes fifth amendment rights in context of digital world. http:

//www.denverpost.com/news/ci_19669803, 2012.

[9] PCWorld. Prepare to take your laptop to another country. http://www.pcworld.com/article/2886367/prepare-to-take-your-laptop-to-another-country.html, 2015.

[10] FBI. Economic espionage. https://www.fbi.gov/about-us/investigate/counterintelligence/economic-espionage.

[11] Ingemar J Cox, Matthew L Miller, Jeffrey Adam Bloom, and Chris Honsinger. *Digital watermarking*, volume 1558607145. Springer, 2002.

[12] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using IC fingerprinting. In *IEEE Symposium on Security and Privacy*, 2007.

[13] Malek Ben Salem. Security challenges and requirements for industrial control systems in the semiconductor manufacturing sector. NIST Workshop on Cyber-Security for Cyber-physical Devices, 2012.

[14] Microsoft Corporation. Windows BitLocker drive encryption frequently asked auestions. http://technet.microsoft.com/en-us/library/cc766200%28WS.10%29.aspx, 2009.

[15] OS X mavericks: Encrypt the information on your disk with filevault. http://support.apple.com/kb/PH13729.

[16] Yinglei Wang, Wing kei Yu, Shuo Wu, G. Malysa, G.E. Suh, and E.C. Kan. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In *IEEE Symposium on Security and Privacy (SP)*, 2012.

[17] Laura M. Grupp, John D. Davis, and Steven Swanson. The bleak future of nand flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST, pages 2–2. USENIX Association, 2012.

[18] Laura M. Grupp, John D. Davis, and Steven Swanson. The harey tortoise: Managing heterogeneous write performance in SSDs. In *Proceedings of the USENIX Conference on Annual Technical Conference*, USENIX ATC, 2013.

[19] Kang-Deog Suh, Byung-Hoon Suh, Young-Ho Lim, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Kwon, Byung-Soon Choi, Jin-Sun Yum, Jung-Hyuk Choi, Jang-Rae Kim, and Hyung-Kyu Lim. A 3.3 v 32 Mb NAND flash memory with incremental step pulse programming scheme. *IEEE Journal of Solid-State Circuits*, 30(11):1149–1156, Nov 1995.

[20] The Register. Good gravy, Toshiba QLC flash chips are getting closer. http://www.theregister.co.uk/2016/07/18/tosh_qlc_flash_chips_getting_closer, 2016.

[21] Taeho Kgil, D. Roberts, and T. Mudge. Improving NAND flash based disk caches. In *35th International Symposium on Computer Architecture (ISCA*, 2008.

[22] H. Nagashima, T. Tanaka, K. Kawai, and K. Quader. Nonvolatile semiconductor memory device which uses some memory blocks in multi-level memory as binary memory blocks, August 3 2006. US Patent App. 11/391,299.

[23] F. Yu, A.C. Ma, S. Chen, and Y.T. Chang. Endurance and retention flash controller with programmable binary-levels-per-cell bits identifying pages or blocks as having triple, multi, or single-level flash-memory cells, January 2 2014. US Patent App. 13/788,989.

[24] S.C. Wong and K. Johnsen. Data coding for multi-bit-per-cell memories having variable numbers of bits per memory cell, October 15 2002. US Patent 6,466,476.

[25] N.J. Wakrat and T.M. Toelkes. Dynamically allocating number of bits per cell for memory locations of a non-volatile memory, March 19 2013. US Patent 8,402,243.

[26] Seungjae Lee, Young-Taek Lee, Wook-Kee Han, Dong-Hwan Kim, Moo-Sung Kim, Seung-Hyun Moon, Hyun Chul Cho, Jung-Woo Lee, Dae-Seok Byeon, Young-Ho Lim, Hyung-Suk Kim, Sung-Hoi Hur, and Kang-Deog Suh. A 3.3 v 4 Gb four-level NAND flash memory with 90 nm CMOS technology. In *IEEE International Solid-State Circuits Conference*, ISSCC, 2004.

[27] Micron eMMC Linux enablement - SLC mode. https://prod.micron.com/~/media/documents/products/technical-note/emmc/tn5205_emmc_linux_enablement.pdf, 2012.

[28] Anandtech. Transcend announces SuperMLC: Pseudo-SLC SSDs for industrial market. http://www.anandtech.com/show/9882/transcend-announces-supermlc-pseudoslc-ssds-for-industrial-market, 2015.

[29] Electronic Design. Pseudo-SLC flash provides design flexibility. http://electronicdesign.com/site-files/electronicdesign.com/files/uploads/2013/09/FAQs-Toshiba-September.pdf, 2013.

[30] Tom's hardware. JMicron JMF670H SSD controller preview. `http://www.tomshardware.com/reviews/jmicron-jmf670h-ssd-controller,4161.html`, 2015.

[31] Open NAND flash interface. `http://www.onfi.org`. 2016.

[32] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. Error characterization, mitigation, and recovery in flash memory based solid-state drives. *arXiv preprint arXiv:1706.08642*, 2017.

[33] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu. Data retention in mlc nand flash memory: Characterization, optimization, and recovery. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture*, HPCA, 2015.

[34] Lorenzo Zuolo, Cristian Zambelli, Rino Micheloni, Marco Indaco, Stefano Di Carlo, Paolo Prinetto, Davide Bertozzi, and Piero Olivo. SSDexplorer: A virtual platform for performance/reliability-oriented fine-grained design space exploration of solid state drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1627–1638, 2015.

[35] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013.

[36] Toshiba's 768gb 3D QLC NAND flash memory: Matching TLC at 1000 P/E cycles? Anandtech, `https://www.anandtech.com/show/11590/toshiba-768-gb-3d-qlc-nand-flash-memory-1000-p-e-cycles`.

[37] Western digital unveils 96-layer nand, 4-bit qlc breakthrough. Extremetech, `https://www.extremetech.com/extreme/251774-western-digital-announces-new-96-layer-nand-4-bit-qlc-breakthrough`.

[38] Yinglei Wang, Wing kei Yu, S.Q. Xu, E. Kan, and G.E. Suh. Hiding information in flash memory. In *IEEE Symposium on Security and Privacy (SP)*, pages 271–285, 2013.

[39] Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Kei S. Yu, G. Edward Suh, Edwin Kan, and Steven Swanson. Extracting device fingerprints from flash memory by exploiting physical variations. In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, pages 188–201, 2011.

[40] Amir Rahmati, Matthew Hicks, and Atul Prakash. Approximate flash storage: A feasibility study. Presented at the Workshop on Approximate Computing Across the Stack, 2016.

[41] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal Processing*, 90(3):727 – 752, 2010.

[42] Fabien A.P. Petitcolas, R.J. Anderson, and M.G. Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7):1062–1078, Jul 1999.

[43] A. Swaminathan, Min Wu, and K.J.R. Liu. Digital image forensics via intrinsic fingerprints. *IEEE Transactions on Information Forensics and Security*, 3(1):101–117, March 2008.

[44] Wojciech Mazurczyk. VoIP steganography and its detection - a survey. *ACM Computing Surveys (CSUR)*, 46(2):20, 2013.

[45] Wojciech Fraczek, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Hiding information in a stream control transmission protocol. *Computer Communications*, 35(2):159 – 169, 2012.

[46] Bernard B. Wu and Evgenii E. Narimanov. Analysis of stealth communications over a public fiber-optical network. *Opt. Express*, 15(2):289–301, Jan 2007.

[47] P.R. Prucnal, M.P. Fok, K. Kravtsov, and Zhenxing Wang. Optical steganography for data hiding in optical networks. In *16th International Conference on Digital Signal Processing*, ICDSP, 2009.

[48] M.P. Fok, Zhexing Wang, Yanhua Deng, and P.R. Prucnal. Optical layer security in fiber-optic networks. *IEEE Transactions on Information Forensics and Security*, 6(3):725–736, Sept 2011.

[49] Qian Wang, Kui Ren, Guancheng Li, Chenbo Xia, Xiaobing Chen, Zhibo Wang, and Qin Zou. Walls have ears! opportunistically communicating secret messages over the wiretap channel: From theory to practice. In *Proceedings of the 22Nd Conference on Computer and Communications Security*, CCS, 2015.

[50] X. Zhou, HweeHwa Pang, and K.-L. Tan. Hiding data accesses in steganographic file system. In *Proceedings of the 20th International Conference on Data Engineering*, 2004.

[51] Ross Anderson, Roger Needham, and Adi Shamir. The steganographic file system. In *Information Hiding*, volume 1525 of *Lecture Notes in Computer Science*, pages 73–82. Springer Berlin Heidelberg, 1998.

[52] Jin Han, Meng Pan, Debin Gao, and HweeHwa Pang. A multi-user steganographic file system on untrusted shared storage. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC, 2010.

[53] Kefa Rabah. Steganography-the art of hiding data. *Information Technology Journal*, 3:245–269, 2004.

[54] Andrew D. McDonald and Markus G. Kuhn. StegFS: A steganographic file system for linux. In *Information Hiding*, volume 1768 of *Lecture Notes in Computer Science*, pages 463–477. Springer Berlin Heidelberg, 2000.

[55] HweeHwa Pang, K.-L. Tan, and X. Zhou. Stegfs: a steganographic file system. In *Proceedings of the 19th International Conference on Data Engineering*, ICDE, 2003.

[56] Open Crypto audit project. `http://opencryptoaudit.org/`.

[57] Adam Skillen and Mohammad Mannan. On implementing deniable storage encryption for mobile devices. In *Network & Distributed System Security Symposium*, NDSS, 2013.

[58] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. Toward robust hidden volumes using write-only oblivious ram. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 203–214, New York, NY, USA, 2014. ACM.

[59] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. Defy: A deniable, encrypted file system for log-structured storage. In *The Network and Distributed System Security Symposium*, NDSS, 2015.

[60] Aviad Zuck, Udi Shriki, Donald E. Porter, and Dan Tsafrir. Preserving Hidden Data with an Ever-Changing Disk. In *ACM Workshop on Hot Topics in Operating Systems*, HotOS, 2017.

[61] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Comput. Surv.*, 37(2):138–163, June 2005.

[62] Anxiao Jiang, V. Bohossian, and J. Bruck. Rewriting codes for joint information storage in flash memories. *IEEE Transactions on Information Theory*, 56(10):5300–5313, Oct 2010.

[63] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, and J.K. Wolf. Characterizing flash memory: Anomalies, observations, and applications. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2009.

[64] Y. Di, L. Shi, K. Wu, and C. J. Xue. Exploiting process variation for retention induced refresh minimization on flash memory. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016.

[65] Y. Pan, G. Dong, and T. Zhang. Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(7):1350–1354, 2013.

[66] Yeong-Jae Woo and Jin-Soo Kim. Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT, 2013.

[67] Nikolaos Papandreou, Thomas Parnell, Haralampos Pozidis, Thomas Mittelholzer, Evangelos Eleftheriou, Charles Camp, Thomas Griffin, Gary Tressler, and Andrew Walls. Using adaptive read voltage thresholds to enhance the reliability of MLC NAND flash memory systems. In *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, GLSVLSI, 2014.

[68] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *2013 IEEE 31st International Conference on Computer Design*, ICCD, 2013.

[69] Ki-Tae Park, Myounggon Kang, Doogon Kim, Soon-Wook Hwang, Byung Yong Choi, Yeong-Taek Lee, Changhyun Kim, and Kinam Kim. A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel msb program scheme for MLC NAND flash memories. *IEEE Journal of Solid-State Circuits*, 43(4):919–928, April 2008.

[70] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, USENIX ATC, 2008.

[71] M. Murugan and D.H.C. Du. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *IEEE 27th Symposium on Mass Storage Systems and Technologies*, MSST, 2011.

[72] Y. J. Woo and J. S. Kim. Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In *Proceedings of the International Conference on Embedded Software*, EMSOFT, 2013.

[73] NAND flash memory tester (SigNASII). http://www.siglead.com/eng/innovation_signas2.html. 2016.

[74] Jong-Ho Park, Sung-Hoi Hur, Joon-Hee Leex, Jin-Taek Park, Jong-Sun Sel, Jong-Won Kim, Sang-Bin Song, Jung-Young Lee, Ji-Hwon Lee, Suk-Joon Son, Yong-Seok Kim, Min-Cheol Park, Soo-Jin Chai, Jung-Dal Choi, U-In Chung, Joo-Tae Moon, Kyeong-Tae Kim, Kinam Kim, and Byung-Il Ryu. 8 gb MLC (multi-level cell) NAND flash memory using 63 nm process technology. In *Technical Digest of IEEE International Electron Devices Meeting*, IEDM, pages 873–876, 2004.

[75] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu. Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN, 2015.

[76] Wei Wang, Tao Xie, and Deng Zhou. Understanding the impact of threshold voltage on mlc flash memory performance and reliability. In *Proceedings of the 28th ACM International Conference on Supercomputing*, ICS, 2014.

[77] Yue Li, Eyal En Gad, Anxiao Jiang, and Jehoshua Bruck. Data archiving in 1x-nm NAND flash memories: Enabling long-term storage using rank modulation and scrubbing. In *2016 IEEE 54th International Reliability Physics Symposium*, 2016.

[78] Mingzhen Xu, Changhua Tan, and MingFu Li. Extended Arrhenius law of time-to-breakdown of ultrathin gate oxides. *Applied Physics Letters*, 82(15):2482–2484, Apr 2003.

[79] Yu Cai, G. Yalcin, O. Mutlu, E.F. Haratsch, A. Cristal, O.S. Unsal, and Ken Mai. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *IEEE 30th International Conference on Computer Design*, ICCD, 2012.

[80] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *15th USENIX Conference on File and Storage Technologies*, FAST, 2017.

[81] Michael Factor, Ealan Henis, Dalit Naor, Simona Rabinovici-Cohen, Petra Reshef, Shahar Ronen, Giovanni Michetti, and Maria Guercio. Authenticity and provenance in long term digital preservation: Modeling and implementation in preservation aware storage. In *First Workshop on Theory and Practice of Provenance*, TAPP, 2009.

[82] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. Lightnvm: The linux open-channel SSD subsystem. In *15th USENIX Conference on File and Storage Technologies*, FAST, 2017.