Part VI
Run-time Expectations under Attack

For a piece of software to operate as expected, the hardware and lower-level software that define its run-time environment must operate expected. Assumptions are potential vulnerabilities, and expectations about a system's run-time environment are no exception. This part discusses some of those expectations, along with attacks to invalidate them. Where viable defenses are know, we discuss those too. As always, whether a given vulnerability should be concerning will depend on the threat. Not all threats have the motivation, equipment, access, and/or expertise needed for a specific attack.

Chapter 12

Hardware Integrity

Physical access to a computer's internals permits attacks that can circumvent authorization checks implemented by software or by hardware. The obvious defense is to prevent attackers from having that physical access. Some defenses, in addition, generate indications of attempted attacks. Such indications are useful, because confidentiality compromises are not always detectable.

12.1 Leveraging Location

Walls and locked doors are one way to ensure that a computer is inaccessible to attackers. You might place the computer in a locked machine room, in a (locked when unoccupied) office, or at somebody's home. For portable devices, carrying the device in your pocket or keeping it in your briefcase impedes access by depending on social norms about personal distance and who can access personal property.

When physical access to a computer cannot be blocked, we can deter attackers if they know that evidence is being created to attribute accesses they make. Surveillance could be implemented by video cameras, or surveillance could be performed in person by employees or law enforcement. Surveillance is particularly effective for deterring insiders, where walls and locked doors would not otherwise impede an attacker's actions.

Walls together with video surveillance have been used to create a *private cloud* within a cloud data center. The private cloud comprises computer racks enclosed by a cage, where the cage door remains locked both while the enclosed computers are running as well as for an additional period after those computers have been powered-off. Video surveillance of the metal cage deters people from unlocking and entering a cage while the processors or memories it encloses might be storing (unencrypted) confidential data. After power-off, the delay period prior to allowing entry ensures that remnants of confidential data in volatile memory will decay before that memory can be read by somebody who has entered the cage.

Large-scale clouds also can hinder attackers from getting physical access to the computers serving a given customer by keeping secret which computers (from the large number in a data center) are running that customer's computations at any time. Here, secrecy is replacing the walls and locked doors as the impediment to access. By periodically migrating a customer's computation from one set of hardware processors to another, a moving-target defense would also be created.

12.2 Enclosure and Construction

The location of a device is not always under our control, and surveillance is not always feasible. A set-top cable box, e-book reader, gaming console, or other consumer electronics for providing access to proprietary digital content usually will be physically accessible to its users, and some of those users could be attackers. Credit-card sized artifacts carried in wallets and used to control access to funds or locks on doors are other examples of devices that could come into the possession of attackers. For these situations, a device's construction is the first line of defense against attacks involving physical access.

Packaging. Packaging can protect a device by making it difficult for an attacker to study the internals or to operate the system while monitoring and/or injecting signals. The starting point for implementing such *tamper resistance* often will be a physical enclosure that is difficult to breach without a physical key or special tool. Since theft of information is not visible, an enclosure might also be designed to make evident that an attack has been attempted. One way to create hardware that is *tamper evident* is by using a frangible or highly finished (e.g., polished or crazed) material for the outer shell of the enclosure, so that attempting a penetration causes irreversible and visible changes to the enclosure's appearance. Unforgeable seals, made with multi-layer paints or tapes, are another way to create evidence that an attack has been attempted. Finally, a packaging might be *tamper responding* and activate circuits that erase memory (deleting cryptographic keys or other secrets) or that disable the device (perhaps even detonating a small explosive charge). But if physical access to internals could be needed for maintenance, then some means of access must be available for trusted individuals. The device, however, now becomes vulnerable to abuse by an untrustworthy insider who exploits that access.

Sensors. Sensors are the obvious starting point for making a packaging be tamper-responding. Penetration of an enclosure can be detected by having photocells inside to sense the increased level of light from the outside. Another way to detect penetrations of an enclosure is to line the enclosure with a membrane that has been printed with a pattern of conductive ink, so the electrical properties of the membrane change when the membrane is punctured. Radiation sensors and temperature sensors will detect attacks aimed at increasing memory remanence (see §12.4.1) in order to facilitate theft of secrets after a system

is powered down. Voltage sensors are useful for detecting attacks that could corrupt operation of the electronic circuitry. And attempts to transport the device elsewhere can be detected by having motion sensors. Sensors do bring challenges, however. First, they require a source of power. Second, there is a risk of false-triggering, so a system must be designed to recover from that.

Potting. An attacker will have a harder time finding and physically accessing specific electronic components and interconnecting wires if that circuitry has been embedded in a block of opaque epoxy potting. Moreover, an attacker seeking to operate a system after modifying its components or connections will be hampered if physical access to those components or wires requires first destroying other components that (by design) were positioned to be in the way.

A net of fine wires or other conductive material that surrounds the electronic circuitry before the epoxy potting is added can serve as a further barrier to attackers, if breaking, shorting, or altering any of those paths triggers circuits to erase secrets or otherwise disable the device. In addition, attackers who attempt access to interior components by using chemicals or a laser to dissolve portions of the epoxy potting will be detected if the chemical composition of the epoxy potting is more resistant to solvents and if it expands faster when heated than the material used for the embedded wires.

Means of Physical Attack. Whether a device's construction will succeed as a defense depends on the attacker's access, capabilities, and goals. Unsupervised access enables attackers to operate, disassemble, and/or alter a device. This is not necessarily changed by requiring that access be supervised by guards—guards are unlikely to intercede when an attacker is dressed to resemble a *bona fide* maintenance technician. When access is supervised, though, the length of time available for performing an attack could constrain an attacker.

If an attacker can move a device to a remote site then specialized tools can be employed.[1]

- *Machining.* Access to a device's internals is enabled by cutting through a shell or by removing potting material that permeates the insides. The cutting might be performed with (fixed or moving) blades, abrasives, high-velocity streams of water, lasers, sandblasting, or shaped (low power) explosive charges. Chemicals also can be used to dissolve potting material.

- *Probing.* Probes provide a means to inject and/or monitor signals being carried by the wires in a device. A probe might be implemented by a narrow gauge tungsten wire, an ion beam, an electron beam, or a laser. Ion beams, in addition, can reconnect fuse links or make other modifications to a chip's circuitry. Electron beams from a conventional scanning

[1] These tools are developed for the semiconductor industry to use in analyzing chips. The tools typically are expensive to purchase, but they often can be rented on an hourly basis with no questions asked. In addition, the improved tool capabilities required for each new generation of chips results in decreased demand for older tools, which then become affordable by attackers.

electron microscope can read/write bits in EPROM, EEPROM and RAM memory chips if the chip's surface has been exposed (say, by chemical etching). Because silicon is transparent at infrared frequencies (IR), it is not necessary to expose the chip's surface for an IR laser to read/write that storage.

The goal of a physical attack is an important factor when developing defenses. Some physical attacks are undertaken to extract secrets that a device is storing. A defense here could be to incorporate sensors and logic that causes stored secrets to be erased when the start of an attack is detected. The goal of other physical attacks is reverse-engineering—for cloning a system or for discovering its vulnerabilities. Packaging plays the critical role in defending against such attacks.

For some devices, an attack would be deemed a failure if it leaves a system inoperable. An attack that that renders a nuclear weapon inoperable will have failed if the attacker's goal was to cause detonation but will have succeeded if the goal was to prevent detonation. For devices used to control access by consumers to proprietary content, an attack is often considered successful if it extracts secrets being stored—even if the device is destroyed by the attack—because the stolen secrets then can be used to provide unlimited access by using some other device.

12.3 *Physical Unclonable Functions

Tamper-resistant packaging would not be needed to protect a circuit that unpredictably altered its state and/or operation in response to any physical accesses by attackers. Providing such functionality for circuitry that stores and/or computes functions of secret values is driving research into the development of *physical unclonable functions* (PUFs). They are not yet ready for general use—and some experts believe that simpler alternatives will always be a more sensible choice. Nevertheless PUFs are an intriguing idea to contemplate.

A PUF is a circuit instance C for implementing a function $\mathcal{F}_C(\cdot)$ that depends on some fixed, unmeasurable, and unclonable features of its realization on a specific chip and that satisfies the following properties.

- Evaluation of $\mathcal{F}_C(\cdot)$ is *repeatable*—the same value is produced every time $\mathcal{F}_C(x)$ is evaluated with a given input x from its domain.
- The value produced by evaluating $\mathcal{F}_C(x)$ cannot be predicted from invasive or non-invasive measurements of the chip that contains C.
- The value produced by evaluating $\mathcal{F}_C(x)$ changes unpredictably if the chip that contains C is modified or probes are attached.

Thus, $\mathcal{F}_C(\cdot)$ is individual, inherent, and unclonable.

The domain of function $\mathcal{F}_C(\cdot)$ depends on the PUF design. Some designs implement a function that takes no inputs and has a fixed (but unpredictable)

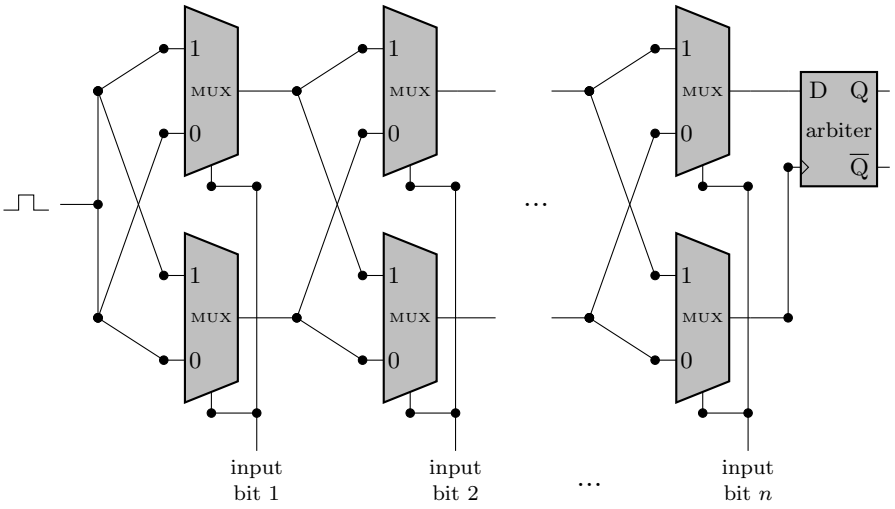

Figure 12.1: Arbiter-based PUF

output, causing the PUF to behave like a small read-only memory. Other PUF designs implement functions that do take inputs. With a *weak PUF*, the number of possible input values is linearly related to the number of components in the circuit used to realize the PUF; with a *strong PUF*, the number of possible input values is exponential in the number of circuit components. Because it is infeasible to collect the values $\mathcal{F}_C(x)$ for the exponential number of possible inputs x to a strong PUF, a strong PUF can be used to implement challenge-response protocols, where each challenge is used at most once.

Examples of PUF Designs. Most PUF designs are circuits whose output is determined by differences in signal propagation delays, where the differences in delays arise from uncontrollable aspects of chip fabrication. The output of a PUF thus depends, in part, on where the circuitry is located on some specific chip.

SRAM PUF. A 1-bit SRAM PUF implements a function having as its output the unpredictable, but (apparently) repeatable, value at power-up for a specific (uninitialized) SRAM volatile memory cell. With m of these, we obtain an SRAM PUF that produces an unpredictable but repeatable, instance-specific m-bit output. To obtain a PUF that maps an n-bit input to an m-bit output, it suffices to have (i) a set containing 2^n of these m-bit SRAM PUFs and (ii) a decoder circuit that uses the value of an n-bit input to select an associated PUF from the set.

Arbiter-based PUF. A 1-bit arbiter-based PUF outputs a 0 or 1 according to the faster of a selected pair of signal paths, where an n-bit input defines the

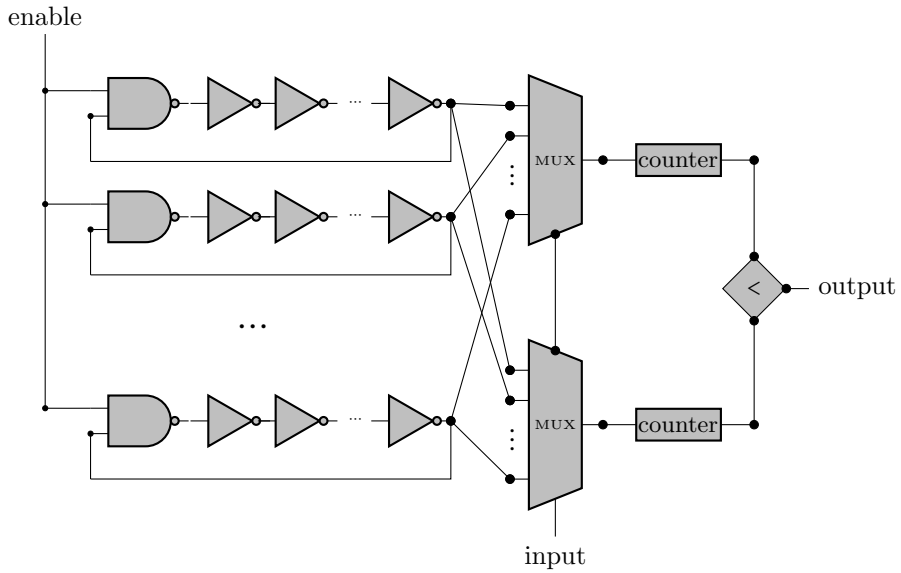

Figure 12.2: Ring-Oscillator PUF

segments used to form the two signal paths of the pair. Figure 12.1 gives a design. The output of that PUF is the output of the flip-flop labeled arbiter. That output is determined by the relative arrival times of the signal reaching that flip-flop's clock input (labeled >) versus its D input. These arrival times depend on the sequence of MUXes and interconnects that are traversed. That sequence is determined by the input bit to each MUX—input bit i determines for the MUXes in the i^{th} column whether the input port labeled 1 or the input port labeled 0 is the input that the MUX outputs. So an n-bit input defines one pair of the 2^n possible n-segment signal paths.

Because each of the segments has an unpredictable but fixed delay, each of the n-segment signal paths will have an unpredictable but fixed delay. An n-bit input selects a specific pair and, thus, always outputs the same unpredictable value. To build a PUF that produces an m-bit output, we use m of these 1-bit arbiter-based PUFs; the i^{th} 1-bit arbiter-based PUF produces bit i of the output.

Ring-Oscillator PUF. The frequency of a ring oscillator is determined by signal propagation delays in a feedback loop, so different instances of a ring-oscillator circuit are likely to have different frequencies. A 1-bit ring-oscillator PUF is built using a set of ring oscillators and some control circuitry. Figure 12.2 illustrates. Each ring oscillator involves a loop that comprises an NAND-gate followed by an even number of inverters. An n-bit input controls a pair of MUXes, causing a pair of ring oscillators to be selected and connected to counters. The frequency of each ring oscillator determines the speed that

counter is incremented, so a 0 or 1 will be output by the PUF according to which ring oscillator in the pair has higher frequency. To build a PUF that produces an m-bit output, it suffices to combine m of these 1-bit ring-oscillator PUFs.

PUF Repeatability and Unpredictability. Signal propagation delays in integrated circuits can be affected by operating temperature, power supply voltage, electrical noise, and other aspects of the environment. So a straightforward realization of the above PUF designs might, for a given input x, produce different values for $\mathcal{F}_C(x)$ depending on the current conditions. One way that a PUF circuit can compensate for environmental variation is to have its output depend on delay ratios (which tend to be more stable) rather than absolute delays. Repeatability also can be improved by incorporating error correcting codes into the output of a PUF. Finally, clients that submit an input and then check for a specific ouput can be designed to accept values that differ from the expected response by a small number of bits.

A second key requirement for a PUF realization is to have *unpredictability* of function $\mathcal{F}_C(\cdot)$:

PUF Unpredictability.

- An attacker who learns some set of input-output pairs $\langle x, \mathcal{F}_C(x)\rangle$ must not be able to predict the outputs for other inputs.
- Given a value y that has not yet been output by the PUF, an attacker must not be able to construct an input x satisfying $y = \mathcal{F}_C(x)$. □

The SRAM PUF above satisfies these properties provided the value of each 1-bit SRAM PUF does. This is because each 1-bit SRAM PUF is used in producing the output associated with only one input, so outputs that an attacker has already observed give no information about unseen outputs that the SRAM PUF will produce.

But unpredictability is harder to achieve when a small set of signal propagation delays are being combined in multiple (different) ways, as in the above arbiter-based PUF and ring-oscillator PUF. With an arbiter-based PUF, for example, submitting two inputs that only differ in their i^{th} bit enables an attacker to learn which alternative for the i^{th} segment is faster; $2n$ inputs thus suffice to reveal the faster signal path for all inputs. Under modest assumptions about possible differences in signal-path segment delays, output $\mathcal{F}_C(x)$ now can be used to predict output $\mathcal{F}_C(x')$ for other inputs x' that differ from x at a small number of bit positions.[2] However, unpredictability still can be obtained with this kind of PUF. One solution is to incorporate a cryptographic hash function at the input and/or output of the PUF. Another solution is to restrict the input domain D_C for $\mathcal{F}_C(\cdot)$ to be the subset of inputs x for which $\mathcal{F}_C(x)$ cannot be predicted from $\mathcal{F}_C(x')$ for other inputs x' in D_C; that PUF implementation also would incorporate circuitry that rejects inputs not in set D_C.

[2] A similar attack is possible for a ring-oscillator PUF. That attack would reduce the number of possibilities by leveraging the transitivity of < used in comparisons of oscillator frequency.

PUF Applications. An unpredictable bit string of any desired length can be formed by concatenating the outputs of one or more inputs to a single PUF or to different PUFs. Such a longer bit string could be used as

- the unique identifier for a chip instance,
- to generate a chip-specific symmetric key,
- to generate a chip-specific public/private key pair, or
- to generate a chip-specific seed for a random number generator.

However, some conditioning of the PUF outputs might be necessary to obtain distributions of values suitable for those applications. The necessary conditioning can be implemented by incorporating a hash function into the final stage of a PUF.

A PUF-generated chip-specific symmetric key K_P for a chip P enables secret values to be stored off-chip in a secure way. K_P would be generated whenever it is needed by P to encrypt secrets[3] for storage off-chip or to decrypt content being retrieved. By storing K_P in P's volatile memory only while K_P is needed for performing an encryption or decryption operation, K_P is vulnerable to theft for only short periods. Moreover, an attacker who removes P and probes the PUF used to generate K_P would (i) not be able to learn K_P and (ii) could well cause unpredictable changes to the value being generated for K_P.

Another use for PUF-generated cryptographic keys is to enable authentication of a chip P by its clients. One approach is to provision each client A with a separate symmetric key K_A generated by a weak PUF on P. K_A is then used in a standard shared key authentication protocol: chip P proves to A knowledge of K_A by performing encryption and/or decryption with that key. Notice, if the value of K_A was obtained by A directly from the chip's manufacturer, then this chip authentication protocol even defends against supply-chain attacks that alter P or substitute a different chip for P.

An alternative to using cryptographic functions for authenticating a chip is to leverage the unpredictability of a PUF C located on the chip. Each client A is provisioned with a disjoint set CR_A of challenge/response pairs $\langle c, \mathcal{F}_C(c)\rangle$. To authenticate the chip, a client A removes from CR_A some pair $\langle c, r\rangle$, submits challenge c to the chip, and deems the chip authenticated if the response *resp* that A receives from the chip satisfies $resp = r$. Replay attacks are prevented if the client never repeats a challenge. So the CR_A sets periodically must be refreshed with fresh challenge/response pairs generated using $\mathcal{F}_C(\cdot)$.

One option for refreshing the CR_A sets, which defends against chip substitution in the supply chain, is for the chip fabricator or system builder to have generated and saved a large set of such pairs produced with the PUF before the chip is put into operation. Another option, which does not defend against chip substitution in the supply chain, is to use the PUF *in situ* for producing these

[3] A sequence number should be included in the encrypted information to defend against a rollback attack that replaces the current version of off-chip storage by an older copy.

sets of pairs and then securely transfer the new set to a client. This second option requires the capability to transfer CR_A sets off-chip in manner that is confidential and can be authenticated.

12.4 Expectations about Memory and Storage

Programmers have expectations about the behavior of random-access main memory (RAM) and disks. Those expectations and attacks to falsify them are the subject of this section. One class of attacks exploits *data remanence*—evidence that reveals information about previously stored values. If stored values are not encrypted, then confidentiality can be compromised by an attacker with access to data remanence.[4] A second class of attacks involves writes to one address in memory that also alters information being stored at a different address, thereby compromising integrity.

Expectations about RAM and Disk. Interfaces to RAM and disks typically provide operations for reading and writing addressable, disjoint, fixed-size objects. Different technologies then lead to storage implementations that differ in cost, performance, as well as other attributes. Programmers, however, will expect `read` and `write` operations to satisfy certain axioms, independent of the technology.

A1: Execution of `read`(x) reveals the value currently stored by x—not a past value or the value stored by some other object.

A2: Execution of `write`(x, val) changes only the value stored by x. The value of no other object changes.

A3: Values stored in *volatile* memory are erased when power is removed; values stored in *stable* storage persist, even after power is removed.

To erase the value stored by x, these axioms imply that a programmer can (i) write a new value to x and depend on axiom A2, or (ii) if x is in volatile memory, then remove power and depend on axiom A3. Note that axioms A1 and A2 together imply that the value `read`(x) returns cannot be changed by executing `write`(y, val) where x and y identify different objects.

12.4.1 Attacks on RAM

Semiconductor RAM is typically structured as an array of cells, where each cell stores 1 bit. A DRAM (Dynamic Random Access Memory) cell represents that bit by the amount of electrical charge a capacitor stores; an SRAM (Static Random Access Memory) cell represents the bit by carrying current in one of two electrical feedback loops. DRAM cells require less chip area, have higher power consumption, tend to be slower, but are cheaper per bit than SRAM cells.

[4] For this reason, newer processors encrypt values written to memory or to other storage.

DRAM is typically used for main memory; SRAM is used for CPU registers and cache.

RAM Imprinting. Semiconductor RAM stores information by harnessing certain physical phenomena. Other physical phenomena exist, however, that can be exploited by attackers to cause *RAM imprinting*, whereby information being stored in semiconductor RAM persists long after it should have decayed:

- Low temperatures impede the flow of charge in semiconductors. So by cooling chips that are implementing a volatile memory, an attacker can imprint the contents of that memory for inspection after power has been removed.
- X-ray band irradiation of CMOS RAM transforms the semiconductor in ways that reflect the distribution of charge, are permanent, and are measurable. A memory implemented with these chips may no longer function as expected, but the chips will have recorded—for later inspection—a snapshot of memory.

The obvious defense against attacks that manipulate a memory's physical environment is to thwart physical access by attackers. A tamperproof enclosure is one such defense.[5] Alternatively, a computer could be situated someplace that is inaccessible to attackers. Also, we frustrate an attacker's efforts to remove and read imprinted RAM chips if the chips are permanently glued to the motherboard, so removal destroys them. An operating system can help defend against RAM imprinting caused by low temperatures if the system startup and shutdown code always overwrites all regions of memory that could have been storing secrets. This overwriting forces an attacker (i) to avoid a normal shutdown and (ii) to boot custom code for accessing the RAM chips.

Cold Boot Attacks. Cryptographic keys are often stored in a computer's main memory. Programmers expect this memory to be volatile and, therefore, they assume cryptographic keys stored there will no longer be available after the computer has been powered down. This is not an unreasonable assumption. Main memory invariably is implemented by DRAM, and at standard operating temperatures (25℃–50℃), a powered-off DRAM chip will retain its values for at most a few seconds before those values decay into random noise. However, when cooled[6] to −50℃, values in a DRAM chip will remain uncorrupted for a minute or more, and when that DRAM chip is submerged in liquid nitrogen (−196℃), data corruption is extremely low, even after an hour. These observations are the basis for so-called *cold boot* attacks, which were developed to learn disk encryption keys from a computer's DRAM main memory after a shutdown or hibernation operation that did not overwrite that DRAM main memory.

[5] X-ray radiation shielding can be unwieldy. Fortunately, X-ray band irradiation attacks can be launched only by well-resourced threats, so X-ray shielding is rarely needed.

[6] Cooling to −50℃ can be achieved through evaporation by spraying a DRAM chip with one of the commercially available compressed-air duster products sold to clean equipment.

Cold Boot Attacks. Secrets stored in DRAM chips on a running computer can be recovered if an attacker cools those chips and then

- restarts the computer, booting a kernel that requires only a small memory footprint and that gives the attacker access to the rest of memory, or
- removes those DRAM chips and inserts them on a computer that gives the attacker access to this memory. □

Other RAM Remanence. If a value V is stored for an extended period at some location in a semiconductor RAM, then electromigration, hot carriers, ionic contamination, and other physical phenomena can change the cell in ways correlated with V. Moreover, these changes remain detectable—even after that location has been overwritten and after the computer is powered-off. So data remanence has been created. To measure some of the changes requires specialized equipment and requires removing the affected RAM chip from the motherboard; only some threats will have those capabilities. Other changes to RAM, though, are directly visible to a running program. For example, a program might be able to deduce what value a given location had stored for a long time simply by reading that location's uninitialized value at power-on.

RAM remanence caused by storing a value for an extended period is more than a theoretical curiosity. It is potentially a significant vulnerability for secure coprocessors, which have separate key memories and are likely to store long-term cryptographic keys and other secrets in the same locations for extended periods. It also is potentially a vulnerability for ordinary operating systems, which typically occupy the same (low) memory region on a given computer and, therefore, use the same fixed memory locations for storing their long-term cryptographic keys and other secrets.

For those of us who do not control the design and fabrication of a system's semiconductor RAM chips, the obvious way to avoid remanence arising from long periods of storing the same value at a given location is to arrange that no memory location holds one of these values for very long. A few minutes is a safe upper bound for storing a value. Two implementations of this defense are:

- Every few minutes, copy the value to a different memory location and then write random values into the memory locations from which the value was just copied.
- Every few minutes, complement the memory locations that are storing the value and update a corresponding *representation indication* that records whether the value or its complement is currently being stored. Modify read operations to check the representation indication and, when appropriate, return a complement of the value retrieved from memory.

Row-Hammer Attacks. Increases to the density of DRAM lead to an increase in *disturbance errors*, which are changes to the value being stored in one cell that are caused by accesses to another cell. Disturbance errors violate the

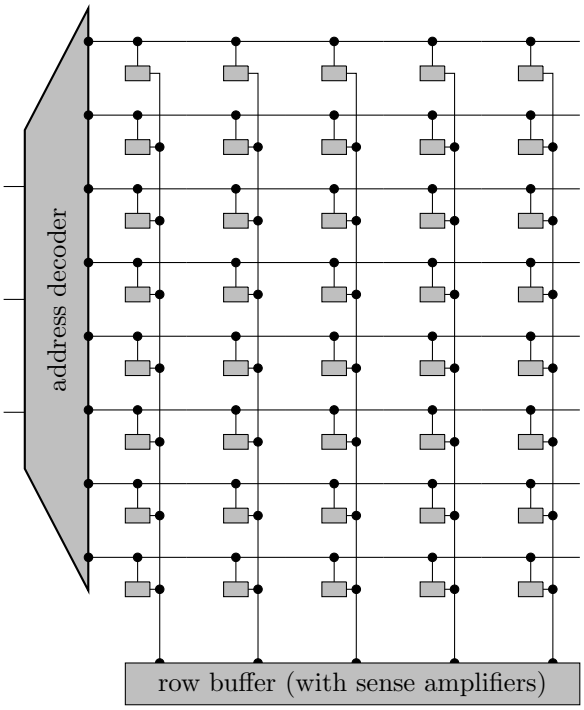

Figure 12.3: DRAM Organization

axiom (A2, page 387) that writing to a given memory location is the only way to change the value that location is storing. Disturbance errors become vulnerabilities if attackers can use them to change values being stored by targeted cells. The vulnerability has been present in many of the DRAM chips produced since 2010. With these DRAM chips, cells are internally organized as an array, and making a series of accesses to cells in one row can alter the values being stored by cells in adjacent rows. Such a series of accesses is known as a *row-hammer* attack.

DRAM Internals. To undestand how row-hammer attacks work requires an understanding of DRAM circuitry. As depicted in Figure 12.3, a DRAM consists of an array of cells (depicted by small rectangles) connected to a *row-buffer* that, for each column in the array, contains a cell and a sense amplifier. DRAM read and write operations access cells in the row-buffer; refresh for a row is done by reloading the entire row from the row-buffer. So the row-buffer at any time will store the same values as the cells in some *selected* row.

All DRAM cells in a given row of the array are connected to a *wordline* for that row, and all DRAM cells in a given column of the array are connected to a *bitline* for that column. Wordlines are driven by an address decoder; it selects a single row for transfer to/from the row-buffer by elevating the voltage on the

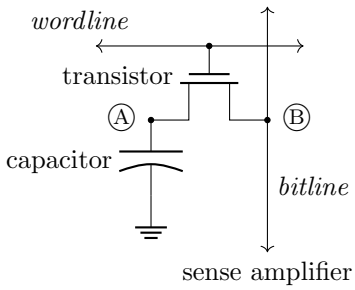

Figure 12.4: Circuit for DRAM Cell

corresponding wordline and having low voltage on all other wordlines. Each bitline is terminated by a separate *sense amplifier* in the row-buffer.

Figure 12.4 shows a circuit that implements a single DRAM cell. The wordline voltage causes the transistor to control current flow between Ⓐ and Ⓑ and, thus, controls current flow between the capacitor and the bitline. During periods when the wordline voltage is low, current flow between Ⓐ and Ⓑ is blocked, so the capacitor is electrically isolated and retains its charge (except for leakage). An elevated wordline voltage allows current flow between Ⓐ and Ⓑ, enabling the capacitor's charge to be measured and/or changed by the sense amplifier that terminates the bitline.

In order to service a read, write, or refresh operation for the contents of some row r (say), the voltage is elevated on the wordline for r and then the sense amplifier terminating each bitline b performs a sequence of 2 steps.

(i) The sense amplifier measures the voltage on bitline b and compares that value to a threshold. The outcome of that comparison indicates the value of the bit being stored by the capacitor $C_{r,b}$ of the row r cell that is connected to bitline b. A DRAM *true-cell* is storing 1 if the measured voltage was found to be above a threshold and it is storing 0 if the measured voltage was below; in a DRAM *anti-cell*, this representation is inverted.[7]

(ii) After making that voltage measurement, the sense amplifier sets the voltage on bitline b to a value that will cause the charge in $C_{r,b}$ to be set according to whether the corresponding cell in the row-buffer has been storing a 0 or 1.

- For a read operation or a refresh operation, the charge in $C_{r,b}$ is restored, thereby compensating for past charge leakage and for charge loss from performing the measurement in step (i).
- For a write operation, the charge in $C_{r,b}$ is set according to the value of the new bit to be stored.

[7] A DRAM chip might include a mixture of true-cells and anti-cells.

```
L: load  r1,x    access row containing x
   load  r2,y    access row containing y
   flush x       evict x from cache
   flush y       evict y from cache
   mfence        flush pipelines
   jmp L         iterate
```

Figure 12.5: Row-hammer Attack

Physics of Row-hammer Attacks. The laws of physics imply that a change to the current flow on a wordline induces a current flow in all physically parallel wordlines. The induced current is higher in wordlines that are physically closer, so a change to the current flow in the wordline for a row r induces the largest flow current in the wordlines for rows $r+1$ and $r-1$. That induced current can cause transistors in row $r+1$ and $r-1$ cells (but possibly cells of other rows too) to allow a short period of modest current flow between the capacitors they control and bitlines. Those current flows leak charge from the capacitors.

If enough charge gets leaked from a capacitor $C_{r,b}$ before the next refresh operation is performed for row r, then $C_{r,b}$ would transition from storing an above-threshold charge to storing a below-threshold charge—a disturbance error. For a true-cell, a 1 changes into a 0; for an anti-cell, a 0 changes into a 1. DRAM designers have an incentive to have a large interval between refresh operations, because performing a refresh, read, or write operation requires exclusive use of the row-buffer and, therefore, read and write cannot be overlapped with refresh. So the interval between refresh operations for a given row on a modern DRAM typically is chosen to be just small enough to remediate ordinary charge leakage. That interval does not forestall disturbance errors caused by charge leaks resulting from repeated access to adjacent rows.

> **Executing a Row-hammer Attack.** Disturbance errors that corrupt the values stored by cells in *victim* rows $r-1$ and $r+1$ of a DRAM can be caused when a program repeatedly performs memory accesses that elevate and drop the voltage on the wordline for *aggressor* row r. □

Code that performs a row-hammer attack is given in Figure 12.5. Variable `x` should be stored in a DRAM row that is adjacent to the victim row; variable `y` should be stored in any other row.[8] The `flush` operations evict `x` and `y` from the cache to ensure that the `load` instructions in each loop iteration elevate the intended wordline voltages by fetching the values from the DRAM. Execution of `mfence` at the end of each iteration drains the pipeline, thereby preventing pipeline logic from suppressing[9] the `load` operations for `r1` and `r2`. By accessing

[8] If `x` and `y` are each stored in different rows that are adjacent to the victim row, then the attack is known as *double-sided hammering*. With *many-sided hammering*, there are more than two aggressor rows; it is effective when the geometry of a specific DRAM realization causes additional inductive couplings between wordlines.

[9] Pipeline logic often will skip performing an update to memory or registers if that update

`x` in alternation with `y`, the loop repeatedly applies the voltage for selecting the wordline for the row storing `x`, even with a DRAM implementation where consecutive read accesses to the same row are serviced by the row-buffer without repeatedly selecting and copying that row to the row-buffer.

Defending Against Row-hammer Attacks. Incorporating an error-correcting code (ECC) into each DRAM row might seem the obvious defense against row-hammer attacks. The number of ECC bits required per row, however, is proportional to the maximum number of cells in a row that could need to be corrected. Since a row-hammer attack can corrupt many cells in a victim row, and any DRAM row could be a victim row, we would require many ECC bits. That storage overhead makes using ECC an impractical defense.

Frequent refresh operations is the other obvious defense. But additional refresh operations consume power and reduce DRAM bandwidth by leaving less time for read and write requests. Moreover, to defend against a row-hammer attack, frequent refresh for all rows is not necessary. Additional refresh is needed only for potential victim rows—any row r adjacent to one or more rows that, in aggregate, were frequently accessed since r was last refreshed.

Various schemes have been suggested for deployment in a DRAM chip or memory controller in order to identify potential victim rows and initiate refresh just for those. Two of the more influential schemes are TRR (Target Row Refresh) and PARA (Probabilistic Adjacent Row Activation). TRR requires additional state; PARA does not require additional state but only gives a probabilistic guarantee.[10]

TRR: A row r' is deemed a potential victim row and refreshed if the number of accesses to any adjacent row has reached a chip-specific threshold MAC within a chip-specific period of length t_{MAW}. Variations include:

- Use the aggregate number of accesses to a set of adjacent rows as the basis for deciding whether a row should be refreshed.
- Use sampling on the stream of DRAM accesses to approximate the number of accesses to each row.
- By using a fixed-depth stack, maintain access counts for only some fixed, small number of rows that have received the largest number of accesses within the last t_{MAW} period.

PARA: Whenever a row is activated then, with probability p, refresh one of its two adjacent rows. Thus, the probability that an access to a neighboring row does not cause r to be refreshed is $1-p/2$, and the probability that row r is not refreshed during a row-hammer attack involving N total accesses

will be overwritten before being read. In the loop of Figure 12.5, registers `r1` and `r2` are not read before being loaded again.

[10] Both schemes have been implemented in some DRAM chips and in memory controllers. TRR support, going by the name refresh management (RFM), appears in recent generations of JDEC (Joint Electron Device Engineering Council) DRAM memory standards.

to rows neighboring r is $(1 - p/2)^N$. The table below uses this formula to give the probability that a row-hammer attack involving N accesses would succeed when refresh is expected to be performed only once for every 1000 accesses (p = .001). Probability of success for a row-hammer attack is given both for a single refresh period (64 ms) and for a year.

Duration	N = 50K	N = 100K	N = 200K
64 ms	1.4×10^{-11}	1.9×10^{-22}	3.6×10^{-44}
1 year	6.8×10^{-3}	9.4×10^{-14}	1.8×10^{-35}

12.4.2 Attacks on Magnetic Storage

Hysteresis is what makes magnetization well suited for implementing non-volatile storage. We create a *magnetic storage medium* by applying a thin film of ferromagnetic material to a disk platter or a tape. The ferromagnetic film enables physically disjoint regions—called *domains*—on the surface of that storage medium to assume either of two magnetic *polarities*. A bit string is then represented by the sequence of magnetic polarity changes encountered by a *read/write head* while traveling above the series of domains that constitute a *track* on the storage medium. On magnetic tapes, tracks run parallel to the length; on magnetic disks, tracks are concentric circles.

A storage device is realized by using (i) some magnetic storage medium, (ii) read/write heads to sense and to set the magnetic polarity for domains in a track segment[11] passing underneath, (iii) mechanisms to select which track segments pass underneath the read/write heads, and (iv) electronics for translating a sequence of magnetic polarity changes to/from the string of bits being represented.[12] On a tape drive, one read/write head typically will cover all of the tracks; a motor spools the tape forward or backward, so some selected track segment passes underneath this read/write head. A disk drive typically has a read/write head for each platter surface; all of the platters are rotating in tandem, and there is a mechanism for positioning read/write heads over a selected track on all platters.

A write operation is performed by activating the read/write head to set the magnetic polarities for the sequence of domains that are passing underneath. Subsequent read operations recover that bit string by measuring the average magnetization in each domain as it passes underneath the read/write head; a sequence of polarity transitions is constructed from those averages. Because the average magnetization that a read measures for a domain will indicate the polarity of the last magnetization from a write to that domain, axioms A1 and A2 (page 387) that we expect to hold for a storage device should indeed hold.

The averaging performed in a read operation when a domain passes underneath provides a way to compensate for two effects:

[11] Depending on the device, a track segment might be known as a *record*, a *sector*, or a *block*.

[12] Modern magnetic storage devices use run-length limited translation to ensure that frequent changes in the direction of magnetization occur for all bit strings—even bit strings containing long sequences of the same bit.

- *Positioning Errors.* Positioning a read/write head over a moving track is a mechanical operation. So a slightly different region of the storage medium passes underneath the read/write head each time a given domain is read or written.

- *Partial Magnetizations.* For performing write operations, an electromagnet in the read/write head is used to set the magnetic polarity of domains passing underneath. An electromagnet that is too strong would set the magnetic polarity for a large surrounding region (perhaps including other domains). So a weaker electromagnet is used. But due to natural variations in magnetic susceptibility of materials, using the weaker electromagnet means that portions of a domain passing underneath the read/write head might not get magnetized with a new polarity.

To overcome these effects, and have read and write work as expected, (i) the threshold used in deciding a region's magnetic polarity is lowered, and (ii) read/write head positioning is engineered to be accurate enough to ensure significant (if not complete) overlap in the area that passes underneath each time a given domain is being accessed. These implementation tolerances, however, can cause remanence.

Remanence in Magnetic Storage. When attackers can move a magnetic storage medium into a laboratory, analysis with magnetic force microscopy (MFM) becomes possible. MFM creates an image of the medium's surface. Each point in that image depicts the magnetization (strength and polarity) for a small area of that medium's surface—an area far smaller than a domain. Not surprisingly, such an image can be used to learn the values being stored. We average the polarities of points in the image that are located within each domain forming a track, detect transitions from those averages, and use that sequence of transitions to reconstruct the values being stored on the medium.

An image produced by MFM also will contain two kinds of points that are forms of remanence.

- *Magnetized points located outside of any domain.* These points are most likely to be near a track edge or at the boundary between domains within a track. They are caused either by positioning errors or by regions of the storage medium that have high magnetic susceptibility and are near a domain.

- *Disparate points within a domain.* Points arise that are within a domain and have a different magnetic polarity than the average for that domain. This occurs whenever the last write to that domain did not change some small region's magnetic polarity, because that region has low magnetic susceptibility.

What can this remanence reveal? A region containing many disparate points probably is indicating the value being stored prior to the last write.

Magnetic Disk Sanitization. Servers, desktop computers, and personal devices will be decommissioned and replaced from time to time, then repurposed, donated, or discarded. Part of the decommissioning should be to erase information that the computer's disk was storing, since erasure protects the confidentiality of information without requiring assumptions about access controls that computer will enforce in the future.

The procedure to erase the contents of a magnetic disk is called *disk sanitization*. Three approaches[13] are: overwriting, degaussing, and shredding. Which approach is the most appropriate for a given setting will depend on the capabilities of the threat and on whether the disk must still be usable after sanitization has been performed.

Overwriting. With a correctly operating magnetic disk, writing a new value prevents a later read operation from recovering the overwritten value. But a write does not necessarily prevent MFM from recovering the overwritten value and perhaps earlier values, depending on the encoding that was used to represent bit strings.

- With the data encodings used for disks in mid 1990's and earlier, overwriting multiple times with different and unpredictable values is likely to erase a value and any associated remanence.

- With the encodings that achieve high density in modern magnetic disks, recovering old values from remanence is virtually impossible, so overwriting a value once suffices to prevent recovering that value by using MFM.

However, if a given domain has the same polarity for long periods of time and/or the magnetic media is stored at elevated temperatures, then a bias for that polarity is created because the threshold for setting the magnetization to that polarity will be permanently lowered. Later writes will not alter this bias. So, in a laboratory, the original value stored using that domain would thereafter be detectable.

Although remanence is not a problem with modern magnetic disks, they do have features that complicate the use of overwriting for performing disk sanitization.

- Some disk controllers buffer values rather than immediately updating the magnetic storage medium. With such a controller, overwriting (to erase a value) is not guaranteed to perform write operations on the magnetic storage medium, and a sequence of overwrites made to the controller might not translate into the same sequence of writes to the magnetic storage medium.

[13] These approaches also work for sanitizing magnetic tape or other magnetic storage media. They do not work for SSD's (solid-state drives), because the command to delete or update the contents of a block on an SSD leaves the contents of that block intact and, instead, re-maps that block's address. Specialized sanitization commands are therefore typically provided by SSD hardware.

- Some disks forestall data loss by detecting when a sector or track is becoming marginal and, in response, copy its contents to an alternate region of the magnetic storage medium. Thereafter, accesses to the marginal region are redirected to the new region by the disk controller. So the contents of the sector or track at the time it was deemed marginal cannot be erased by overwriting.

Degaussing. A *degauser* employs an electromagnet to create a strong magnetic field. When the degauser is brought into close proximity to a disk, this strong magnetic field changes the magnetic polarities for all regions of the magnetic storage media. Values represented by magnetic polarities in domains and in remanence are thus destroyed. Use of a degauser, however, also can leave a disk inoperable by (i) corrupting formatting information that had been magnetically encoded on the storage media, and (ii) altering the magnets used in the motors that rotate the disk and that position the read/write heads.

Shredding. The magnetic storage medium is cut into small pieces that cannot be reassembled. To be completely effective, none of these pieces should be large enough for MFM to recover useful information—with modern high-density drives, the pieces must be made quite small. Needless to say, shredding leaves a disk unusable, though some drives do have replaceable magnetic storage media and, therefore, some of the mechanism might be reused with new media.

12.4.3 Remanence Software Generates

Developers favor interfaces that hide implementation details. Security engineers, however, must have knowledge of those implementation details when seeking to avoid remanence. A trivial cause of remanence is operations that we expect would provide sanitization but don't. File systems offer good examples. Invoking a file system's `delete` or `write` operations would seem an obvious way to obliterate a file's contents.

- In some file systems, invoking `delete` on a file in some directory has no effect on the file contents being stored on disk—it merely removes the file's name from that directory after copying that file name to another `trash` directory. So `delete` does not implement sanitization, because it does not prevent subsequent access to the file's contents through the `trash` directory.

- In other file systems, a log records the old and new values for each `write`, thereby allowing earlier versions of a file to be recreated. So overwriting a file does not implement sanitization, because it does not prevent subsequent disclosure of file contents that had been overwritten.

You might expect that every file system interface would provide an operation to perform sanitization, even if `delete` or `write` do not have that effect. The

prevailing view, however, is to favor user convenience over security. So contemporary file systems make it easy for a user to reverse a `delete` or `write` operation and make it difficult to obliterate information irreversibly. Therefore, sanitization operations are not provided.

A second way that a program might generate remanence is by (i) being assigned access to some stateful resource[14] R, (ii) writing and reading R, and finally (iii) relinquishing access to R. If R is not sanitized before being assigned to some other program P (say), then R will contain remanence that P can read. Note, sanitization of R not only requires overwriting its state; state also would have to be flushed from caches and buffer pools (which might be hidden in lower levels). The operating system would seem a natural place to perform this sanitization, since an operating system allocates resources and has access to memory, buffer pools, and caches. Operating system designers, however, cite performance degradation as the reason for not doing sanitization by default—overwriting state consumes processing time, and flushing caches and buffer pools cause degraded performance when execution restarts. Remanence should thus be expected in a stateful resource R unless each individual program to which R is allocated sanitizes R before relinquishing that access.

A third cause of remanence can arise when an implementation maintains copies of state in order to satisfy performance goals.

- A large virtual memory fits into a small real memory because only some pages are present in real memory at any time. But all pages reside in a paging file on disk, so state is duplicated.

- A file system is able to deliver faster access by buffering copies of a file's disk blocks in main memory. So state is duplicated.

Clients have no direct way to delete or overwrite these state copies. Moreover, because the state copies are invisible to clients, an implementation would not necessarily be sanitizing the state copies when an associated abstraction is sanitized. So the state copies are a form of remanence. Whether that remanence can be accessed by an attacker will depend on whether access to those state copies is being controlled. Often a state copy will be protected by different access control mechanism than used to protect the original, as illustrated in the virtual memory and file system examples above.

Finally, remanence—whether created by software or exhibited by RAM or magnetic storage—is harmless if it derives from encrypted state. Moreover, encrypted state can easily be sanitized by deleting or overwriting the key. The costs for encrypting and decrypting state, however, can be significant if done by software. To lower those costs, newer I/O devices and CPUs provide hardware support. A disk controller might, for example, include hardware to generate a symmetric key, thereafter using that key to encrypt blocks as they are written

[14] The resource might be a register, region of real or virtual memory, disk block, or a software abstraction that directly or indirectly uses these hardware storage mechanisms to maintain state.

to the disk and to decrypt blocks as they are read. Modern CPU designs increasingly will generate, store, and use per-principal (sometimes called *enclave*) keys to encrypt, decrypt, and/or digitally sign information to/from the CPU chip, whether that information is en route to a cache, to a page frame in the main memory, to a page file on a disk, or to a network adapter.[15]

Notes and Reading

Tamperproof Processors. Programmers assume that computer hardware will function as expected. But even mainframe computers in locked machine rooms are vulnerable to tampering during regularly-scheduled maintenance periods. Molho [17] discusses how a maintenance technician, by changing just a few wires in an IBM 360/50 mainframe, could disable that processor's circuits for restricting execution of privileged instructions and for protecting parts of memory.

With the advent of small and low cost microprocessors, computers no longer had to be housed in machine rooms. Cash machines, smartcards, and personal computers were now feasible. Because legitimate users required physical access in order to operate this equipment, attackers had physical access, too. Price [20] introduced many of the now standard approaches for building tamperproof packagings: potting to obstruct access to components and connections, fine wires in that potting to detect penetration attempts, and different positioning for those wires in each chip instance so that attackers who deconstruct one instance of a chip do not learn information that helps in compromising another instance.

IBM researchers were among the first to build and write about a system that used these methods. They argued that sales of personal computers would benefit from a rich market for application software, but developers would be reluctant to invest in building such software absent barriers to prevent illicit copying. Software alone could not solve that piracy problem, since its execution could be subverted by tampering with the hardware. Tamperproof hardware was required. So a group at the Yorktown research laboratory developed the tamperproof μABYSS coprocessor [24] to support their ABYSS (A Basic Yorktown Security System) architecture [26] for preventing illicit distribution of personal computer software. Although μABYSS never became a product, it was a precursor to a series of tamperproof crypto-coprocessor products from IBM that enjoyed considerable success in the market.

Tamperproof packaging interferes with only certain ways of getting physical access in order to monitor or change the operation of a circuit. Physical access to a circuit is less concerning, however, if a circuit's realization is itself tamperproof. Physically unclonable functions (PUFs) are a class of intrinsically tamperproof circuit realizations. Gassend et al. [6] introduced the term PUF and proposed incorporating a PUF into an integrated circuit. Those authors,

[15] Arithmetic calculations and determining transfers of control require plaintext. So the CPU internally uses plaintext, which forces its registers and other on-chip memory to store plaintext.

originally seeking a way to authenticate silicon chips, also discuss in [5] how to use a PUF in solving other security problems. For more details about PUFS, see the primer by Bauer and Hamlet [3] or the tutorial by Herder et al. [11]. Figures 12.1 and 12.2 are based on Suh and Devadas [23].

The idea of capturing unique physical properties of an inanimate object in a digital signature had been brought to the cryptography community a decade earlier by Simmons [21], who described two schemes developed in the 1980's at Sandia by a colleague Don Bauder. One scheme facilitated detection of counterfeit paper money [2]; the other—a *reflective particle tag* (RPT)—enabled inventorying nuclear weapons in support of the Intermediate-range Nuclear Forces (INF) treaty. Oliver and Fritz Kömmerling documented the next step with a December 2000 patent filing [14] that showed how properties in the packaging or substate for an integrated circuit could be used for a tamperproof approach to generating a cryptographic key—in effect, describing a special-purpose PUF.

Attacks on Memory. The effects of cooling on RAM remanence was reported by Link and May in 1979 [15], and those effects were reconfirmed for circa 1988 commercially available DRAM in Wyns et al. [28, 27] and for circa 2002 commercially available SRAM in Skorobogatov [22]. Weingart [24] in 1987 suggests that by freezing the RAM chips implementing a volatile memory, an attacker could recover information stored there before the computer was powered down. Actual attacks to recover encryption keys from DRAM after a computer had been powered down were demonstrated by a group at Princeton [10]; the term "cold boot attack" was introduced in that paper.

Cooling is not the only physical effect that attackers can use to prolong remanence for RAM chips. Weingart [25] notes that X-ray band irradiation of a RAM chip will imprint the chip's contents for later inspection. He also suggests that short duration high-voltage spikes might have the same effect. See Gutmann [9] for explanations of how various physical phenomena cause data remanence in semiconductor memory devices.

Kim et al. [12], describes why row-hammer attacks ought to be possible, gave code (the basis for Figure 12.5) to cause these targeted disturbance errors, and analyzed possible defenses. Probabilistic Adjacent Row Activation (PARA) was introduced in that paper as a lower-cost alternative to row-hammer defenses that use row-access counts (or approximations) for instigating additional refresh operations of likely victims. DRAM disturbance errors, however, had been observed starting with the first commercially available DRAM, the Intel 1103 introduced in October 1970. By 1999, Van de Goor and de Neef [19] were considering a "hammer test" in experiments to assess ways to evaluate DRAM chip reliability; the hammer test would write each cell 1000 times and then verify that nearby cells were not disturbed.[16] The goal of avoiding disturbance errors for all workloads—especially given expectations of the higher-density chips

[16]The term "hammer test" had further evolved by August 2013, where we see a slide deck [16] for a MemCon talk that is using the term "row hammer" for this source of DRAM disturbance errors.

to come—resulted in Intel engineers developing schemes that used row-access counts to instigate additional row refreshes. These schemes are described in patent applications [1, 7] that were filed in 2012 (becoming public only some months after Kim et al. [12] had been submitted for publication). The Intel work doubtless is the basis for Target Row Refresh (TRR) found in the various DRAM standards from JDEC (Joint Electron Device Engineering Council).

Since virtually all systems included DRAM, the revelations in Kim et al. [12] prompted the security community to engage. A 2020 retrospective by Mutlu and Kim [18] surveys that work, including how attacks to flip a bit can be leveraged for taking control of a system, how software might be modified to resist row-hammer attacks, and various proposals for hardware defenses. Various TRR versions are being implemented today by DRAM manufacturers—probably because TRR is part of JDEC DRAM standards and involves no changes to other hardware or to software. Frigo et al. [18] measure the effectiveness of these TRR implementations in defending against row-hammer attacks, finding that the defenses being deployed in 2020 were not completely effective.

Attacks on Magnetic Storage. Guttmann [8] discusses the relevant physics foundations and engineering challenges for implementing magnetic storage (circa 1996), how remanance is being produced, recovery of values using magnetic force microscopy and other laboratory instrumentation, and protocols for erasing values. Although some of that material does not apply to newer storage technologies, that paper remains an important resource, and Guttmann has been providing updates on his web site. Generally accepted guidance for sanitization of magnetic media is given by NIST [13]. This guidance is not followed often enough, though, as a study by Garfinkel and Shelat [4] showed. In that study, the authors collected a large number of decommissioned computers and, because disk sanitization had been performed poorly or not at all, were able to recover confidential personal from the disks.

Bibliography

[1] Kuljit S. Bains, John B. Halbert, Christopher P. Mozak, Theordore Z. Schoenborn, and Zvika Greenfield. Row hammer refresh command. US Patent Application Publication US 2014/0006703 A1. Filed June 30 2012, publication date January 2, 2014.

[2] D. W. Bauder. An anti-counterfeiting concept for currency systems. Technical Report PTK–11990, Sandia National Labs, Albuquerque, NM, 1983.

[3] Todd Bauer and Jason Hamlet. Physical unclonable functions: A primer. *IEEE Security and Privacy*, 12(6):97–1015, November/December 2014.

[4] Simson L. Garfinkel and Abhi Shelat. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security and Privacy*, 1(1):17–27, January 2003.

[5] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Controlled physical random functions. In *Proceedings of* 18^{th} *Annual Computer Security Applications Conference*, CSAC '02, pages 149–160. IEEE Computer Society Press, December 2002.

[6] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 148–160. Association for Computing Machinery, November 2002.

[7] Zvika Greenfield, John B. Halbert, and Kuljit S. Bains. Method, apparatus and system for determining a count of accesses to a row of memory. US Patent Application Publication US 2014/0085995 A1. Filed September 25 2012, publication date March 27, 2014.

[8] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the 6th Conference on USENIX Security Symposium*, USA, July 1996. USENIX Association.

[9] Peter Gutmann. Data remanence in semiconductor devices. In *10th USENIX Security Symposium (USENIX Security 01)*, Washington, D.C., August 2001. USENIX Association.

[10] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *17th USENIX Security Symposium (USENIX Security 08)*, San Jose, CA, July 2008. USENIX Association.

[11] Charles Herder, Meng-Day (Mandel) Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[12] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 361–372. IEEE Press, 2014.

[13] Richard Kissel, Andrew Regenscheid, Matthew Scholl, and Kevin Stine. Guidelines for media sanitization. Technical Report NIST Special Publication 800–88, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, Gaithersburg, Maryland, December 2004.

[14] Oliver Kömmerling and Fritz Kömmerling. Anti tamper encapsulation for an integrated circuit. U.S. Patent 7,005,733 B2. Filed December 26, 2000, issued February 28, 2006.

[15] W. Link and H. May. Eigenschaften von MOS-ein-transistorspeicherzellen bei tiefen temperaturen. *Archiv für Etektronik und Übertragungstechnik*, 33, June 1979.

[16] Mike Micheletti. Tuning DDR4 for power and performance. Slides from presentation at MemCon, August 2013. http://cdn.teledynelecroy.com/files/whitepapers/tuningddr4_for_power_performance.pdf.

[17] Lee M. Molho. Hardware aspects of secure computing. In Harry L. Cooke, editor, *Proceedings of the 1970 Spring Joint Computer Conference*, volume 36 of *AFIPS Conference Proceedings*, pages 135–141. AFIPS Press, May 1970.

[18] Onur Mutlu and Jeremie S. Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, August 2020.

[19] J. de Neef and A van de Goor. Industrial evaluation of DRAM tests. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 623–630. IEEE Computer Society, March 1999.

[20] W. L. Price. Physical security of transaction devices. Technical Memo DITC 4/86, National Physical Laboratory, January 1986.

[21] G. J. Simmons. Identification of data, devices, documents and individuals. In *Proceedings 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*, pages 197–218, 1991.

[22] S. Skorobogatov. Low temperature data remanence in static RAM. Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory, June 2002. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf.

[23] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, June 2007. Association for Computing Machinery.

[24] S. H. Weingart. Physical security for the μABYSS system. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 52–52. IEEE Computer Society Press, April 1987.

[25] Steve H. Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems—CHES 2000*, pages 302–317, Heidelberg, 2000. Springer.

[26] S. R. White and Liam Comerford. ABYSS: A trusted architecture for software protection. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 38–38. IEEE Computer Society Press, April 1987.

[27] P. Wyns and R. L. Anderson amd W. F. DesJardins. Temperature dependence of required refresh time in dynamic random access memories. In *Proceedings Symposium Low Temperature Electronics and High Temperature Superconductors*, volume 88–9, pages 234–239. The Electrochemical Society, 1988.

[28] P. Wyns and R. L. Anderson. Low-temperature operation of silicon dynamic random-access memories. *IEEE Transactions on Electron Devices*, 36(8):1423–1428, 1989.