Invisible Bits: Hiding Secret Messages in SRAM's Analog Domain

Jubayer Mahmod
jubayer@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Matthew Hicks
mdhicks2@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

Electronic devices are increasingly the subject of inspection by authorities. While encryption hides secret messages, it does not hide the transmission of those secret messages—in fact, it calls attention to them. Thus, an adversary, seeing encrypted data, turns to coercion to extract the credentials required to reveal the secret message. Steganographic techniques hide secret messages in plain sight, providing the user with plausible deniability, removing the threat of coercion.

This paper unveils *Invisible Bits* a new steganographic technique that hides secret messages in the analog domain of Static Random Access Memory (SRAM) embedded within a computing device. Unlike other memory technologies, the power-on state of SRAM reveals the analog-domain properties of its individual cells. We show how to quickly and systematically change the analog-domain properties of SRAM cells to encode data in the analog domain and how to reveal those changes by capturing SRAM's power-on state. Experiments with commercial devices show that *Invisible Bits* provides over 90% capacity—two orders-of-magnitude more than previous on-chip steganographic approaches, while retaining device functionality—even when the device undergoes subsequent normal operation or is shelved for months. Experiments also show that adversaries cannot differentiate between devices with encoded messages and those without. Lastly, we show how to layer encryption and error correction on top of our message encoding scheme in an end-to-end demonstration.

CCS CONCEPTS

• **Hardware → Communication hardware, interfaces and storage**; • **Security and privacy → Pseudonymity, anonymity and untraceability**.

KEYWORDS

Steganography, SRAM aging, covert channel

ACM Reference Format:
Jubayer Mahmod and Matthew Hicks. 2022. Invisible Bits: Hiding Secret Messages in SRAM's Analog Domain. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3503222.3507756

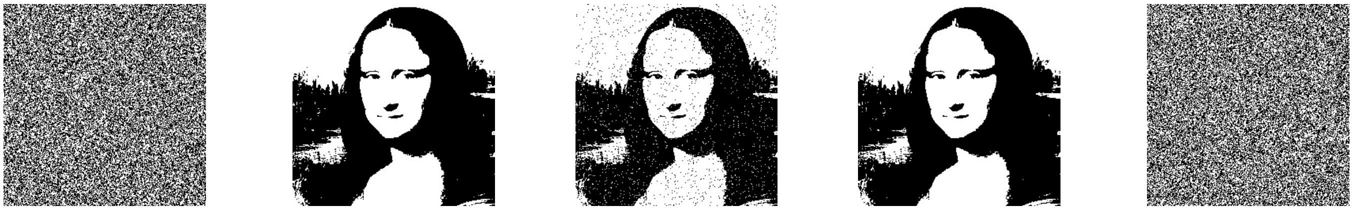
ASPLOS '22, February 28 – March 4, 2022, Lausanne, Switzerland
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9205-1/22/02.
https://doi.org/10.1145/3503222.3507756

1 INTRODUCTION

In many cases, encryption alone is insufficient as revealing the transmission of secret messages opens one to the threat of physical coercion (commonly called a rubber hose attack [6]) or even death [27]. Historically such ramifications were limited to spies, who used dead drops as a cover for communication. For example, during the cold war, Martha Peterson used dead rats as literal dead drops to communicate with Aleksandr Ogorodnik, a member of the Soviet Foreign Ministry, who was relaying secret foreign cables [49]. The proliferation of computing devices and the rise of concern about information passing across national borders push the need to hide the presence of secret messages down to the level of normal citizens. Consider that at the United States border alone, searches of electronic devices increased 400% over the last five years [50, 51]. Such searches extend well beyond the borders of the United States, with potentially worse consequences in other nations [1, 11, 14]. In response, travelers are already looking for ways to hide information. For example, in 2012, a Syrian refugee smuggled evidence of human rights violations by hiding memory chips in a wound [4, 33]. Given that border controls are getting tighter, with greater emphasis on electronic devices, and more advanced analysis [29], the need to hide the act of communication grows.

As restrictions continue to reduce the free flow of information, hiding the presence of secret messages becomes paramount; hiding the existence of a secret message prevents forceful key disclosure or other attacks from being attempted, because the attacker is unaware of the communication. Such message hiding is referred to as steganography. Steganography is communication over a covert channel that is modulated on top of everyday objects. A key property of steganography is plausible deniability: attackers are unable to distinguish between objects that carry covert information and objects that do not. Plausible deniability is what allows for covert communication in plain sight. An effective steganography system provides both analog- and digital-domain plausible deniability and is able to withstand normal use of the carrier object without disturbing or revealing the secret message modulated on top.

Given the value of being able to hide the existence of data, researchers propose a range of stenographic techniques that work at various levels of the system stack. The most popular techniques work at the system level, hiding data in images [12, 53], audio files [54], video files [40], and within file system metadata [5, 30, 35]. Another set of techniques hides data in the transmission of cover data across a network [55]. These approaches tend to be very low capacity, because they cannot interfere with the cover data—otherwise, plausible deniability is lost. A higher capacity set of techniques encodes data in the device memory itself as opposed to in digital files stored by the memory; this set of techniques that hides data in the analog domain of a device is referred to as on-chip



(a) Original SRAM power-on state.

(b) Secret message to encode into SRAM.

(c) Power-on state after encoding.

(d) Received message after error correction.

(e) SRAM's power-on state after encrypted message encoding.

Figure 1: *Invisible Bits* and its various use cases on SRAM embedded in Texas Instruments MSP432 [21]

stenography. The main target of existing on-chip steganography techniques is Flash memory: (1) the charge held in a Flash cell [57] and (2) the program/erase time of a Flash cell [52]. While these approaches marginally increase the capacity of file- and network-based techniques, they **all suffer from limited resilience, i.e., it is easy for the attacker to erase the message, even if they cannot prove one exists**.

In this paper, we design and implement an on-chip steganographic technique that uses a more ubiquitous on-chip memory that dramatically increases capacity and resilience—while providing both digital- and analog-domain plausible deniability: Static Random Access Memory (SRAM). Our system *Invisible Bits*, encodes secret messages into SRAM's analog-domain properties by controlling and accelerating the natural aging process that all transistors experience. SRAM's power-on state is unique in that individual cells power-on to a value determined by the relative analog-domain properties of the cross-coupled inverter gates at the heart of every cell. The value an SRAM cell holds determines which of the two inverters is active. While an inverter is active, it gradually ages, changing its analog-domain properties; given enough aging, the power-on state of the cell will change, making analog-domain changes visible in the digital domain through its power-on state. We show how to control the direction and rate of this aging process in a way that allows us to dictate the power-on state of >90% of SRAM cells. We show that even with such a significant change in the analog domain, the device functions normally at the digital level. Figure 1 shows an example image encoded into a device using *Invisible Bits*.

Invisible Bits has the following advantages as a steganographic technique:

- **Covert:** *Invisible Bits* does not store information in a traditional sense, which is attributable to the volatility of SRAMs. The *payload* is decoupled from the normal functionality of SRAM as memory, which provides digital-domain plausible deniability. To protect against more powerful adversaries, *Invisible Bits* encrypts the payload before encoding, which ensures the indistinguishability of the power-on states of devices with a message encoded and those without.
- **Erase/Write tolerant:** *Invisible Bits* is robust against general-purpose device operation. That is, an encoded SRAM withstands regular memory operation (*i.e.,* read/write) without introducing significant error in the hidden message.
- **Ubiquity:** Most computing devices use on-chip SRAM either as a cache (e.g., desktop-class devices), or main memory (e.g., embedded devices), or configuration memory (e.g., Field-Programmable Gate Arrays), and so *Invisible Bits* applies to all of these devices.[1]
- **Constant time operation:** Since SRAM cells age due to the feedback loop inherent in their structure, all SRAM cells (that have power) age concurrently. Given the speed of writing to SRAM, payload encoding time is largely independent of SRAM size and payload size, but dominated by the time it takes to change SRAM's analog-domain properties so that they affect its power-on state.
- **Copy tolerant:** Sampling power-on state at normal operating conditions does not alter the encoded payload.

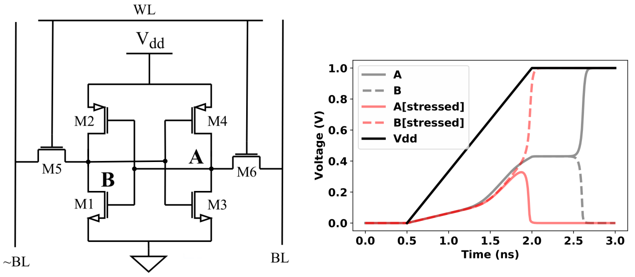
This paper makes the following contributions:

- We show that directed SRAM aging is a covert channel that one can exploit to transfer data with plausible deniability (§2 and §4).
- We implement and evaluate an end-to-end steganography system using modern, commercially available computing devices (§6).
- We improve information capacity by 100x compared to Flash-based on-chip steganographic techniques (§5.3).
- We show that the proposed system is resilient to long-term natural recovery (§5.1.3) and that it retains the encoded message reliably under normal operating conditions (§5.1.4).
- We implement and evaluate both error correction and encryption on top of *Invisible Bits*, showing that encryption provides analog-domain plausible deniability (§5).

2 BACKGROUND AND MOTIVATION

After turning-on a device, the SRAM embedded within the device retains its power-on state until software overwrites it. SRAM's uninitialized power-on state possesses interesting properties such as temporal and spatial randomness, making it an attractive security primitive in numerous applications such as physical unclonable functions (PUF), random number (TRNG), and device fingerprint generators [16, 17, 38]. As long as SRAM stores some data, regardless of whether it is from a power-on event or from software, it undergoes analog-domain changes in a process called aging; such analog-domain changes are reflected digitally in future power-on

[1] *Invisible Bits* requires the ability to access SRAM power-on state for a given device, before it is overwritten.



(a) (b)

Figure 2: (a) 6T SRAM cell schematic (b) simulation waveform of startup state of the cell.

states [25]. *Invisible Bits* harnesses, directs, and accelerates SRAM aging as means to reliably encode secret messages—without altering the functionality of the SRAM.

2.1 SRAM Startup Properties

Figure 2a shows the conventional 6-transistor SRAM cell, which consists of two cross-coupled inverters (transistors M1 and M2 form *Inverter-1*, while M3 and M4 form *Inverter-2*) and two access transistors (M5 and M6) that allow access to the logic state stored in the cell. At design time, aspect ratios of the *Inverter-1*'s PMOS and NMOS match perfectly with the PMOS and NMOS of *Inverter-2* $[(W/L)_2 = (W/L)_4$ and $(W/L)_1 = (W/L)_3]$. The symmetric design maximizes the static noise margin and minimizes the data retention voltage [18, 19]. Process variation causes post-manufacturing transistor mismatch—making($(W/L)_2 \neq (W/L)_4$ and $(W/L)_1 \neq (W/L)_3$; $v_{th_2} \neq v_{th_4}$ and $v_{th_1} \neq v_{th_3}$.

As a result of inverter asymmetry, one inverter turns on faster than the other, causing an SRAM cell to prefer one power-on state over another. When the cell is unpowered, all wires are at the ground voltage. As power is applied to the cell, both node A and B attempt to follow the supply voltage's (V_{dd}) ramp-up. After this initial mutual ramp-up, one of the transistors, M2 or M4, turns on faster due to differences in analog-domain properties, causing its output node (B or A, respectively) to pull up to the supply voltage, while the other node is pulled down to the ground by M3 or M1, respectively. Thus, the post-power-on steady-state voltage at nodes A and B depends on a hardware-level race condition between the inverters. The relative analog-domain properties (primarily threshold voltage) of the transistors largely determine the race winner. The grey curves in Figure 2b show that after $2ns$ of powering the cell up, node A and B settle at V_{dd} and ground, respectively. Because the threshold voltage of M4, $|v_{th_4}|$, is less the threshold voltage of M2, $|v_{th_2}|$, M4 switches on before M2 and eventually pulls node A up to V_{dd}. Node A is connected to the gate of M1, and it enters the saturation region (from cut-off) and pulls down node B to the ground. Thus, the cell's power-on state is 1.

2.2 Influencing Power-on Behavior

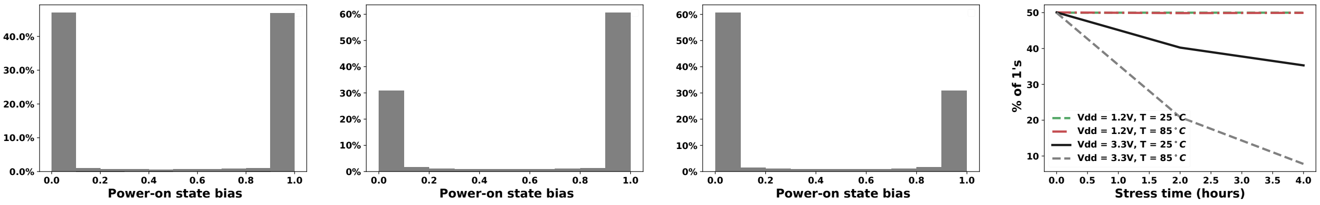
Circuit aging is a natural phenomenon that gradually *increases* the threshold voltage of transistors [3]. Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) are the primary causes of circuit aging in modern technology nodes [39]. For the purposes of analog-domain changes revealed through power-on states, BTI is of interest, while HCI has less effect [39] and affects both inverters equally since HCI involves switching and both inverters switch together. MOS devices suffer from BTI because of surface state generation in the oxide and substrate interface [15]. Positive Bias Temperature Instability (PBTI) affects the NMOS, while Negative Bias Temperature Instability (NBTI) affects the PMOS [23]. When a PMOS is operational (*i.e.,* $V_{sg} > |V_{th}|$), a strong vertical electric field creates interface states, which increases the threshold voltage over time. When $V_{sg} \leq |V_{th}|$, the NBTI effect recovers, it partially brings back the threshold voltage closer to nominal. Note that the recovery is partial, and so the magnitude of the threshold voltage increases over time. A similar effect is observable in NMOS devices [31].

Aging-induced degradation has the ability to determine the power-on state of SRAM cells.[2] The power-on state of a cell depends on the post-manufacturing variation in transistors and the magnitude of aging the transistors have experienced up to this point; the value held by the cell determines which transistor is aged. This means that SRAM cells undergo data-directed aging. We illustrate data-directed aging using a MOSFET Reliability Simulation (MOSRA) in HSpice [48]. Figure 2b plots waveforms generated from pre-aging and post-aging transient simulation using 45nm predictive technology model [2]. Assume the cell in Figure 2a is initially biased towards 1. Considering only PMOS mismatch, M4 turns on earlier than M2; hence, node A is pulled up to V_{dd} while node B is pulled down to the ground. If we keep the cell operational in this state (with node A = 1 and node B = 0), M4's threshold voltage gradually increases (due to NBTI aging). The increased threshold voltage causes M4 to turn-on at a later time during the next power-on event. Given enough aging time, the cell ultimately becomes biased towards 0 as M4 now turns-on later than M2. The red waveforms in Figure 2b capture this change.

We run two experiments to demonstrate the concept of (1) software directed and (2) accelerated aging. As discussed, stressing a cell with a logic state biases its power-on state complement of that logic state. That is, we expect the number of cells that power-on to 1 to increase if we write 0 to the entire SRAM and then aggressively age the device (Figure 3b). Conversely, we expect increased 0s if we fill SRAM with 1s before aging (Figure 3c). Compare these figures to the unaged SRAM in Figure 3a.

In the second experiment, we demonstrate the knobs that *Invisible Bits* exploits to accelerate NBTI aging for the purposes of encoding messages in SRAM's analog domain. Previous work shows that both supply voltage and operating temperature of a device affect the rate of NBTI aging. To validate this, we write 1 to all cells of four unaged MSP432 microcontrollers. We then exposed each microcontroller to a different combination of voltage and temperature, with nominal conditions being (1.2V, 25°C) and accelerated being (3.3V, 85°C). Figure 3d shows that voltage has the largest acceleration effect, which is magnified by increased temperature, a result supported by previous work [28].

[2] The impact of transistor aging is known to some degree by the SRAM PUF community, as modest aging has been used as a denial-of-service attack on SRAM PUFs [37]. Though it is not the focus of this paper, the results of our extreme/controlled aging suggest that it is possible to clone SRAM PUFs.



(a) (b) (c) (d)

Figure 3: Illustrates (a) startup distribution of an SRAM, data directed aging (b) when an SRAM holds logic 0 , (c) logic 1 during stress. (d) Illustrates accelerated aging using supply voltage and operating temperature.

3 THREAT MODEL

We adopt a threat model similar to Flash-based on-chip steganography techniques [52, 57]. In this threat model, the goal is for two parties to communicate via a plausibly-deniable covert channel; depending on the use case, the parties may be separate entities (e.g., for data transmission), or the same entity separated in time (e.g., for data storage). The communicating parties have full control over selecting a device as a steganographic medium. The threat model also assumes that the parties have a pre-shared key that they use for message encryption/decryption. Lastly, in the on-chip threat model, data transmission is the limiting factor, not throughput, as encoding and decoding are separated by the time it takes to transfer the device; capacity remains crucial as it determines if a message can fit on a single device.

The threat model assumes that the adversary gets temporary access to a device, otherwise, the attack devolves into a denial-of-service attack, which is impossible to stop. While in the adversary's possession, they can inspect, copy, overwrite, and erase its digital contents. In addition to data manipulation, the adversary may use the device to validate its functionality (e.g., scroll through pictures and contacts). The only restriction that we place on the adversary is that their analysis is non-invasive and non-destructive.

4 DESIGN

We leverage the stress-induced analog-domain change of an SRAM to design a plausibly deniable and resilient information encoding scheme: *Invisible Bits*. The design of *Invisible Bits* consists of two major parts: information encoding and information decoding. Figure 4 shows the steps involved with sending and receiving a message using *Invisible Bits*. The covert communication process starts by pre-processing the message. Pre-processing is where the transmitting party (i.e., Alice in Figure 4) applies their desired error correction and encryption schemes to the original message. The result is the *payload*. The transmitting party takes the payload and creates a program from it; the program writes the payload to SRAM and then halts execution by busy waiting. With SRAM set to the desired state, the transmitting party encodes the state in SRAM's analog domain by stressing the device with increased voltage and temperature. After sufficient stress time (we evaluate the impact of stress knobs in §5), the transmitting party passes the device to the receiving party. Examples of this could be through the mail, via a dead drop, or by crossing a monitored border.

Once the receiving party (i.e., Bob in Figure 4) has possession of the device, they need to extract the information encoded in SRAM's analog domain. For this, they first load a program crafted to retain SRAM's power-on state, then they capture several power-on states from the SRAM; this process can either be automated, as we do it for our experiments or by hand, since it only requires temporarily removing and replacing the power. The recovered payload is then passed through decryption and error correction steps if they were used during message encoding.[3]

The information encoding scheme consists of two major parts:

4.1 Message Pre-Processing

Before encoding a message into SRAM's analog domain, it passes through two optional pre-processing steps: (1) where an Error Correcting Code (ECC) is added on top of the message and (2) where the message is encrypted. As shown in the evaluation (§5), not all SRAM cells will undergo sufficient aging to change their power-on state to reflect the value loaded during the payload encoding step. Thus, to reduce error beyond what is possible through the stress and time knobs that *Invisible Bits* provides, we make it possible to layer ECC on top of *Invisible Bits*. The actual ECC method is orthogonal to *Invisible Bits*, but our analysis of errors from payload encoding and aging recovery in Section 5.2 provides guidance to users as to the most appropriate error correction approaches and the expected error rate and channel capacity.

We also allow users to layer encryption on top of *Invisible Bits*. The obvious benefit of encryption is providing confidentiality, but our analysis in Section 6 shows that encryption also provides analog-domain plausible deniability, because SRAM's power-on state properties before payload encoding match that of a random function. For a visual example of this effect, compare the pre-encoding power-on state in Figure 1a, with the post-encoding power-on states for a raw message in Figure 1c and an encrypted message in Figure 1e. Unlike ECC, we note that the choice of the encryption algorithm is crucial: the diffusion that is an important property for block ciphers means that even a single error in a block results in roughly half of the bits of the block flipping—dramatically increasing error rate. For example, using the industry-standard cipher AES-CBC turns an error rate of 0.8% into an error rate of 50% as the first erroneous bit causes the output of all subsequent blocks to become random. Thus, we advocate a stream cipher, which is error-neutral, i.e., error bits in the ciphertext are exactly the error bits in the plaintext, no less, no more. Our implementation uses AES-CTR as the stream cipher,

[3] We assume that the presence and order of error correction and encryption information are pre-shared between communicating parties.

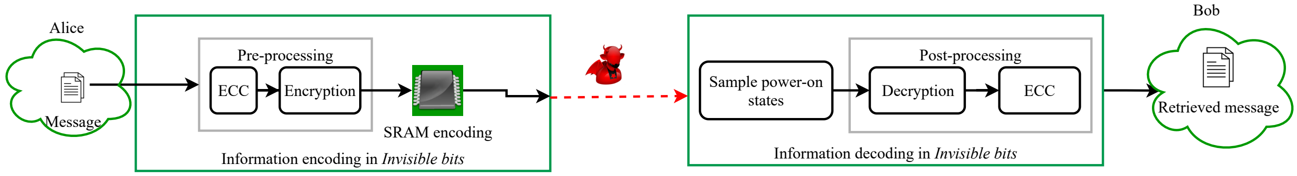


Figure 4: *Invisible Bits* overview.

where the nonce is the manufacturer's device ID.[4] Once passed through any error correction and encryption steps, the message becomes a *payload* that *Invisible Bits* encodes in the SRAM.

4.2 SRAM Analog-Domain Payload Encoding

In this step, *Invisible Bits* encodes the payload into the analog-domain properties of the SRAM embedded in a computing device. The first step is the creation of a software program that writes the payload to the SRAM. To automatically create such a program, we write a tool that takes a payload expressed as a binary file, and returns an assembly program that writes that payload to the SRAM. After the program initializes SRAM's state, it busy waits in an infinite loop. The instructions in the assembly program run from non-volatile memory on the device, i.e., not the SRAM. *Invisible Bits* then assembles this program and loads it onto the target device using the debugger. The second step powers on the target board at nominal conditions to allow the program to initialize the SRAM state. The third step is where the encoding occurs as we increase temperature and voltage to accelerate SRAM aging. After the user-defined stress time, encoding completes, the device is removed from the thermal chamber, and a camouflage program is loaded onto the device.

Algorithm 1 summarizes the entire message encoding process using *Invisible Bits*. First, we apply an error correction code (ECC) on the data (Line 1) and use a pre-shared key to encrypt the data along with the parity bits (Line 2). Then, we load the program to the device running at nominal voltage and temperature (Line 3-4). The physical encoding process starts at Line 5, where we elevate the voltage and temperature to $V_{acc.}$ and $T_{acc.}$, respectively. *wait(stress time)* controls how long we stress the device. Voltage $V_{acc.}$, temperature $T_{acc.}$, and stress time depend on the required bit error rate, as discussed in §5.

Algorithm 1: *Invisible Bits* message encoding

```
Input: Message (d)
Output: Encoded computing device
     Message pre-processing
  1: Apply ECC on the message
  2: Apply encryption on the message and parity
     SRAM analog-domain payload encoding
  3: Set {Vdd, Temp} = {$V_{nom.}$, $T_{nom.}$}
  4: Load binaries (Instructions and payload)
  5: Set {Vdd, Temp} = {$V_{acc.}$, $T_{acc.}$}
  6: wait(stress time)
```

4.3 Decoding Payloads via SRAM Power-on State

Once the receiving party gets access to the device, they have to extract the payload that the sender encoded in SRAM's analog domain. For this, *Invisible Bits* leverages the observation that the power-on state of SRAM reflects coarse-grain analog-domain information about each SRAM cell; specifically, the power-on state reveals which inverter gate at the heart of the SRAM cell turns-on first. The goal of the encoding process is to force one of the inverters to turn-on first by slowing the other inverter through accelerated NBTI aging (§2). Note that this means that the payload revealed through the power-on state is actually the complement of the payload used during the encoding step (like a negative in photography). Our experiments suggest dealing with inverted encoding requires no special treatment other than complementing power-on state at the decoding phase.

Before capturing the power-on state, the receiver loads a program on the device that ensures the integrity of SRAM's power-on state. For our implementation, we use a program that boots to an infinite loop, that runs entirely out of Flash memory. Then, to extract a reliable representation of the power-on state, the receiver captures multiple power-on states from the device. They use a majority voting scheme on the state captures to produce the retrieved payload. While any odd number of state captures works, we find that taking five captures is sufficient to filter noise.

4.4 Decoded Payload Post-Processing

Post-processing in the decoding phase is the same as encoding phases but in the reverse order. The only difference is that we have to start by inverting the majority power-on state from the decoding step. Algorithm 2 lists the steps to decode the information hidden in the power-on states of an SRAM. The process starts with power cycling the device N times and reading out the SRAM power-on states (Line 1-5), where N is an odd number. In Line 6, *Invisible Bits* apply majority voting on the SRAM states across trials. Once we have SRAM power-on state, we use the decryption and error correction and retrieve the message (Line 7).

5 EVALUATION

Table 1 lists the devices we test for the implementation feasibility of *Invisible Bits*. The list contains devices from single-cycle micro-controllers to complex general-purpose processors from different

[4] Having per-device nonces ensures that even the same messages produce different payloads. This protects against attacks the look for common prefixes in the ciphertext and attacks that look for the same power-on state across devices.

Algorithm 2: *Invisible Bits* message decoding

```
Input: Encoded device
Output: Decoded message
    Decoding payloads via SRAM power-on state
    Set {Vdd, Temp} = {$V_{nom.}, T_{nom.}$}
    repeat
        Power cycle the device
        Get power-on states
    until N times
    Decoded payload post-processing
    Apply majority voting
    Apply decryption and ECC
```

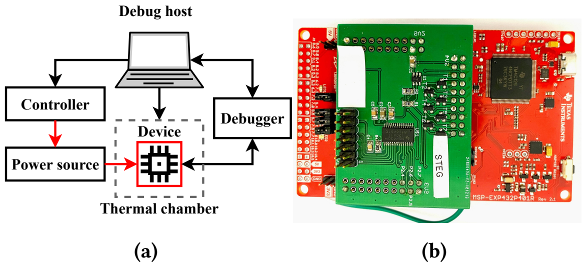


Figure 5: (a) Schematic diagram of the evaluation setup. (b) Evaluation platform. The red board is an MSP432P401 launchpad and the green board is our custom control board.

vendors.[5] We test these devices for their power-on state accessibility and aging acceleration. Although the magnitude of $V_{acc.}, T_{acc.}$, stress time, and message extraction error rate vary based on a device's manufacturing technology, SRAMs across the spectrum of devices behave regularly and predictably. Unless otherwise stated, we use MSP432P401 [21] devices for most of our experiments, keeping the analysis and discussion consistent.

We develop a control board that allows us to expedite the evaluation process by automating the power-on state sampling and accelerated aging. Although our setup is automated, encoding and decoding are able to be done by hand without changing the results. Figure 5a shows the high-level schematic diagram, and Figure 5b is a picture of the evaluation platform. The setup mainly consists of a controller, debug hardware, debug host, power source, and thermal chamber. The controller supplies power directly if the target device consumes a small amount of power (a few mA) but switches to an external power supply unit if the target demands higher current (*e.g.,* processor). The target device runs software written in assembly to read and write to the SRAM. When the target device is a microcontroller, a debugger directly reads out its main memory's power-on state and sends it to the debug host (*e.g.,* PC). However, processor cache access requires co-processor (for ARM Cortex-A device) operations to read cache contents.

All of our measurements eliminate the SRAM data remanence effect by driving the supply voltage of the device to the ground state, rather than waiting for it to discharge naturally. Our evaluation setup uses a Test-Equity thermal chamber [47] to control the temperature during measurements. To achieve maximum acceleration, we elevate the supply voltage and temperature to $3.3V$ (2.75x the nominal) and $85°C$ (3.4x the nominal),[6] respectively, during the encoding period.

5.1 Error Profiling

To exchange or store information reliably, we need an understanding of the system's error profile. The spatial distribution of error and capacity are the determinants in managing errors in a system. In this section, we analyze the origins and distribution of the error in *Invisible Bits.*

5.1.1 Primary error source. Failure to direct cells' power-on states, natural recovery, and standard device operation are potential error sources in *Invisible Bits.* The error magnitude in *Invisible Bits* relies on the degree to which SRAM power-on states change in response to stress-induced analog-domain degradation. Symmetric and asymmetric SRAM cells respond to stress differently. Stress tends to quickly impact the power-on state of symmetric cells because they are not strongly biased towards any logic state. These cells have inverters with similar analog-domain properties, and it is easier to increase relative differences between them by exposing the cell to stress. On the other hand, variance in the manufacturing process makes some cells extremely biased towards a logic state, limiting our ability to direct the power-on state in the opposite direction. The manufacturing mismatch between the inverters can be so large that stress-induced degradation fails to overcome such bias and leaves the power-on state unchanged after encoding; this is the main error source.

We elaborate on this with an example using Figure 2a. Let us assume that $|v_{th_2}| > |v_{th_4}|$. If all other parameters (*e.g.,* aspect ratio, doping density, etc.) are the same for M2 and M4, the cell powers up with 1 because M4 wins the hardware race condition. Suppose the cell is stressed with this logic state (node A = 1) for a long time. In that case, we expect the cell to eventually flip in a future power-on state. This implies that there is a flip in the threshold voltage inequality (*i.e.,* $|v_{th_2}| < |v_{th_4}|$) or at least the gap is reduced to a reasonable range so that across trials, the majority power-on state is 0. Although most cells respond to such stress, a few cells keep their power-on state unchanged even after exposure to stress for a long period. As a result, we cannot direct power-on states of all the cells to our desired direction because of extreme manufacturing biases (*e.g.,* $|v_{th_2}| >> |v_{th_4}|$), which fundamentally sets the error floor of *Invisible Bits.*

Stress conditions and duration decide the magnitude of error in *Invisible Bits.* In Figure 6, we plot the mean errors of five devices along with maximum and minimum. We keep the stress condition the same for each of the devices to study the effect of stress time on the error magnitudes. As shown in the figure, the error rate follows a logarithmic relation with time; therefore, achieving lower error requires exponentially longer time.

[5] Most of our tested devices are from ARM as these are popular in mobile devices and provide a standard debug port to access internal memories across different architectures. However, we test devices from different silicon vendors as SRAM aging responses vary depending on the manufacturing technology.

[6] We use the device datasheet to guide our search for the stress voltage and temperature.

Table 1: A list of tested devices.

Devices	CPU core	On-chip SRAM size	On-chip Flash size	Access to power-on state	Accelerated aging	Manufacturer
MSP430G2553 [20]	MSP430 single cycle	0.5KB	16KB	✓	✓	Texas Instruments
MSP432P401 [21]	ARM Cortex-M4	64KB	256KB	✓	✓	Texas Instruments
EFM32WG990F256 [42]	ARM Cortex-M4	32KB	256KB	✓	✓	Silicon Labs
ATSAML11E16A [32]	ARM Cortex-M23	16KB	64KB	✓	✓	Microchip
M263KIAAE [10]	ARM Cortex-M23	96KB	512KB	✓	✓	Nuvoton
M2351SFSIAAP [8]	ARM Cortex-M23	96KB	512KB	✓	✓	Nuvoton
M252KG6AE [9]	ARM Cortex-M23	32KB	256KB	✓	✓	Nuvoton
M251SD2AE [9]	ARM Cortex-M23	12KB	64KB	✓	✓	Nuvoton
R7FS1JA783A01CFM [13]	ARM Cortex-M23	32KB	256KB	✓	✓	Renesas Electronics
STM32L562 [46]	ARM Cortex-M33	40KB	256KB	✓	✓	STMicroelectronics
LPC55S69JBD100 [34]	Dual-core ARM Cortex-M33	320KB	640KB	✓	✓	NXP Semiconductors
BCM2837 (RPi3) [36]	Quad-core ARM Cortex-A53	L1:256KB, L2:512KB	0KB	✓	✓	Broadcom

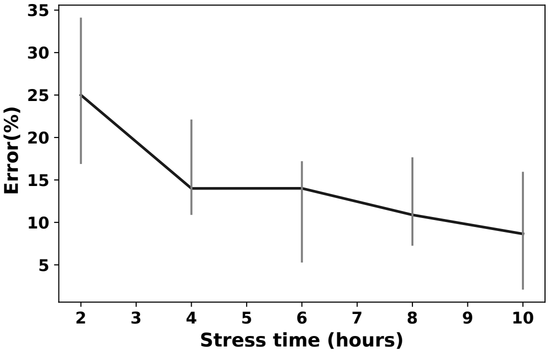


Figure 6: Influence of stress time on error.

5.1.2 *Spatial distribution.* The distribution of errors decides what type of error correction is needed in the system to get the desired accuracy. We calculate *Moran's I* to study the spatial distribution of errors, which allows us to determine whether the error is randomly distributed or bursty. A *Moran's I* statistic close to zero indicates that error is spatially random and a value further away from 0 indicates non-random patterns, closer to 1.0 indicates a positive correlation, and closer to −1.0 indicates a negative correlation.

To quantify the spatial distribution of errors, we write all-1s to one SRAM and all-0s to another SRAM, then expose them to accelerated aging. Table 2 lists the spatial autocorrelation of unstressed and stressed SRAMs. For the stressed condition, the autocorrelation is of errors, since we write a single value to all of SRAM. The results of this analysis say that the location of errors is nearly randomly distributed as the autocorrelation approaches the expected value of $-1/(N-1)$, where N is the number of error locations.

Table 2: Spatial autocorrelation of power-on states of two SRAMs before and after stress. We stress each SRAM with one logic value (0/1); therefore, the post-stress power-on state reflects spatially-correlated errors in encoding.

Stress condition	SRAM	Spatial autocorrelation	p-value
Unstressed	1	0.011	0.00
	2	0.009	0.00
Stressed	1 (Stress logic = 1)	0.005	0.00
	2 (Stress logic = 0)	0.004	0.00

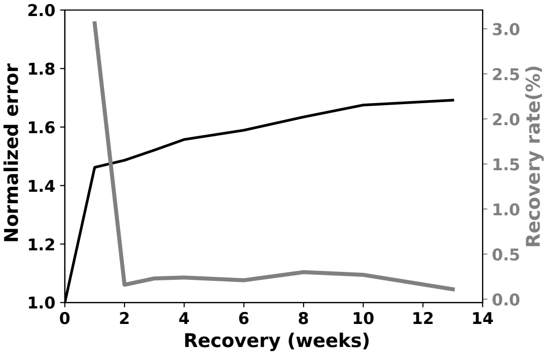


Figure 7: Normalized errors in *Invisible Bits* when stress-induced degradation recovers naturally.

5.1.3 *Effect of natural recovery.* Aging-induced analog-domain changes partially recover, which increases errors in the retrieved payload. We study the effect of natural recovery when a device is inactive during covert communication, assuming that the message needs to stay in the analog domain for a long time.

Most of the transistors in a circuit retain their stress-induced degradation of analog-domain properties [41]. Some transistors, however, partially recover when stress is released [56], which eventually affects the extraction accuracy of *Invisible Bits*. To study the effect of natural recovery on the error, we shelve an encoded SRAM and sample the power-on state at 7-day intervals. The results, shown in Figure 7, show that recovery follows a logarithmic relation with time [23], which gets reflected in *Invisible Bits*'s error rate. We plot the change in error rate as well to show that the recovery rate decays exponentially with time. The error increases ≈ 1.6x times after one month, which still keeps the error within 10%.

5.1.4 *Effect of normal operation.* The threat model allows the regular operation of a device without restriction. To evaluate the effect of regular device operation, we design software that continuously writes values to SRAM. To ensure fairness and all possible software, we write a pseudo-random sequence of words to SRAM. The pseudo-random number generator uses a 32-bit linear feedback shift register tailed by a linear congruential generator (from *glibc*, $x_{n+1} = 1103515245 \times X_n + 12345\, mod\, 2^{31}$) as seed generator to avoid repetition of numbers in our long-running experiment. We shelve an encoded device operating at nominal conditions (1.2V and room temperature) for a week.

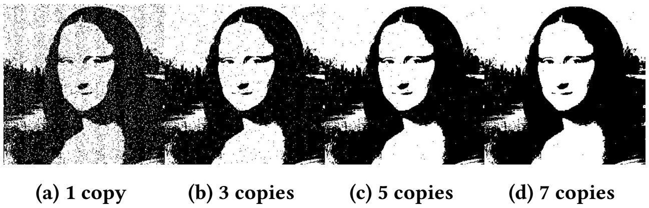


Figure 8: Visual example of how a repetition code removes error from the decoded message.

We observe a ≈ 1.2x increase in the error after a week-long recovery, which is less than what we observe in the natural recovery experiment (≈ 1.4x). While this seems counter-intuitive, it makes sense when you consider that half the time, the original stress value is being reinforced. Additionally, there is no risk of significantly stressing the opposing inverter because regular operation for a short time period does not significantly impact age transistors (see Figure 3d).

5.2 Reducing Error

Invisible Bits allows users to layer on an error-correcting code to further reduce error beyond what is feasible with encoding and recovery. There is a rich history of error-correcting codes, many targeting specific error patterns. To provide guidance on the most suitable ECC algorithms, we analyze the error rate and patterns from earlier experiments.

Our analysis shows that the errors in *Invisible Bits* are largely randomly located. Randomly distributed errors are the worst-case for error correction. Depending on error rate, the conventional approaches for dealing with randomly distributed errors are a repetition code, for high error rates, and Hamming codes, for low error rates.

Since our mean error is 10% (§5.1.3), we must start with a repetition code. A repetition code encodes several copies of the payload in SRAM and uses majority voting to determine a representative payload. Having randomly distributed errors allow us to estimate the effect as repetition code Bernoulli trials. Assuming the probability of success is p, n trials reduces the error according to Equation 1. For example, 10% error becomes 2.8% when three copies are encoded.

$$Error = 1 - \sum_{i=\frac{(n+1)}{2}}^{n} \binom{n}{i} \times p^{i} \times (1-p)^{n-i} \tag{1}$$

The error performance of *Invisible Bits* increases logarithmically as we increase the number of copies (see Figure 8). To quantify this effect, Figure 9 shows how we explore the effect of stress time and the number of copies as means for error reduction. We encode a payload into an SRAM at three two-hour-long stress cycles. The payload is replicated into many copies so that we can use additional bits to decode a 1-bit message using majority voting. We see that both stress time and increased copies reduce the error rate. The immediate repercussion of using multiple copies is the reduction in capacity. Increasing the number of copies provides diminishing returns on error reduction at the cost of reduced capacity.

Once the error rate is low enough, more efficient error correction codes are available. We illustrate this idea with experimental results where a Hamming code is added to the repetition code. Grouping bits into bins to decide the correct codeword is, in fact, a version of Hamming code. For instance, a 3-bit codeword of repeated 1-bit information is essentially a *Hamming(3,1)* with 000 and 111 valid codewords. In such error correction code, any codeword with Hamming weight 1 or 0 gets corrected into logic 0, otherwise, it is a logic 1. To show how a repetition and Hamming code work together, we add a Hamming(7,4) code on top of up to 17 copies of the payload. At the decoding phase, we calculate errors in each copy of the retrieved *payload* without any error correction, which gives us 6.5% mean error with 0.68% standard deviation. We apply this error in Equation 1 to calculate theoretical errors when we use multiple bits to decode the majority payload (the blue line in Figure 10). We observe that the *repetition code* closely follows theoretical predictions. Only *repetition code* itself brings the error to an absolute zero with 13 copies. The dotted black line in Figure 10 shows a Hamming(7,4) code combined with repetition code works. The combined codes work more efficiently because the combination drastically reduces the error within fewer copies, resulting in increased capacity.[7]

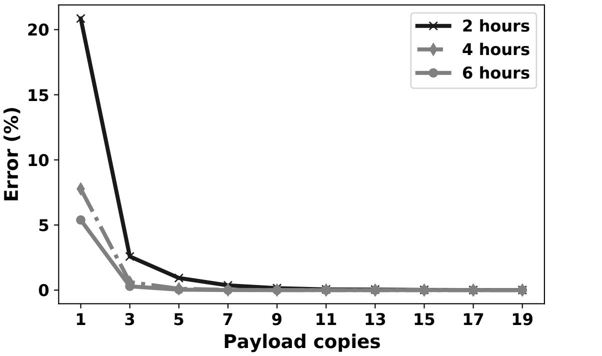


Figure 9: Error reduction due to changing stress time and the number of copies used in the repetition code.

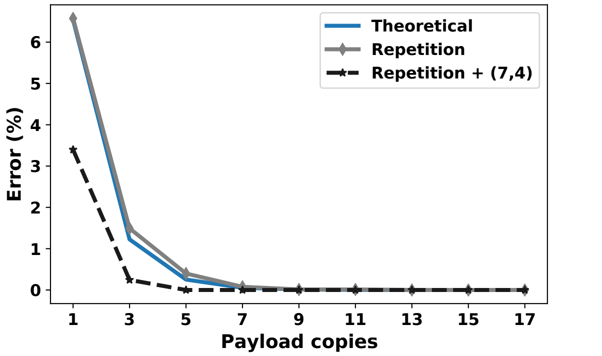


Figure 10: Application of Hamming(7,4) and repetition codes show how we can improve error performance of a message hiding scheme that uses *Invisible Bits*.

[7] Experiments suggest that the order of ECCs (repetition and hamming(7,4)) does not significantly affect the overall error rate.

Table 3: Comparison among on-chip information hiding techniques.

Method	Ubiquity	Capacity	Resilience	Read stable
Zuck *et al.* [57]	◓	●	●	◎
Wang *et al.* [52]	◓	●	◓	◎
Invisible Bits	◎	◎	◎	◎

5.3 Performance Comparison

In this section, we show how *Invisible Bits* performs compared to Flash-based message hiding schemes. The noisiness of the Flash's analog characteristics is the source of plausible deniability. Occupying the entire Flash to store confidential information reveals its existence. Flash-based information hiding schemes suffer from low capacity because they allow only a few hundred cells per page to store hidden information. Besides, programming all the pages is challenging because encoded information in one page interferes with the neighboring pages' analog characteristics. As a result, they cannot select every page to encode information.

Additionally, Flash-based methods require grouping bits into a large bin (128-bit or more) to achieve low error. Considering these limiting factors, a Flash-based hiding scheme achieves 0.05% capacity [52]. When considering that on-chip Flash memories are usually is reserved for program data, capacity is reduced further. For example, consider an MSP432P401 microcontroller that has 256KB of Flash. Assuming that the entire Flash is available, write-time-based Flash hiding approaches can only transmit 131 bytes of information [52]. The more recent voltage-based technique [57] doubles this capacity by hiding information within the public data (*i.e.,* overlapping), but this assumes that the entire Flash is filled with data.

On the contrary, since encoding the payload in SRAM's analog domain does not interfere with its digital operation, *Invisible Bits* has access to the entire SRAM—no matter how software uses it. Combined with its low error rate, *Invisible Bits* more than makes up for Flash's 4:1 size advantage over SRAM. We provide a qualitative comparison between the on-chip steganography techniques in Table 3.[8] We consider an encoded device with 6.5% (§5.1) and 5-copies *repetition code* to bring down the error at the same level (<0.3%) across all approaches. Using five copies allows *Invisible Bits* to hide 12.8KB of payload (20% × 64KB), which is 100x of the Flash write-time-based method. Table 4 summarizes the errors when a hidden message contains a single copy and the settings of acceleration parameters for four devices.

Another advantage of *Invisible Bits* is that devices can be encoded in parallel. Given the importance of capacity in a steganographic covert channel, one can encode many devices and select the one with the least error. From our experimental results in Figure 6, a device with 2.7% error is possible. Such a device produces 33% information capacity with only 3 copies at less than 0.01% error rate. In such case, *Invisible Bits* hides 160x more information than Flash-based on-chip approaches. Table 3 lists qualitative comparison between *Invisible Bits* and Flash-based methods.

Table 4: Summary of the results when we encode a message in different devices.

Device	SRAM usage	$V_{acc.}$	$T_{acc.}$	Bit rate	Encoding time
ATSAML11E16A	Main memory	4.8V	$85^{\circ}C$	97.2%	16 hours
MSP432P401	Main memory	3.3V	$85^{\circ}C$	93.5%	10 hours
LPC55S69JBD100	Main memory	5.5V	$85^{\circ}C$	88.5%	24 hours
BCM2837	Cache	2.2V	$85^{\circ}C$	79.2%	120 hours

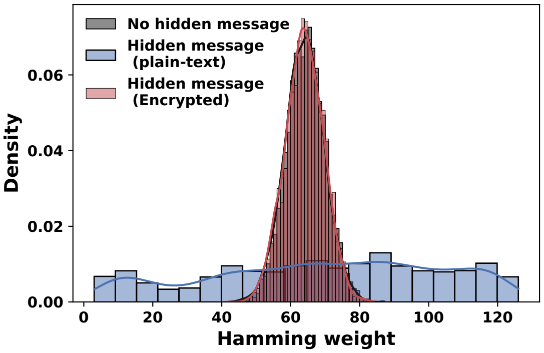


Figure 11: Hiding information as plain-text, and ciphertext.

6 INVISIBLE BITS IN ACTION

Invisible Bits is a high capacity on-chip covert channel that hides messages in the analog domain of SRAM. An immediate advantage of decoupling the message from regular computing space is digital plausible deniability. In this section, we expand our threat scenario and consider that an adversary inspects SRAM's power-on state as a medium for information hiding. We show that using a stream cipher eliminates such threat and provides robust defense against analog-domain plausible deniability compromise. Then, we show an example of a covert and resilient end-to-end message hiding scheme using *Invisible Bits.*

Hiding information affects the symmetric distribution of an SRAM power-on state (Figure 3a), revealing the existence of a hidden message. We hide a message in an SRAM with error correction using both Hamming(7,4) and *repetition code.* In Figure 11, we plot the post-encoded Hamming weight distributions of the power-on states of the devices. The distributions of hamming weights reveal the non-ideal distribution of SRAM power-on state, therefore, violates the analog-domain plausible deniability.

We attain analog-domain plausible deniability by using a stream cipher because it destroys the message's structural properties, while being error neutral (*i.e.,* 1-bit error in the plain-text affects the 1-bit error in the ciphertext). We use AES-CTR to encrypt messages before encoding into an SRAM because standard AES encrypts a message and distributes the message bit uniformly. The counter mode allows decryption even if the ciphertext is partially corrupted due to error.

While the design remains the same as discussed in Section 4, we illustrate the application of *Invisible Bits* in steganography using a specific system shown in Figure 13. First, we apply a Hamming(7,4) on a message d and replicate the message and parity seven times to make it a hamming(7,1) code ($x = ECC(d)$). Second, x contains

[8] ◎ = Excellent, ◓ = Very good, ○ = Good, ◒ = Fair, and ● = Poor.

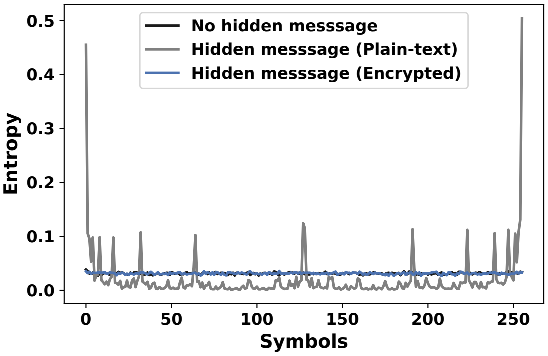


Figure 12: Shannon's entropy of SRAMs' power-on state. The normalized (by the number of symbols) entropy of an SRAM's power-on state is 0.0312, and the entropy becomes 0.0195 and 0.0312 in SRAMs that hide message as plain-text and encrypted text, respectively.

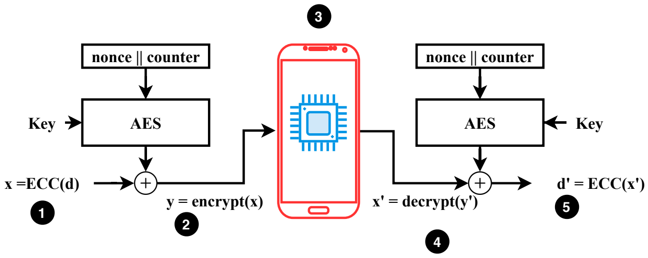


Figure 13: An end-to-end steganography system using *Invisible Bits.*

the data and parity bits, and AES converts these bits into a *payload* ($y = encrypt(x)$). Third, we store the *payload* into the SRAM of an MSP432P401 microcontroller. Fourth, SRAM undergoes the encoding process for 10 hours (§5), which changes the analog properties of transistors and, hence, influences the SRAM's power-on state.

The *Payload* extraction process follows the steps described in Section 4. The process begins with extracting the encoded SRAM's power-on states and using a shared key to decrypt the payload ($x' = decrypt(y')$). Finally, ECC retrieves the message (d'), which completes the steganographic message transfer.

We analyze the post-encode power-on state's properties to show that it is indistinguishable from an SRAM with no hidden message.

First, we run a hypothesis test on the post-encoded power-on distributions. We apply Welch's t-test on the sample of two classes of devices, with the null hypothesis being the chips have no hidden messages (*i.e.,* identical mean Hamming weight). We use a p-value heuristic to determine the significance of the t-statistic. Our test's p-value is 0.071 (one-tailed); therefore, we cannot reject the null hypothesis. From this, we conclude that an adversary cannot statistically differentiate a chip with a hidden message from a chip with no hidden message (See Figure 11).

Second, we study the spatial autocorrelation to investigate whether an encoded device shows any difference compared to a device without any hidden message. In Table 5, we list spatial autocorrelation, the power-on biases for three classes of chips. We observe that chips without any encryption show spatial non-random patterns. On the other hand, the chips with encrypted hidden messages have similar *Moran's I* statistic and mean power-on bias.

Table 5: Spatial autocorrelation and mean power-on bias of SRAM embedded within an MSP432. Layering encryption on top of *Invisible Bits* makes devices with no hidden message indistinguishable from devices with messages encoded. Without encryption, devices with messages encoded have biases in their power-on state (e.g., more 1s than 0s) and spatially-correlated values. p-values are $<<$ 0.05 for all the measurements.

Condition	Spatial autocorrelation	Mean power-on bias
Hidden message (no encryption)	0.502	0.537
Hidden message (no encryption)	0.409	0.535
No hidden message	0.001	0.500
No hidden message	0.004	0.501
No hidden message	0.002	0.501
No hidden message	0.007	0.502
No hidden message	0.009	0.500
Hidden message (encrypted)	0.001	0.500
Hidden message (encrypted)	0.008	0.501
Hidden message (encrypted)	0.002	0.499
Hidden message (encrypted)	0.001	0.499

Third, we apply Shannon's entropy, a widely used metric to quantify uncertainty in information [26], in both pre- and post-encoded SRAM's power-on states. The entropy of a bit varies between 0 and 1; 0 being no uncertainty and 1 being fully random (*e.g.,* fair coin flip). We divide the power-on state of an SRAM into byte granularity (symbol) and count the frequency of each 2^8 symbols (similar to earlier work [17]). Analyzing 64K (size of the SRAM is 64KB) symbols from an SRAM's power-on state, we form a distribution and calculate the Shannon's entropy using $\sum_{i=0}^{n} -P(x_i) \times log_2(P(x_i))$. Here, $P(x_i)$ is the probability of i^{th} possible value out of n symbols. We plot the distribution of entropy in Figure 12 calculated for SRAMs with no hidden message, hidden message as plain-text, and encrypted hidden message. The entropy distribution of an SRAM that hides a message as plain-text deviates from the regular SRAM's power-on state entropy, whereas an encrypted hidden message closely resembles it.

Upon analyzing different statistical properties, we conclude that chips that encode encrypted messages are indistinguishable from 'clean' chips. Once the process is completed, the device containing SRAM becomes a covert and resilient information carrier with an ability to withstand both analog and digital domain steganalysis.

7 DISCUSSION

7.1 Multiple-Snapshot Adversary

We assess a potential plausible deniability compromise by considering a stricter threat model where an adversary takes multiple power-on measurements (*i.e.,* snapshots) at different times to determine temporal discrepancies in the device's power-on state.

Our recovery experiments show that some cells return to their original power-on values, and this recovery trend is greatest in the early post-encoded period (see Figure 7). We study the effect of capturing power-on states at different times in the recovery phase with the greatest power-on state change to assess whether it reveals the existence of an encoded message. The analysis starts with

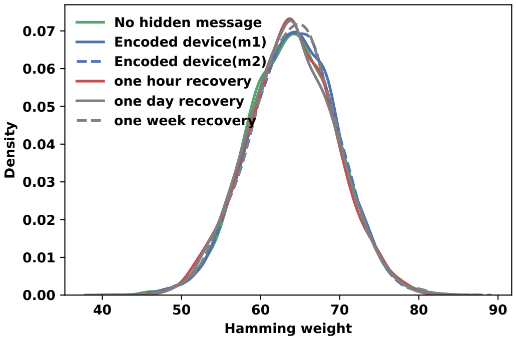


Figure 14: Distribution of Hamming weights captured at different phases and times of a covert communication.

capturing the power-on states of a pre-encoded and post-encoded device. Then, we compare these measurements to the power-on states captured after one hour, one day, and one week of encoding. We calculate the spatial autocorrelation <0.01 with p-value << 0.05 for all of the measurements. This suggests that the spatial pattern of errors remains indistinguishable from an unencoded device during encoding and during recovery. The second statistic that would potentially reveal an encoded message is the distribution of Hamming weights for the SRAM when adjacent cells are grouped into fixed-size blocks. Figure 14 shows the distribution of Hamming weights for the measurements, where m1 and m2 indicate two back-to-back measurements of the same device. We expect some difference, even from subsequent measurements, as the cells with similar inverter turn-on voltages in an SRAM introduce noise in the measurement of power-on states (§2). Following a similar analysis discussed in Section 6, we conclude that the difference in the snapshots captured at multiple points in time is indistinguishable from measurement errors. Thus, a multiple-snapshot adversary gains no special advantage in *Invisible Bits.*

7.2 Aging Complex Systems

High voltage plays a major role in accelerated aging (§2), and applying such voltage in a complex device requires special consideration on its power supply system. When a device is simpler such as MSP430, we directly apply voltage to its V_{dd} line. Complex devices (*e.g.,* Raspberry Pi), however, regulate voltage before feeding it to the core (and SRAM). These circuits prevent elevating the supply voltage directly from the V_{dd} line. Our observation is that most devices use switching regulators at run-time to save energy because these regulators are inherently efficient in switching voltage levels. By design, these regulators need large passive components such as inductors. Since these passive components have a large footprint, the most practical design decision is to place them in the PCB and use a pin to connect it with the internal supply line. We exploit this pin to reach the core supply line directly and elevate the core voltage.

7.3 Device Selection & Capacity Considerations

The required capacity and the least tolerable error in the reconstructed message guide the device selection for *Invisible Bits.* While these two requirements are somewhat inversely proportional, the *baseline error rate* is determined largely by the SRAM's technology node. The manufacturing technology node determines the maximum allowable stress level, which eventually determines the minimum noise in the hidden message. SRAM manufactured at smaller technology nodes are more susceptible to aging as the electric field density in these devices is higher as compared to larger devices [24]. Thus, newer devices that can endure higher voltage and temperature for longer time provide the lowest error rate, which makes the most of the available SRAM cells. That being said, desired capacity is the driver behind device selection.

Finding average capacity in each device class requires testing a large sample of devices with full system implementation, including appropriate ECC selection for a message. We provide a theoretical analysis to illustrate the capacity and error trade off using Table 4 and augmenting it with ECC (*i.e.,* repetition codes and Hamming (7,4)) to calculate errors at different capacity-levels. Figure 15 shows how error rate affect the capacity across device classes.

Regardless of their locations, Flash memories tend to be larger memory devices compared to SRAMs. When we compare on-chip SRAM with off-chip Flash memories, the size difference is orders of magnitude larger. While density of hidden information remains higher for SRAM-based steganography, Flash-based methods hide more information when off-chip memories are considered because inherently Flash-based storage media, such as SSDs, are much larger than on-chip SRAMs. Being external enables Flash to have high capacity, but it also makes it trivially separable from any system that it is attached to.

7.4 Adversarial Aging to Inject Noise

In this section, we discuss a scenario where an adversary attempts to erase or inject noise in the hidden message in a device's analog domain by exposing the device to aging. It is possible to inject noise in the hidden message as aging-induced degradation has both reversible (NBTI) and irreversible (HCI) components. While it portrays an unlikely scenario where authorities (*e.g.,* border security) expose a large sample of devices to invasive aging for a prohibitively long time, we explore this possibility with an experiment to

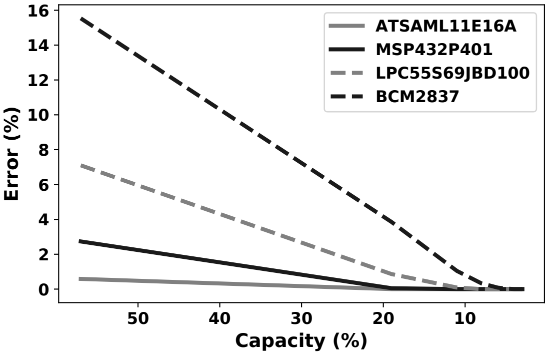


Figure 15: Error and capacity trade off. We encode a single copy message in each device and simulate Bernoulli trials for different payload copies and Hamming(7,4) error correction code.

understand the impact of such adversarial aging. In this experiment, we expose an encoded MSP432 to aging with a power-on state for 1 hour. This increases the message reconstruction error 1.12×. When aged again for 1.5 hours with the power-on state, the error comes to its pre-adversarial aging state (error 0.98×). Cells with symmetric inverters are more susceptible to aging compared to asymmetric cells, and aging for short times injects noise by changing the power-on state of the symmetric cells. The receiving party can reduce the impact of noise by aging it in a similar way.

This scenario above considers an adversary who overcomes the following barriers to inject noise on a device that *may* contain a hidden message in SRAM. First, aging a device involves physical tampering: an adversary needs to open up a device, decouple its power supply system, and externally elevate the voltage of the core (§7.2). Second, since *Invisible Bits* encodes information in the analog domain by exploiting *physical* changes in transistors, reversing the aging-induced degradation is *prohibitively time-intensive.* Third, the next challenge is to improve the effectiveness of adversarial aging, *i.e.,* maximizing the injected noise. As discussed in Section 2, aging pushes the power-on state of a cell to the complement of the stored logic state, and therefore aging an encoded SRAM with a power-on state injects the maximum number of noise bits. Setting a device in this state (*i.e.,* SRAM with its power-on state) *requires firmware tampering* because as soon as a device boots, the system software clobbers the power-on state of its SRAMs.

8 RELATED WORK

The closest related work to this paper is Flash program time-based message hiding method [52]. This method deliberately stresses a group of cells to encode information in them. The program time of cells is distributed over a long-tailed spectrum, and so changing a few cells' program time becomes indistinguishable from non-encoded cells. A group of 128-bit cells encodes 1-bit information, and addresses of the cells that are grouped are encrypted using a symmetric key cipher. Zuck *et al.* [57] uses the voltage level of a flash memory cell as the medium for hiding information. The technique uses two passes to encode a logic state. The first pass stores encrypted cover data, and the second pass selects a few cells from the same public bits, preferably logic 1, to encode hidden data. To encode 0, a non-programmed bit, cells currently holding public data are incrementally charged beyond their preset voltage level. As long as the cover data is not erased or re-programmed, the hidden data remains stored, meaning—unlike *Invisible Bits*—the encoded data is fragile. An active adversary can promptly stop covert communication by copying the encrypted cover data and re-programming it again without modification. In such a case, data is lost. Another limitation of this approach is that it is not plausibly deniable at the digital level as the encrypted cover data signals that the device carries some secret message, even if the actual message is in the analog domain. On top of these limitations, both methods suffer from low capacity when compared to the *Invisible Bits*, because they have to deal with relatively large operational noise margins, where the tightly-packed nature of SRAM cells filters operational noise as common-mode noise.

Srinivasan *et al.* exploit Flash transport layer (FTL) and over-provisioning of solid-state drives to hide information in the physical layer of a mass storage [45]. DEFY introduces multiple security levels in the data handling of FTL [35]. The existence of higher security level data is not visible to the lower security level. Therefore, a victim can release the lower security level data to deny higher security level data. These methods suffer a general problem, unintentional overwriting, that stems from the coexistence of public and private information [7, 43, 44]. Even worse, Jia *et al.* [22] shows that adversaries can detect hidden messages in the lower-level security and copy the data and rewrite it back.

9 CONCLUSION

This paper uncovers an on-chip steganographic medium based on SRAM's analog-domain properties, namely the relative threshold voltage of the cross-coupled inverters that compose every SRAM cell. To exploit this new covert channel, we design and implement *Invisible Bits*, a steganographic system that uses accelerated, directed transistor aging to encode information into SRAM's analog-domain properties and reveals these changes at the digital level by capturing SRAM's power-on state. Our evaluation, using modern computing devices, shows that *Invisible Bits* increases capacity to over 90%, a 100x improvement over existing on-chip steganographic techniques—while retaining both digital- and analog-domain plausible deniability. In addition to improved capacity, our evaluation also shows that *Invisible Bits* improves resilience as it maintains the vast majority of the encoded data after being shelved for a month and even after being used continuously for over a week. Lastly, we use the results of our evaluation to guide future users on the most appropriate error-correcting codes and ciphers to layer on top of *Invisible Bits*.

ACKNOWLEDGMENTS

We thank our shepherd Trevor E. Carlson for his guidance and the anonymous reviewers for their helpful suggestions. The project depicted is sponsored by the Defense Advanced Research Projects Agency. The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. Approved for public release; distribution is unlimited.

REFERENCES

[1] Wikipedia Article. 2020. Key disclosure law. https://en.wikipedia.org/wiki/Safety_of_journalists.

[2] ASU. 2020. Predictive Technology Model (PTM). http://ptm.asu.edu/.

[3] Marta Bagatin, Simone Gerardin, Alessandro Paccagnella, and Federico Faccio. 2010. Impact of NBTI aging on the single-event upset of SRAM cells. *IEEE Transactions on Nuclear Science* 57, 6 (2010), 3245–3250.

[4] Austen Barker, Staunton Sample, Yash Gupta, Anastasia McTaggart, Ethan L Miller, and Darrell DE Long. 2019. Artifice: A deniable steganographic file system. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19).*

[5] Erik-Oliver Blass, Travis Mayberry, Guevara Noubir, and Kaan Onarlioglu. 2014. Toward robust hidden volumes using write-only oblivious RAM. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 203–214.

[6] Hristo Bojinov, Daniel Sanchez, Paul Reber, Dan Boneh, and Patrick Lincoln. 2012. Neuroscience meets cryptography: designing crypto primitives secure against rubber hose attacks. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12).* 129–141.

[7] Bing Chang, Zhan Wang, Bo Chen, and Fengwei Zhang. 2015. Mobipluto: File system friendly deniable storage for mobile devices Proceedings of the 31st Annual Computer Security Applications Conference. *Los Angeles, CA, USA, December* (2015).

[8] Nuvoton Technology Corporation. 2019. NuMicro®Family M2351 Series Datasheet. https://www.nuvoton.com/export/resource-files/DS_M2351_Series_EN_Rev1.01.pdf.

[9] Nuvoton Technology Corporation. 2020. NuMicro®Family M251/M252 Series Datasheet. https://www.nuvoton.com/export/resource-files/DS_M251_M252_Series_EN_Rev1.01.pdf.

[10] Nuvoton Technology Corporation. 2020. NuMicro®Family M261/M262/M263 Series Datasheet. https://www.nuvoton.com/export/resource-files/DS_M261_M262_M263_Series_EN_Rev1.02.pdf.

[11] CPJ. 2020. 10 Most Censored Countries. https://cpj.org/reports/2019/09/10-most-censored-eritrea-north-korea-turkmenistan-journalist/.

[12] Xintao Duan, Liu Nao, Gou Mengxiao, Dongli Yue, Zimei Xie, Yuanyuan Ma, and Chuan Qin. 2020. High-Capacity Image Steganography Based on Improved FC-DenseNet. *IEEE Access* 8 (2020), 170174–170182.

[13] Renesas Electronics. 2019. Renesas Synergy™ Platform Synergy Microcontrollers S1 Series. https://www.renesas.com/us/en/document/man/s1ja-microcontroller-group-users-manual.

[14] The Guardian. 2020. Chinese border guards put secret surveillance app on tourists' phones. https://www.theguardian.com/world/2019/jul/02/chinese-border-guards-surveillance-app-tourists-phones.

[15] S. Han, B. S. Kim, and J. Kim. 2013. Variation-Aware Aging Analysis in Digital ICs. *Transactions on Very Large Scale Integration Systems* 21, 12 (Dec. 2013), 2214–2225. https://doi.org/10.1109/TVLSI.2012.2228886

[16] Helena Handschuh. 2012. Hardware-anchored security based on SRAM PUFs, part 1. *IEEE Security & Privacy* 10, 3 (2012), 80–83.

[17] D. E. Holcomb, W. P. Burleson, and K. Fu. 2009. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Trans. Comput.* 58, 9 (Sept. 2009), 1198–1210. https://doi.org/10.1109/TC.2008.212

[18] Daniel E. Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P. Burleson, and Kevin Fu. 2012. DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification. In *Proceedings of the 8th Intl. Conference on Radio Frequency Identification: Security and Privacy Issues (RFIDSec).* 165–179.

[19] Daniel E. Holcomb, Amir Rahmati, Mastooreh Salajegheh, Wayne P. Burleson, and Kevin Fu. 2013. DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification. In *Proceedings of the 8th Intl. Conference on Radio Frequency Identification: Security and Privacy Issues (RFIDSec'12).* Springer-Verlag, Berlin, Heidelberg, 165–179. https://doi.org/10.1007/978-3-642-36140-1_12 event-place: Nijmegen, The Netherlands.

[20] Texas Instruments. 2014. MSP430G2x53 Automotive Mixed-Signal Microcontrollers. https://www.ti.com/lit/ds/symlink/msp430g2553-q1.pdf?ts=1618423366624&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430G2553-Q1.

[21] Texas Instruments. 2019. SimpleLink^{TM} ultra-low-power 32-bit Arm Cortex-M4F MCU With Precision ADC, 256KB Flash and 64KB RAM. https://www.ti.com/lit/ds/symlink/msp432p401r.pdf?ts=1604876136889&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP432P401R.

[22] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. 2017. DEFTL: Implementing plausibly deniable encryption in flash translation layer. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 2217–2229.

[23] K. Kang, H. Kufluoglu, K. Roy, and M. Ashraful Alam. 2007. Impact of Negative-Bias Temperature Instability in Nanoscale SRAM Array: Modeling and Analysis. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26, 10 (Oct. 2007), 1770–1781. https://doi.org/10.1109/TCAD.2007.896317

[24] Naghmeh Karimi, Thorben Moos, and Amir Moradi. 2019. Exploring the effect of device aging on static power analysis attacks. *UMBC Faculty Collection* (2019).

[25] Saman Kiamehr, Mohammad Saber Golanbari, and Mehdi B Tahoori. 2017. Leveraging aging effect to improve SRAM-based true random number generators. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.* IEEE, 882–885.

[26] Jeremie S Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. 2019. D-RaNGe: Using commodity DRAM devices to generate true random numbers with low latency and high throughput. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 582–595.

[27] Robert R. King. 2019. North Koreans Want External Information, But Kim Jong-Un Seeks to Limit Access. https://www.csis.org/analysis/north-koreans-want-external-information-kim-jong-un-seeks-limit-access.

[28] R. Maes and V. van der Leest. 2014. Countering the effects of silicon aging on SRAM PUFs. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).* 148–153. https://doi.org/10.1109/HST.2014.6855586

[29] Heather Mahalik. 2020. 5 Questions That Will Unlock More Data with checkm8. https://www.cellebrite.com/en/blog/iphone-extractions-5-questions-to-unlock-more-data/.

[30] Andrew D McDonald and Markus G Kuhn. 1999. StegFS: A steganographic file system for Linux. In *International Workshop on Information Hiding.* Springer, 463–477.

[31] Joseph W McPherson. 2006. Reliability challenges for 45nm and beyond. In *2006 43rd ACM/IEEE Design Automation Conference.* 176–181.

[32] Microchip Technology Inc. 2020. SAM L10/L11 Family: Ultra Low-Power, 32-bit Cortex-M23 MCUs with TrustZone, Crypto, and Enhanced PTC.

[33] J Mull. 2012. How a Syrian refugee risked his life to bear witness to atrocities. *Toronto Star Online, posted* (2012).

[34] NXP Semiconductors. 2019. UM11126:LPC55S6x/LPC55S2x/LPC552x User manual.

[35] Timothy M Peters, Mark A Gondree, and Zachary NJ Peterson. 2015. DEFY: A deniable, encrypted file system for log-structured storage. (2015).

[36] Raspberry Pi Foundation. 2021. Raspberry Pi 3 Model B (v1.2). https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[37] Alec Roelke and Mircea R. Stan. 2018. Controlling the Reliability of SRAM PUFs With Directed NBTI Aging and Recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 10 (2018), 2016–2026. https://doi.org/10.1109/TVLSI.2018.2836154

[38] Ulrich Ruhrmair. 2020. SoK: Towards Secret-Free Security. In *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security.* 5–19.

[39] Sachin S Sapatnekar. 2013. What happens when circuits grow old: Aging issues in CMOS design. In *2013 Intl. Symposium on VLSI Technology, Systems and Application (VLSI-TSA).* 1–2.

[40] Shikha Sharma and Devendra Somwanshi. 2016. A DWT based attack resistant video steganography. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies.* 1–5.

[41] Jeonghee Shin, Victor Zyuban, Pradip Bose, and Timothy M Pinkston. 2008. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. *ACM SIGARCH Computer Architecture News* 36, 3 (2008), 353–362.

[42] Silicon Labs. 2012. EFM32 Wonder Gecko Family EFM32WG Data Sheet. https://www.silabs.com/documents/public/data-sheets/efm32wg-datasheet.pdf.

[43] Adam Skillen and Mohammad Mannan. 2013. Mobiflage: Deniable storage encryptionfor mobile devices. *IEEE Transactions on Dependable and Secure Computing* 11, 3 (2013), 224–237.

[44] Adam Skillen and Mohammad Mannan. 2013. On implementing deniable storage encryption for mobile devices. (2013).

[45] Avinash Srinivasan, Jie Wu, Panneer Santhalingam, and Jeffrey Zamanski. 2014. DeadDrop-in-a-Flash: Information Hiding at SSD NAND Flash Memory Physical Layer. *SECURWARE 2014* (2014), 79.

[46] STMicroelectronics. 2020. Ultra-low-power Arm® Cortex-M33 32-bit MCU+TrustZone®+FPU, 165DMIPS, up to 512KB Flash, 256KB SRAM, SMPS, AES+PKA. https://www.st.com/resource/en/datasheet/stm32l562ce.pdf.

[47] TestEquity. 2020. Model 123H Temperature/Humidity Chamber. https://www.testequity.com/category/Environmental-Chambers-Ovens/Temperature-Humidity-Chambers/TestEquity-123H-Temperature-Humidity-Chamber-North-America-Version-17267-1.

[48] B. Tudor, Joddy Wang, W. Liu, and Hany Elhak. 2011. MOS Device Aging Analysis with HSPICE and CustomSim.

[49] United States of America Central Intelligence Agency. 2016. TRIGON: Spies Passing in the Night. https://www.cia.gov/news-information/featured-story-archive/2016-featured-story-archive/trigon-spies-passing-in-the-night.html.

[50] United States of America Customs and Border Protection Agency. 2017. CBP Statement on Border Search of Electronic Devices. https://www.cbp.gov/newsroom/national-media-release/cbp-releases-statistics-electronic-device-searches-0#wcm-survey-target-id.

[51] United States of America Customs and Border Protection Agency. 2019. CBP Statement on Border Search of Electronic Devices. https://www.cbp.gov/newsroom/speeches-and-statements/cbp-statement-border-search-electronic-devices.

[52] Y. Wang, W. Yu, S. Q. Xu, E. Kan, and G. E. Suh. 2013. Hiding Information in Flash Memory. In *2013 IEEE Symposium on Security and Privacy.* 271–285. https://doi.org/10.1109/SP.2013.26

[53] Dipti Watni and Sonal Chawla. 2019. A Comparative Evaluation of JPEG Steganography. In *2019 5th International Conference on Signal Processing, Computing and Control (ISPCC).* IEEE, 36–40.

[54] Yunzhao Yang, Yuntao Wang, Xiaowei Yi, Xianfeng Zhao, and Yi Ma. 2019. Defining joint embedding distortion for adaptive MP3 steganography. In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security.* 14–24.

[55] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials* 9, 3 (2007), 44–57.

[56] Rui Zheng, Jyothi Velamala, Vijay Reddy, Varsha Balakrishnan, Evelyn Mintarno, Subhasish Mitra, Srikanth Krishnan, and Yu Cao. 2009. Circuit aging prediction for low-power operation. In *2009 IEEE Custom Integrated Circuits Conference.* 427–430. https://doi.org/10.1109/CICC.2009.5280814

[57] Aviad Zuck, Yue Li, Jehoshua Bruck, Donald E. Porter, and Dan Tsafrir. 2018. Stash in a Flash. In *16th USENIX Conference on File and Storage Technologies (FAST 18).* USENIX Association, Oakland, CA, 169–188. https://www.usenix.org/conference/fast18/presentation/zuck