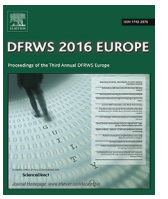
Digital Investigation 16 (2016) S65–S74

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin



DFRWS 2016 Europe — Proceedings of the Third Annual DFRWS Europe

Lest we forget: Cold-boot attacks on scrambled DDR3 memory

Johannes Bauer*, Michael Gruhn**, Felix C. Freiling***

Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstr. 3, 91058 Erlangen, Germany

Keywords:
Cold-boot memory acquisition
Scraping
Scrambling
Whitening
Decryption

A B S T R A C T

As hard disk encryption, RAM disks, persistent data avoidance technology and memory-only malware become more widespread, memory analysis becomes more important. Cold-boot attacks are a software-independent method for such memory acquisition. However, on newer Intel computer systems the RAM contents are scrambled to minimize undesirable parasitic effects of semiconductors. We present a descrambling attack that requires at most 128 bytes of known plaintext within the image in order to perform full recovery. We further refine this attack using the mathematical relationships within the key stream to at most 50 bytes of known plaintext for a dual memory channel system. We therefore enable cold-boot attacks on systems employing Intel's memory scrambling technology.

Introduction

For several reasons, the contents of volatile memory (RAM) are a valuable piece of digital evidence during a forensic investigation. Firstly, the keys for full disk encryption are usually stored in RAM. Extracting such keys from a memory snapshot therefore allows to access contents of encrypted storage that would be inaccessible otherwise. Secondly, a plethora of other information about the current system state can be recovered from RAM, including ephemeral cryptographic communication keys, the list of running processes and the details of active network connections. Last but not least, new forms of memory-only malware can only be analyzed while they are active in memory. So overall, with the increasing use of encryption technology, cloud storage and memory-only malware, forensic memory image acquisition has become increasingly important.

There has been a lot of debate on how to properly perform imaging of volatile memory since there exist a variety of options (Vömel and Freiling, 2011). One commonly chosen option is runtime acquisition via software using specific memory acquisition tools like WinPmem.[1] While this method appears convenient in many cases, memory imaging may be manipulated by malware that hides in inaccessible memory regions (Stüttgen and Cohen, 2013), thus creating a memory image that is not forensically sound. Another problem of software acquisition methods is that they operate concurrent to regular system activity and therefore produce inconsistencies that do not occur in “perfect” memory snapshots (Vömel and Freiling, 2012). A generic option that avoids these problems is to perform a so-called *cold boot attack*. These attacks exploit the *remanence effect* of modern RAM technology.

Modern RAM technology is commonly based on dynamic random access memory (DRAM), a type of RAM in

* Corresponding author.
** Corresponding author.
*** Corresponding author.
E-mail addresses: johannes.bauer@cs.fau.de (J. Bauer), michael.gruhn@cs.fau.de (M. Gruhn), felix.freiling@cs.fau.de (F.C. Freiling).

[1] http://www.rekall-forensic.com/.

http://dx.doi.org/10.1016/j.diin.2016.01.009

which the cells which store data are constituted of an array of capacitors. Each capacitor is either charged or discharged, depending on whether the cell bit is set or cleared. Since capacitors have a leakage current, their data content slowly dissipates over time. Therefore, in order to effectively use DRAM, each and every cell has to be periodically refreshed. This is achieved by reading the contents and writing it back to the RAM chip. The time that DRAM will keep its contents without leakage affecting the content is referred to as *retention time*. The fact that the retention time is nonzero and that memory will keep its contents for a while even when it is not actively refreshed, is often referred to as the *remanence effect*. It is well-known from electrical engineering that the leakage current of capacitors grows exponentially with their temperature (Wyns and Anderson, 1989). Therefore the retention time of RAM dramatically decreases with increased chip temperature.

Cold boot attacks exploit the remanence effect and can be executed in two ways: One way (Halderman et al., 2009) is to reset the target computer by using the reset button and boot from an alternative medium such as USB using a special imaging USB stick that contains only a minimal operating system together with imaging software such as memimage (Halderman et al., 2009). Ideally, the original contents of RAM are maintained and can be recovered, apart from those parts that have been overwritten by the acquisition software. Unfortunately, such an attack is easily thwarted by using trivial protection mechanisms like BIOS passwords.

The other way to perform cold boot attacks is to physically "transplant" the RAM module at runtime from the device under investigation into an acquisition computer and perform image extraction on that second computer. When the semiconductors are properly cooled before transplantation, they will retain most of their content. It is therefore beneficial to freeze the DRAM modules using cooling spray in order to increase the remanence effect. To the best of our knowledge, this second form of cold boot attack is paradigmatic for the class of memory acquisition procedures since it combines genericity, availability and offers the highest level of integrity and atomicity for the acquired memory images (Vömel and Freiling, 2011, 2012).

While the remanence effect itself is already well-studied (Hamamoto et al., 1998; Liu et al., 2013; Halderman et al., 2009) were the first to exploit it to attack full disc encryption systems on desktop PCs. However, it has been shown that cold boot attacks also affect a multitude of other devices such as smart phones (Müller and Spreitzenbarth, 2013). The dominating RAM technology at that time was called DDR2. Recently, however, Gruhn and Müller (2013) reported that the results of Halderman et al. (2009) could not be repeated with the more modern DDR3 RAM technology. Even worse, the memory images obtained from cold booting DDR3 devices appeared to be random for reasons inherent in that technology.

With increasing speed of semiconductors, their undesirable parasitic effects also grow in magnitude. Current spikes and electromagnetic interference in the speed categories of DDR3 start to affect reliability and regulatory compliance. To counteract this, RAM manufacturers in general and Intel in particular perform memory scrambling in DDR3 memory controllers. As we explain later (and as observed by Gruhn and Müller (2013)), these scramblers severely limit the potential of forensic image acquisition. While Lindenlauf et al. (2015) performed measurements of the remanence effect magnitudes with DDR3 memory, they excluded a discussion of memory scrambling. So, to the best of our knowledge, there is no published work which investigates the possibilities of performing cold boot attacks on modern DDR3 systems in general, i.e., possibilities to "descramble" the scrambler effects.

van Zandwijk (2015) recently performed some related work using an analytic approach to descrambling NAND flash chips. Their approach, however, is unfortunately not easily transferable to RAM acquisition since they require error-free source bit streams which cannot be guaranteed for a cold boot process. Faintly related is also work by Rahmati et al. (2012) which use the remanence effect in a constructive way to increase security of embedded systems. For completeness, we also mention work by Kim et al. (2014) because they also exploit physical properties of RAM semiconductors to provoke bit flips within the RAM modules.

To summarize, the ramification of memory scrambling is currently not well understood and there is no general method of performing cold boot attacks on scrambled DDR3 memory. In this paper, we present an in-depth analysis of DDR3 scrambling (using the Intel memory scrambler as an example) and we show how to use this knowledge to develop a practical method of descrambling DDR3 memory in real-world scenarios.

Contributions

Our contributions which we make in this paper are as follows:

- We present a template attack on scrambled DDR3 memory systems which requires 64 bytes of known plaintext per memory channel (i.e., at most 128 bytes for a dual-channel system) within the memory snapshot in order to yield complete descrambling of the image.
- We refine this template attack by exploiting the mathematical relationships present within the key stream to reduce the number of known plaintext bytes to only 25 (i.e., 50 bytes for a dual-channel system).
- We present methods which can be used to deinterleave dual-channel memory and give an algorithm which can construct an interleaved key stream of arbitrary length from the subkeys for each channel.

We make the source code and documentation of our research freely available to the community at https://www1.informatik.uni-erlangen.de/filepool/mem/ddr3descramble.tar.gz.

Outline

We first give some necessary background information in Section Background, then precisely formulate the problem of descrambling DDR3 memory in Section Problem description. We then show how the problem can be

solved in Section Towards descrambling and present some experimental results confirming our findings in Section Experimental results. We finally conclude in Section Conclusions and Outlook.

Background

Scrambling

Storage of bit streams which are strongly biased towards zero or one can lead to a multitude of practical problems: Modification of data within such a biased bit stream can lead to comparatively high peak currents when bits are toggled. These current spikes cause problems in electronic systems such as stronger electromagnetic emission and decreased reliability. In contrast, when streams without DC-bias are used, the current when working with those storage semiconductors is, on average, half of the expected maximum.

Scrambling can be applied to biased bit streams to remove these undesirable side effects. In the simplest case such a scrambler is a pseudo-random binary sequence (PRBS) that is added onto the input bit stream using the exclusive or (XOR) addition. We call these devices *additive* or *synchronous scramblers*. For the receiving side, the inverse operation, *descrambling*, must be applied in order to get the original content. A pleasant side effect of the XOR operation is that scrambling and descrambling is symmetric; it effectively is the same operation.

Linear-feedback shift registers

Since the goal of scrambling is only to achieve bias removal, cryptographic requirements need not be satisfied by a scrambling PRBS. A solution that is often chosen because of the simplicity and efficiency with which it can be implemented in hardware are linear-feedback shift registers (LFSRs). As the name suggest, these hardware building blocks are shift registers of a fixed bit width n. At any given point in time, the content of the register is referred to as the internal *state*. The bit which is fed back to the register on each clock cycle is a linear combination, i.e., XOR, of some of the state bits. Bits which are inputs to the linear function are called *taps*.

Mathematically, LFSRs can be modeled as polynomials over $\mathbb{F}_2$ in which the state bits $b_0,\ldots,b_{n-1}$ represented the polynomial coefficients. Clocking corresponds to repeated multiplication of the whole polynomial with x and reduction modulo the *characteristic polynomial* P of the LFSR.

Since an LFSR only has 2^n possible states, it will always produce a periodical bit sequence. The period of an LFSR with a primitive characteristic polynomial is 2^n-1 and it is generated for every nonzero initialization of the shift register. One possible hardware instance of such a shift register is shown in Fig. 1.

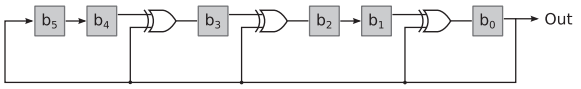


Fig. 1. LFSR with polynomial $x^6 + x^4 + x^3 + x + 1$.

DRAM

As explained before, DRAM needs to be continuously refreshed in order to keep its content. For modern DRAM generations such as DDR3 this process is implemented by logic within the DRAM module itself (Micron, 2014). It is, however, *triggered* by the host. In Intel systems, this is the task of the memory controller hub (MCH). While the CPU and MCH were separate chips in early hardware generations, the MCH is completely contained within modern CPUs.

When the retention time is exceeded – for example because the DRAM chip is unpowered – the stored charge of the capacitors slowly decays and the RAM takes its *ground state*. The ground state is not a trivial pattern of, for example, all zero bits, but depends on the physical construction of the chip itself. Namely, whether the statically connected sides of the cells are biased against GND or against the positive supply voltage, Vcc. The memory pattern that a DRAM chip will therefore show when it far exceeded its retention time is heavily dependent on the physical semiconductor construction.

Another important aspect is the mapping between physical addresses and physical storage location within memory modules. When more than one memory module is present in a computer, the RAM is usually operated in so-called *dual-channel* mode. In this mode, consecutive data is alternated in a certain pattern between modules. This is done for performance reasons, as it allows use of two memory modules in parallel.

LFSR RAM scrambling

We now take a look at the actual process implemented by the MCH construction of Intel. As the authors explain in the patent on the topic, their aim is to reduce both electromagnetic interference (EMI) and current spikes by scrambling (Mozak, 2011; Falconer et al., 2013).

In order to achieve this, they use a set of parallel LFSRs which generate a PRBS that is XORed with the data. Therefore the data on the memory bus appears to be random and ideally exhibits no bit-bias. To be able to generate the PRBS for an arbitrary memory location, a short secret value, the so-called *seed*, is chosen on first power-up. Upon a read or write request to the RAM, this seed is mixed together with the memory address that the request was made to. Fig. 2 illustrates this process schematically. This combined value then serves as the parameterization of the LFSR that generates the PRBS for the requested memory access. In this way, it is possible for the MCH to create the required PRBS for every random memory request. While the Intel patent (Falconer et al., 2013) gives an example seed calculation in which only the lesser significant bits of the address are involved to parameterize the localized PRBS, it is unclear if this is indeed the method that is used in practice.

Note that when memory is operated in dual-channel mode, as explained in Section DRAM, each of the two channels will have its own scrambler and both scramblers

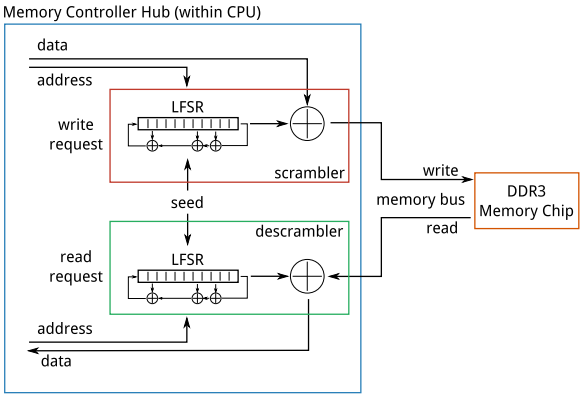


Fig. 2. Schematic display of a Intel DDR3 scrambler.

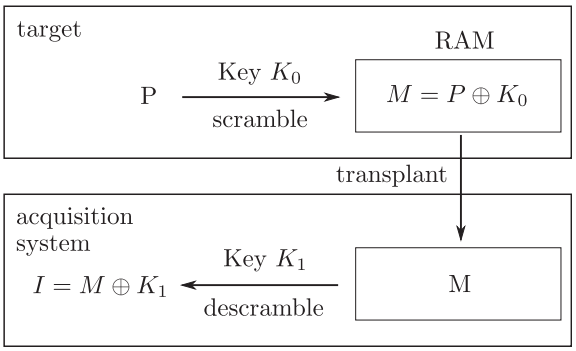


Fig. 3. Scrambled storage of data and image acquisition.

will also usually have distinct seed values. They are two completely independent hardware units.

During the boot process, the MCH is programmed by code that is part of the computer's firmware (i.e., the BIOS or UEFI). It is during this initialization that the MCH parameters – including the scrambler configuration and scrambler seed – are programmed as well. Therefore a reseeding of the MCH scrambler can only occur when the computer is performing a cold start (i.e., transitions from unpowered to powered state). In our trials, the seed was never reset when the computer was merely rebooted using the reset button.

Problem description

As mentioned in the introduction, one approach to perform image acquisition is to reset the target computer by using the reset button (thereby *not* reinitializing any memory scramblers) and boot from an alternative medium such as USB. Tools like the memimage imaging software (Halderman et al., 2009) have a sufficiently small memory footprint as to leave most of the memory contents intact. However, such attacks are easily thwarted using BIOS passwords for example. The other approach is to physically transplant the RAM module from the suspect's computer into an acquisition computer and perform image extraction on that second computer, as shown in Fig. 3. It is in this case that scrambling comes into full effect, since the contents of the RAM chip will contain only scrambled data and the acquisition computer cannot have the correct seed to replicate the original key stream.

As shown in Fig. 2, all data that is passed to or from the DRAM chips is first passed through the scrambler circuitry within the memory controller hub. It is transparently scrambled when writing to and transparently descrambled when reading from RAM. Therefore at any point in time, the RAM module will only contain a scrambled image M. All connected peripherals will not be aware that scrambling is even happening, but will only see the plain RAM image P. The scrambling data stream will be referred to as K and the connection between the three is simply an XOR relationship, as in a stream cipher: $M = P \oplus K$. When such an image is forensically acquired via means of cold booting, the captured image is referred to as I. While during normal operation of the computer the key K_0 was used by the scrambler, during the image acquisition phase this key might be K_1, where K_0 is not necessarily equal to K_1. Fig. 3 shows this formally: During the normal use, the plain image P is scrambled by K_0 and the memory M consists of the value $K_0 \oplus P$ which resides in RAM:

$$P \underset{scramble}{\overset{K_0}{\rightarrow}} P \oplus K_0 = M \tag{1}$$

When the RAM module is transplanted to the analysis machine, generally the scrambling key will be different. We denote the new key by K_1. Subsequently, during the acquisition phase the descrambler adds K_1 to the RAM image, yielding an image

$$I = P \oplus K_0 \oplus K_1 = M \oplus K_1 \underset{descramble}{\overset{K_1}{\leftarrow}} M \tag{2}$$

Therefore, in a cold boot scenario the descrambler does not do what the name suggests (i.e., recover the original plaintext image), but instead actually adds another layer of scrambling on top of the already scrambled image. During normal operation of the computer, K_0 is equal to K_1 so that the key streams cancel each other out and the descrambler actually recovers the plaintext image.

Intel's patent on their scrambler mechanics explains that a parallel LFSR is used to generate the scrambler bit stream K. Therefore it seems that decrypting such a scrambled image would be a rather simple, straightforward task. Surprisingly enough, in practice it turns out to be more complicated than initially anticipated. This has a number of reasons:

1. *Nonexistent public documentation*: All documentation that explains the registers which are used by the memory controller hub (MCH) – the component that contains the scrambling unit – is non-public. Parts of the documentation which are publicly available, such as the patents Intel has filed on the issue (Mozak, 2011; Falconer et al., 2013), are worded as broadly as possible to include a plethora of different options. It is unclear which one is used in practice.

2. *Lossy image acquisition*: Forensic image acquisition when using cold boot techniques relies on the remanence effect of the semiconductors. This effect is neither guaranteed nor reliable. Bit flips occur frequently as the DRAM cells lose their content. When trying to reverse engineer the used scrambling mechanisms, this poses a problem since algorithms like Berlekamp-Massey (Massey, 1969) for synthesis of a LFSR from a given bit stream rely on perfect input data to produce correct results. When the algorithms are fed noisy input they will not indicate failure, but instead synthesize misleading output.
3. *Unknown ground state*: If the DRAM content of a chip which was inactive for a long time, i.e., the ground state of that chip G, were known, then it would be easy to determine the pure scrambler bit stream. We could perform a cold boot attack on a machine that had been turned off for a long time. While then assuming that $M = G$, we can determine $K = G \oplus I$, since for this machine the memory content C would be equal to G and it would run through the descrambler during forensic image acquisition. However, the ground state G is highly dependent on the actual constitution of the hardware itself and forms a nontrivial pattern. Therefore it is difficult to gain access to the pure scrambling bit stream.
4. *Interleaved memory*: Lastly, most modern systems with more than one physical RAM chip will be configured to use dual channel mode in order to improve system performance. This means that consecutive data will be put on alternating RAM modules in an unknown pattern. Since each channel has its own, completely separate scrambler instance, it must be known which pieces of data have been scrambled by which unit in order to perform descrambling for such images.

Towards descrambling

We now describe an approach to descramble the contents of a DDR3 memory image that was acquired using cold boot. The first steps which we describe are necessary to calculate certain parameters of the hardware that are necessary for descrambling to work. These steps can, however, also be performed *after* the memory image has been taken.

Calculating memory offsets

As mentioned before, the scrambler LFSR is parameterized by a global seed and (parts of) the memory address that is accessed. It is therefore vital to know the exact physical memory address of every byte in the acquired image. Unfortunately, not all acquisition software works reliably; in fact, there are many examples in which areas of memory which are inaccessible are simply skipped instead of being correctly filled with padding data (Vömel and Stüttgen, 2013). When scanning through plaintext images in order to locate cryptographic keys, this is not a problem. For our purposes, however, it is not only important to get the data, but also important to be able to pinpoint the exact storage location of that data. Only then can we select the correct key stream offset with which we will be able to descramble the image. In order to work around inherent limitations of acquisition software, we wrote a custom *data placer* program to store the 64-bit physical address every 8 bytes throughout all available memory. Then a soft reset was performed as to not reseed the memory scrambler and a forensic image was created with the same software which would later also capture the data images.

Note that using this approach we would also be able to determine the effects of memory address scrambling when performing acquisition on transplanted memory. While this was not the case in our experiments, there are indeed hints in literature that this would be something that could be expected in the future (Gould, 2009).

By examining these dumped images, it was easy to identify the locations where the acquisition process was discontinuous. These discontinuities are referred to as *hidden memory* regions (Stüttgen, 2015; Stüttgen et al., 2015). This is expected, as the BIOS memory map (which the acquisition software usually relies on) only loosely correlates with the intricate details of the actual memory mapping (which the MCH uses). As a consequence, such forensic images may contain holes within where hidden memory regions were present. Capturing exactly where these discontinuities were located for any given combination of computer and DRAM allowed us to calculate the actual address in the original physical memory of a given offset within the dumped memory image file.

Distinguishing the scrambler type

Now we know which addresses in physical memory map to an acquired image file offset, but we do not yet know about the scrambling behavior of the device under test at all. We now show how it is possible to determine how the scrambler is configured by the computer's firmware.

Here is the procedure to distinguish the different scrambler types (see Fig. 4):

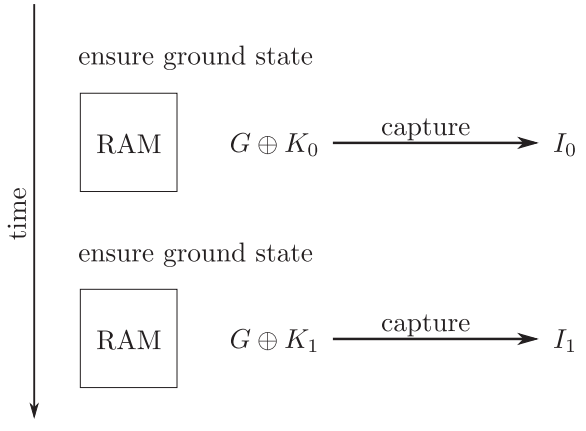


Fig. 4. Experimental setup for image recovery.

1. Turn the device completely off and leave it off for an extended period of time (e.g., 1 min). This ensures that the DRAM content will definitely be the ground state G.
2. Turn the device on and immediately perform cold boot image acquisition.
3. Repeat these two steps twice to capture two independent cold boot images I_0 and I_1.

To determine the scrambler type, it is now sufficient to investigate the image content which has not been modified by the BIOS or dumping software. (Note that the amount of RAM that is overwritten by the BIOS on reboot needs to be evaluated on an individual basis.) When analyzing this memory, there are three possible outcomes:

1. The two captured images I_0 and I_1 are identical (except for noise) and look non-random. Long sequences of consecutive 0 and 1 bits are expected to be present in the output. The details of the pattern depend on the physical hardware wiring of the respective memory cells. This finding implies that *scrambling* is *disabled* on the machine.
2. The captured images are identical (except for noise), but look random (i.e., equal distribution of all bytes with approximately identical probability). This implies that *scrambling with a constant seed* is used on the machine. Note that disabled scrambling is a special case of constant scrambling, where the constant scrambling key stream $K_C = \vec{0}$.
3. The captured images look completely different and also both look random. This implies that *scrambling with a random seed* is used on the machine.

In our experiments, we did not find any machine which disabled the scrambling feature altogether. But there were machines of either of the two latter types. Which type a machine belongs to is determined by the system firmware, as explained earlier in Section LFSR RAM scrambling. For example, we found a system consisting of an MSI H55M-P33 mainboard with an Intel Core i5-760 CPU to use constant scrambling, while an Intel Core i3-3225 CPU within a MSI B75MA-P45 mainboard used random scrambling.

Attacking constant scrambling

Assume a machine that performs constant scrambling. For such a computer, descrambling the memory contents is not necessary in most scenarios, since the scrambling and descrambling key K is, as the name suggest, constant over power cycles. Therefore, $K = K_0 = K_1$ and therefore $I = P \oplus K_0 \oplus K_1 = P$. The intuitive reason is that scrambling and descrambling cancel each other out on the same system when the key stream remains the same for both. Therefore, if the forensic image acquisition is performed on the same computer which also wrote the data into the RAM, the system can be treated as if there were no scrambling used at all. This also applies to RAM that was transplanted from one system to another one which uses the exact same firmware and therefore same, constant, scrambler seed.

Unfortunately, in most practical cases the machine with which the image recovery was performed will be different from the computer which contains the forensically interesting data. In such cases, things become more complicated. Fortunately, for all hardware that we tested, the basic principle of how the scrambler works was always identical, so there is reason to assume that there currently is only one generation of scrambling hardware available. This is also the case which we assume and deal with here. If the two computers use different scrambler generations, the results could vary greatly depending on the exact scrambler mechanisms which are employed.

Assuming that the two computers use at least the same scrambler generation and merely differ in the parameterization (e.g., the host system uses constant scrambling with K_0, but the acquisition system uses a different key stream K_1), then the captured image can simply be treated as if it were created on a randomized scrambling system, as described next.

Attacking randomized scrambling

Consider again the two images I_0 and I_1 from Fig. 4. They both capture the same, unknown, ground state G with different scrambling keys applied to them. In other words,

$$I_0 = K_0 \oplus G \quad \text{and} \quad I_1 = K_1 \oplus G.$$

We can therefore apply a differential approach by constructing the image D

$$D = I_0 \oplus I_1 = K_0 \oplus G \oplus K_1 \oplus G = K_0 \oplus K_1$$

By eliminating the unknown ground state from the equation we now only deal with the differential of two unknown key streams. Since we know that the key stream is periodic, as explained in Sect. Linear-feedback shift registers, it can be written as the repeated concatenation of some unknown partial key stream S (the *subkey*):

$$D = S^x$$

We then inspect chunks from this D of varying size (concretely, we used powers of two from 32…1024). Using autocorrelation on these chunks we can identify the periodicity π of S within D. To do this, we define an equality function on two bit streams X, Y of equal length:

$$X \approx Y \Leftrightarrow \frac{\mathrm{H}(X \oplus Y)}{|X|} < \varepsilon$$

Here, H is the Hamming weight of a bit vector. Intuitively, we consider two bit streams X,Y to be approximately equal if and only if the average Hamming weight of their bitwise difference is below a certain threshold ε. We group n of these chunks into an equivalence class:

$$\{C_0, C_1, \ldots, C_{n-1}\} \quad \text{with} \quad C_i \approx C_j \;\forall\; i,j \in \{0, \ldots, n-1\}$$

Once the periodicity π is selected correctly, only one equivalence class will emerge with lots of approximately equal differential subkey candidates C_i, all of length π.

During our experiments, we determined the smallest value for π at which this occurred to be 64 bytes.

Due to bit flips during the lossy acquisition, we still do not, however, know S. Under the assumption that all candidates C_i are just deviations from S caused by random noise during acquisition, we can calculate S by performing a majority vote on each individual bit s_j of S:

$$s_j = \begin{cases} 0 & \text{if } \sum_{i=0}^{n-1} C_{i_j} < \frac{n}{2} \\ 1 & \text{otherwise} \end{cases}$$

As a result, we have the most likely subkey candidate S, where $D = K_0 \oplus K_1 = S^x$. We can now use this information to recover P using a known plaintext attack.

Stencil attack

From our measurements we found that on all machines we investigated, the differential of two keys K_0 and K_1 exhibited a 64-byte periodicity (i.e., $\pi = 64$). This directly enables what we refer to as the *stencil attack* for DDR3 descrambling. The attack works as follows:

1. Perform forensic recovery of the image that shall be descrambled. Without loss of generality, the memory image content M of the image P is $M = P \oplus K_0$. Here, K_0 is the key stream that is applied to the data by the scrambler unit on the target system.
2. The captured image will be $I = P \oplus K_0 \oplus K_1$, with both K_0 and K_1 unknown. Note that the descrambled image P is the information of actual forensic interest and K_1 is the key stream added by the descrambler unit on the acquisition system.
3. Therefore, $I = P \oplus (K_0 \oplus K_1) = P \oplus D$, where D is still unknown. However, we know from Sect. 4.4 the periodicity π of D. Therefore, scan through the image I at π-byte boundaries and cluster π-byte chunks together using the approximative equality function described above. Select a partition that has lots of candidates: this is likely to be a pattern of all 0×00. Construct the maximum likelihood candidate S by majority vote.
4. Construct $P = I \oplus S^x \oplus T^x$ where T is the known plaintext. If the known plaintext was a chunk of 0×00, i.e., $T = \overrightarrow{0}$ then $P = I \oplus S^x$.

To reconstruct P, we therefore only need a known plaintext of length π, i.e., 64 bytes in our case.

Mathematical approach

The stencil attack allows an attack to be mounted against scrambled DDR3 memory, effectively yielding the original image with relatively few pieces of known plaintext required. However, we now look at the mathematical relationships within the differential subkey stream with the purpose of constructing a key stream from less known plaintext.

In our approach we are limited to examination of a differential key stream $K_0 \oplus K_1$. This is as an inherent limitation of performing acquisition with systems which contain an active descrambler. During analysis, we found some interesting congruencies within this differential key stream. First, we partitioned the 64-byte differential key stream stencil into 32 values of 2 bytes each. Each value was interpreted as a little endian integer. We ended up with 32 16-bit integer values $v_0,\ldots,v_{31}$. We then were able to find three 16-byte polynomials p_0,p_1,p_2 for which 24 congruencies hold for $0 \le i \le 7$ and $0 \le j \le 2$:

$$\left((v_{4i+j} \gg 1) \oplus p_j\right) \& 0\text{x}7\text{fff} = v_{4i+j+1}$$

If you recall Section Linear-feedback shift registers, this relationship can immediately be recognized as a LFSR relationship: Two related values, v_{4i+j} and its adjacent word v_{4i+j+1} can be constructed from each other when the first one is shifted right by one bit ($\gg 1$) and then has the LFSR polynomial added onto it ($\oplus\ p_j$). Note however that in this congruence we can only determine 15 of the 16 bits of the adjacent word (& 0x7fff). This is because it is lost when the difference between two LFSR output streams is constructed.

These relationships are a useful additional method to confirm the validity of key streams and aid the search for a known plaintext in the differential memory snapshot D. It reduces the number of known plaintext bits to less than 40% of the original stencil attack: in our case, instead of 64 bytes we now need only $(8 + 3) \cdot 2 + 3 = 25$ bytes if we exploit the mathematical inter-stream relationships. Only the eight initial states v_{4i} (16 bytes), the three polynomials $p_0 \ldots p_3$ (6 bytes) and the three most significant bits in every group of 8 bytes (total of 3 bytes) need to be known to construct the entire differential key stream.

Deinterleaving of memory

As stated before in Sects. 2.3 and 2.4 a system with more than one DRAM module will usually operate in dual-channel mode to improve system performance. Each memory channel has an independent scrambler, so the attack as described in Section Stencil attack still works when it is known which part of the memory image needs to be descrambled with which channel key. In our experiments we determined how the algorithm to split data between channels works in Intel systems. Consider two channel subkeys A and B of 64 bytes each. The two basic interleaved streams Q_1 and R_1, each of length 256 bytes, are defined to be:

$$Q_1 = A^2 || B^2$$

$$R_1 = B^2 || A^2$$

All further key streams are then defined recursively:

$$Q_n = Q_{n-1} || R_{n-1}^2 || Q_{n-1}$$

$$R_n = R_{n-1} || Q_{n-1}^2 || R_{n-1}$$

This definition can be applied until one finds a Q_n of sufficient length. This Q_n is then the complete key stream K.

Since every channel subkey is 64 bytes in length, the length of an interleaved key stream Q_i is exactly:

$$|Q_i| = 64 \cdot 4^i$$

Solving for i with given $|Q_i|$:

$$i = \log_4 \frac{|Q_i|}{64}$$

i.e., for a dual-channel memory system of 4 GiB, one would choose $i = 13$.

The interleaving pattern for $i = 6$, i.e., for 4096 streams of 64 bytes each, is graphically shown in Fig. 5. It is exactly 64 by 64 pixels (i.e., 4096 pixels) in size and every pixel's color indicates whether stream 0 or stream 1 is in effect. The stream order is shown left-to-right, top-to-bottom.

With this knowledge, we can perform the stencil attack even when RAM is accessed in dual channel mode. The acquired image I has to be deinterleaved into two channel images I_A and I_B which can be treated independently before interleaving them back together to form a plain image P.

Experimental results

Investigated machines

Our measurements were performed predominantly on the Intel Core i3-3225 with a MSI B75MA-P45 mainboard. We took care to verify that the results also apply to different machines. In the process, we confirmed that our results also apply to the following combination of CPUs and mainboards:

- Intel i5-760/MSI H55M-P33
- i5-2520M/Dell 03PH4G
- i5-2400/Esprimo P900 E90+

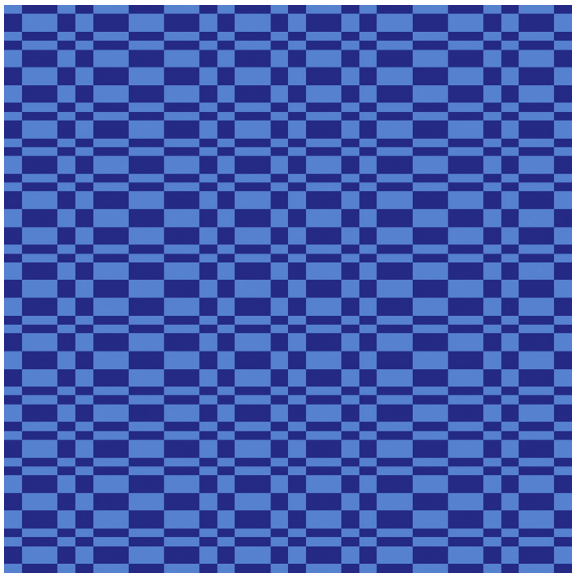


Fig. 5. Dual channel interleaving graphically.

Applying the stencil attack

In our experiments we first started out with a single memory module present in the target computers. We then used our data placer code to place 512 × 775 pixel gray scale images of Mona Lisa at every 1 MiB boundary. At the space in between images we placed an easily recognizable, distinct pattern block.

We then froze the memory module by applying cooling spray to it until it had reached around −30 °C. Then we cut power to the system by shutting it completely off and restarted immediately afterwards. The latency that our targets took from complete shutdown to new boot-up ranged from around 2 to up to 5 s. We then drew memory images using the memimage toolkit. By using the techniques described in Section Stencil attack we were able to recover the original memory image.

You can see the results of our experiments in Fig. 6. The first image, Fig. 6a shows an image acquisition that was

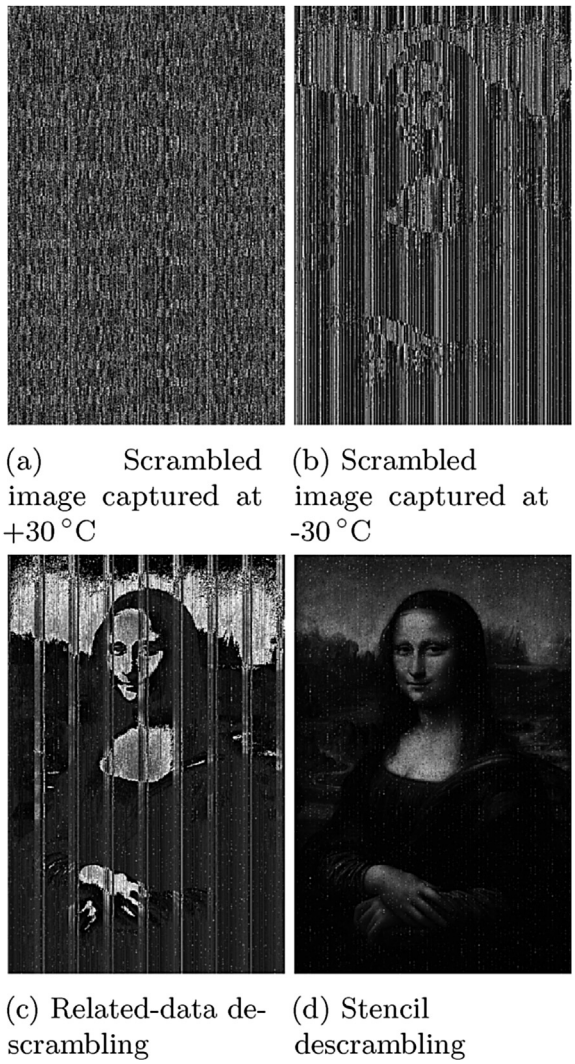


Fig. 6. Images of descrambling single-channel memory.

performed at operating temperature (about +30 °C). No data could be recovered from this test, as everything was completely decayed.

On the second image, Fig. 6b, the image is shown when it was drawn from the target after cooling to about −30 °C was applied. The basic shape of Mona Lisa is still visible, but it is distorted by a repeating pattern. This repeating pattern is exactly the subkey stencil which we need to apply to descramble the memory images. When this key is unknown (because there is no known plaintext or at least no known plaintext yet) we made a related-data experiment, shown in Fig. 6c. For this we make the assumption that consecutive 64-byte plaintexts will – at least to some degree – repeat within the plaintext. Therefore the first 64-byte block was chosen as a stencil subkey. You can see clearly that the image looks a lot better than the completely scrambled variant, but still has lots of distortions.

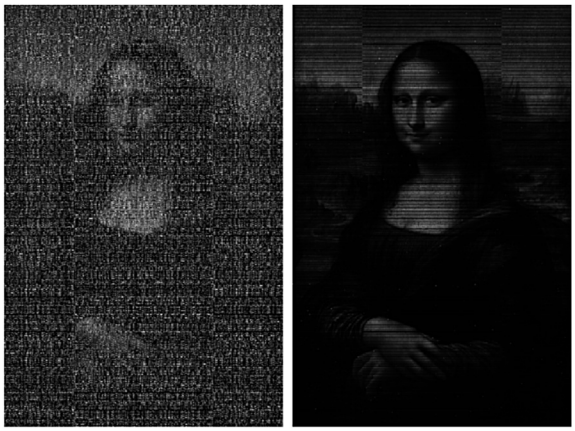
Finally, we used the method described in Section Stencil attack and recovered the most probable key using majority vote and known plaintext. This key was then applied to the captured image, yielding a result that was, except for the occasional bit error, very close to the original image.

Dual channel mode and decay rate

During our experiments we found the high decay rate of DDR3 RAM when at operating temperature curious. To verify our hypothesis that the retention time of memory was indeed much less than what can be seen in DDR2 counterparts, we performed an experiment: We placed two memory modules in our target PC and took care that they were operated in dual-channel mode. Then we placed the Mona Lisa images in RAM using our data placer program. At this point in time, roughly every second 64-byte block of the image is on one RAM module and every other 64-byte block is on the other RAM module. We then inserted a rectangular piece of 500 μm PET (polyethylene terephthalate) in between the two RAM modules to achieve thermal isolation between them. One of the two modules – but not the other – was then frozen by us before performing a cold boot attack.

This experiment served a dual purpose: First, it allowed us to confirm that the thermal dependency of DDR3 is really as critical as we assumed it was. This is because the only difference in the process was the temperature of the modules – all other parameters like the power-down time were exactly identical. Secondly, it allowed us to confirm that our algorithm to decode dual-channel memory, as explained in Section Deinterleaving of memory, worked as expected.

The results are depicted in Fig. 7. On the left side, Fig. 7a shows the interleaved, descrambled, memory image. The results verified our hypothesis: Approximately every other row was completely decayed and shows up as white noise in the image. Even though the image in Fig. 7b gives a rather noisy visual impression this noise is really only present in the parts that were acquired from the warm RAM module. This becomes obvious when the noisy channel is masked out according to the algorithm we presented in Section Deinterleaving of memory. The result is Fig. 7b, a successfully descrambled one-channel image of a RAM module that was operated in dual-channel mode. The image shows virtually no noise because our algorithm correctly masks out only the module which had decayed content.



(a) Interleaved dual channels (b) Deinterleaved masked image

Fig. 7. Images of descrambling dual-channel memory.

Remanence effect in DDR3

In their original paper, Halderman et al. (2009) found that DDR2 RAM exhibits a comparatively strong remanence effect. They performed tests at operating temperature and even without externally applied cooling to the chips, some DDR2 chips had decay times of up to 35 s before showing complete data loss. To determine these values for DDR3 memory, Lindenlauf et al. (2015) did similar decay measurements. They found an astonishingly low bit error rate when the modules were cooled to a temperature between −30 °C and −35 °C and kept the RAM modules unpowered for up to 50 s. It is not apparent, however, if these long decay times were also performed with DDR3 memory or just with DDR2, and while they do show the dependency of the decay rate on the die temperature for DDR2 memory they omit how these results transfer to DDR3 memory.

In our experiments, we found DDR3 memory to be much less forgiving during cold booting. Much in contrast to DDR2 memory it was absolutely essential for us to always keep modules at low temperatures (around −30 °C) in order to produce usable results. We observed retention times of about 10 s before total decay occurred even when such cooling was performed. At operating temperature (around +30 °C) we were not able to acquire a single usable image, because all data content had dissipated.

It is our assumption that this can be explained by the different types of memory modules that were used by Lidenlauf et al. compared to ours. While they used modules that were produced in 2011, we used slightly more recent modules (produced 2013) that were also a bit faster (666 and 833 MHz types).

Conclusions and outlook

Memory acquisition of DDR3 memory in the real-world is more complicated than with a laboratory setup: While in a lab setup researchers can choose systems which work for their demonstration – systems which will usually use constant scrambling – this luxury is not available in a real-world scenario. On top of the intricacy of descrambling images come practical aspects like dual-channel decoding. Both are obstacles that are not in place to deter cold boot attacks, but they still complicate memory acquisition significantly in practice.

We have demonstrated that our explanations and assumptions about the internal construction of the Intel DRAM scrambler are in line with the observations we made from our experimental results. Cracking a dual-channel system requires only 128 bytes of known plaintext to apply our stencil method and only 50 bytes if the mathematical approach is chosen. This is a negligible amount of data compared to the huge amount of RAM that is present in the computer systems of today. Large chunks of the RAM will usually be set to zero in regular operation of a computer, be it either by the operating system or by any running application. We further demonstrated that we are able to correctly deinterleave RAM images. This is a prerequisite to correct descrambling of dual-channel systems, as each channel has an independent scrambler.

Since the operation of memory scramblers is transparent for the system during normal operation, it could well be possible that newer MCH revisions choose to use different mechanisms for scrambling. Of particular interest for forensic investigation would be if our results can be applied to DDR4 memory as well. This is something we would like to explore in future work.

In order to improve on our attack, it would be most interesting to mathematically attack the generated key stream itself. Since our approach only works with differential streams we at no point in time are able to reconstruct the original key stream – we only ever reconstruct differential key streams. Our ideas for future work are to utilize custom-built hardware around an FPGA development board in order to be able to read out the raw key stream from a cold booted DDR3 memory module. This would then enable brute forcing of key streams by trying different seeds, but it would also be an attack that would be significantly more difficult than what we show in this work.

Investing this time would be interesting not only from an academical standpoint, but also within a real-world scenario. The reason that scramblers are present in the first place is because Intel deemed it necessary to limit excessive current spikes on the memory bus and in the memory modules. This leads us to believe that there could be possibly exploitable detrimental effects if one could purposefully produce these excessive current spikes. If the scrambling key stream were known to an attacker, this could be leveraged from any unprivileged application to mount an attack which aims to distort RAM integrity. Since disturbing RAM-integrity is a relevant topic and is receiving increased attention after the inspiring row hammer attacks of Kim et al. (2014), this might prove to be a worthwhile investment of time after all.

References

M. Falconer, C. Mozak, A. Norman, Suppressing power supply noise using data scrambling in double data rate memory systems, US Patent 8,503,678 (Aug. 6 2013). URL http://www.google.com.ar/patents/US8503678.

G. Gould, Address scrambling to simplify memory controller's address output multiplexer, US Patent 7,493,467 (Feb. 17 2009). URL http://www.google.com/patents/US7493467.

Gruhn M, Müller T. On the practicability of cold boot attacks. In: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, IEEE; 2013. p. 390–7.

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, et al. Lest we remember: cold-boot attacks on encryption keys. Commun ACM 2009;52(5):91–8.

Hamamoto T, Sugiura S, Sawada S. On the retention time distribution of dynamic random access memory (DRAM). Electron Devices IEEE Trans 1998;45(6):1300–9.

Kim Y, Daly R, Kim J, Fallin C, Lee JH, Lee D, et al. 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. IEEE Press; 2014. p. 361–72.

Lindenlauf S, Höfken H, Schuba M. Cold boot attacks on DDR2 and DDR3 SDRAM. In: Availability, Reliability and Security (ARES), 2015 10th International Conference on, IEEE; 2015. p. 287–92.

Liu J, Jaiyen B, Kim Y, Wilkerson C, Mutlu O. An experimental study of data retention behavior in modern dram devices: implications for retention time profiling mechanisms. ACM SIGARCH Comput Archit News 2013;41(3):60–71.

Massey JL. Shift-register synthesis and BCH decoding. Inf Theory IEEE Trans 1969;15(1):122–7.

Micron. DDR3 SDRAM datasheet for MT41J256M4, MT41J128M4 and MT41J64M4. 2014. URL, https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/1gb_ddr3_sdram.pdf.

C. Mozak, Suppressing power supply noise using data scrambling in double data rate memory systems, US Patent 7,945,050 (May 17 2011). URL https://www.google.com.ar/patents/US7945050.

Müller T, Spreitzenbarth M. Frost: forensic recovery of scrambled telephones. In: Applied cryptography and network security. Springer; 2013. p. 373–88.

Rahmati A, Salajegheh M, Holcomb D, Sorber J, Burleson WP, Fu K. TARDIS: time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In: Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), USENIX, Bellevue, WA; 2012. p. 221–36. URL, https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/rahmati.

Stüttgen J. On the viability of memory forensics in compromised environments. Ph.D. thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Department of Computer Science; 2015.

Stüttgen J, Cohen M. Anti-forensic resilient memory acquisition. Digit Investig 2013;10:S105–15.

Stüttgen J, Vömel S, Denzel M. Acquisition and analysis of compromised firmware using memory forensics. Digit Investig 2015;12(Suppl. 1):S50–60. http://dx.doi.org/10.1016/j.diin.2015.01.010.

van Zandwijk JP. A mathematical approach to NAND flash-memory descrambling and decoding. Digit Investig 2015;12:41–52.

Vömel S, Freiling FC. A survey of main memory acquisition and analysis techniques for the windows operating system. Digit Investig 2011;8(1):3–22. http://dx.doi.org/10.1016/j.diin.2011.06.002. URL, http://dx.doi.org/10.1016/j.diin.2011.06.002.

Vömel S, Freiling FC. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. Digit Investig 2012;9(2):125–37.

Vömel S, Stöttgen J. An evaluation platform for forensic memory acquisition software. Digit Investig 2013;10(Supplement):S30–40. http://dx.doi.org/10.1016/j.diin.2013.06.004.

Wyns P, Anderson RL. Low-temperature operation of silicon dynamic random-access memories. Electron Devices IEEE Trans 1989;36(8):1423–8.