Robust covert channels based on DRAM power consumption*

Thales Bandiera Paiva[1], Javier Navaridas[2], and Routo Terada[1]

[1] Institute of Mathematics and Statistics, University of Sao Paulo, Sao Paulo, Brazil
{tpaiva,rt}@ime.usp.br
[2] School of Computer Science, University of Manchester, Manchester, U.K.
javier.navaridas@manchester.ac.uk

Abstract. To improve the energy efficiency of computing systems, modern CPUs provide registers that give estimates on the power consumption. However, the ability to read the power consumption introduces one class of security concerns called covert channels, which are communication channels that enable one process to transmit a message to another one in a system where these processes were meant to be isolated. Our contribution consists in the first covert channel in which messages are transmitted by modulating the DRAM power consumption. The channel implementation outperforms similar proposals, achieving 1800 bps with 10% error, and 2400 bps with 15% error, when running on a notebook and on a desktop platforms, respectively, To test its robustness against application interference, we considered the channel's performance when running concurrently with different benchmarks: MRBench, Terasort and LINPACK. When running on the notebook, the channel is fairly robust, achieving between 300 and 600 bps with around 10% error depending on the workload considered.

Keywords: Covert channel · Intel RAPL · Power consumption.

1 Introduction

Power consumption is a major concern for both small embedded devices and huge clusters. In the first case, the low battery life is a major hardware constraint, while in the second, the amount of power needed by HPC environments can make its use unfeasible, or at least unprofitable. To improve the energy efficiency of computing systems, it is important to be able to measure and profile the power consumption. One common solution is to use external power measuring tools, such as Watt's Up Pro Power Meter, or PowerMon 2 [2]. Another solution is to use internal power measurement interfaces such as Running Average Power Limit (RAPL), introduced by Intel on SandyBridge chips, or the AMD Application Power Management.

* T. B. Paiva is supported by CAPES. R. Terada is supported by CNPq grant number 442014/2014-7.

The RAPL components provide a software interface for the user to obtain estimates on the processor's power consumption based on models validated by Intel. Since its inception, researchers have studied how accurately RAPL measurements correspond to real power consumption and how one can use this interface to efficiently profile an application energy usage [24,5]. Two security concerns that comes with the ability to measure power consumption in software are side-channel attacks and covert channels. With respect to RAPL components, side-channel attacks were first studied by Mantel et al. [16], while covert channels were first considered by Miedl and Thiele [20].

A covert channel enables two processes called the Trojan and the spy to communicate in a system where they are meant to be isolated [13]. As such, this type of channel is a central tool in data exfiltration, where an attacker wants to recover some secret information from a target system without leaving any trace. Figure 1 illustrates the use of a covert channel.

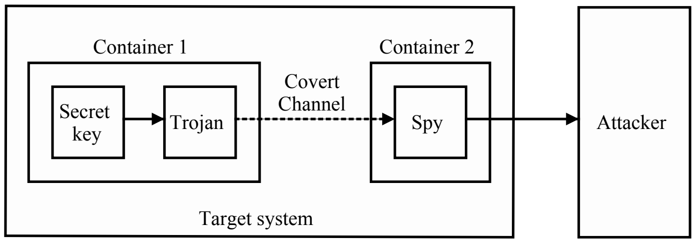

Fig. 1. In this setup, the Trojan and spy run on different isolated containers of the target system. The Trojan obtains some secret information, but because of the system's security policy, unauthorized applications in the container where the secret key reside cannot access the Internet. The Trojan then uses a covert channel to transmit the secret key to the spy process, which has access to the Internet and therefore can transmit the secret key to the attacker.

To successfully build a covert channel, the spy and the Trojan must have access to some shared resource, but not the usual ones such as shared memory, files, or network, because communication over these ones could be detected by security auditors or explicitly prohibited by isolation mechanisms. There are numerous examples of covert channels. Some recent constructions are based on inaudible audio [4], branch predictors [7], core temperature [17], and cache coherence protocols [26].

The US Department of Defense standard Trusted Computer System Evaluation Criteria (TCSEC) [23] defines a high bandwidth covert channel as one which can transmit at rates higher than 100 bps. To protect systems against information leakage, it is important to lower, to 0 if possible, the bandwidths of known covert channels. This can be done by introducing noise, lowering the rate in which the channel can be modulated, or preventing the processes from accessing the resources in which covert channels are based.

In this work, we investigate the possibility of building a covert channel using the DRAM power consumption estimates given by the Intel RAPL interface. In Linux, these estimates can be read using the `powercap` module without any special privileges. Our main motivation is that, for devising mechanisms to protect the systems against these channels without compromises to energy efficiency, it is necessary to understand the true risk posed by these RAPL covert channels.

Contribution. We present the first covert channel based on the RAPL measurement registers for the DRAM power consumption. This covert channel can transmit at 2400 bps, with error rates around 15%. We tested the channel performance under two platforms: a notebook and a desktop computer. The channel robustness against application interference was evaluated under three different workloads: MRBench, Terasort and LINPACK. The channel appears to be much more robust against interference than previous work. In particular, under the notebook setup, the channel achieved high bandwidths between 300 and 600 bps with error rates close to 10%, depending on the workload considered. On the desktop, the covert channel achieves 400 bps under MRBench or Terasort workload with 10% error rate, but it is not reliable when LINPACK is running.

This paper is organized as follows. Section 2 reviews relevant side and covert channels based on power consumption. The threat model and experimental setup are presented in Section 3. In Section 4, we show how to modulate CPU and DRAM power usage and discuss how the benchmarks affect the power consumption. The algorithms for Trojan and spy are presented in Section 5. The performance evaluation of the covert channel is done in Section 6. In Section 7, we consider two methods to achieve higher transmission rates. Section 8 consists of a discussion on our results. We finish with the conclusion and discussion of possible future work in Section 9.

2 Related work

2.1 Side-channel attacks

In a side-channel attack, an attacker obtains secret information by observing or measuring the system when some critical computation is running. Side-channel attacks are one of the main problems faced when implementing cryptographic primitives, since the mathematical security models usually abstract away software, hardware, and implementation details.

In 1999, Kocher et al. [12] presented two powerful techniques for side-channel analysis of cryptographic schemes: SPA (simple power analysis) and DPA (differential power analysis). Both of these attacks work by measuring the power trace of a system when it is performing some cryptographic computation. When performing an SPA, the attacker tries to recover some secret information by directly interpreting the power trace. In contrast, in a DPA, the attacker uses statistical analysis to correlate the power consumption to the data processed. As such, DPA can better eliminate noise in the measurements. Power analysis has been used to attack widely used schemes such as AES [15], RSA [18], and Elliptic Curves [3].

The main limitation of the attacks mentioned above is that direct access to the hardware is needed to perform the measurements. Therefore, it makes them less practical than side-channel attacks based on timing [12] or cache [27,8]. There are some examples of power analysis attacks that do not use dedicated hardware against mobile devices [19,25], where the power information is obtained by monitoring the device's battery.

Closely related to our research is the work by Mantel et al. [16] that shows, to the best of our knowledge, the first energy consumption side-channel attack using the software-based measurements provided by the RAPL interface. In this work, the authors mount a key distinguishing attack against a popular implementation of the RSA cryptosystem. The authors reported that, with only 25 energy consumption observations, an attacker can distinguish between two RSA keys with 99% of success.

2.2 Covert channels

One of the first covert channels related to power usage was introduced in 2006 by Murdoch [21]. In this work, the Trojan, running on a server, modulates a message as the usage of a device to heat it up or cool it down. The spy then recovers the message by observing the clock skew on timestamps of the collected responses from the server that were caused by differences in temperature. The capacity of Murdoch's channel was later estimated as 20.5 bph (bits per hour) by Zander et al. [28].

In 2015, two covert channels directly based on power consumption and measurement were proposed. Guri et al. [9] showed a covert channel achieving between 1 and 8 bph between two separated desktop computers, where the Trojan modulates the heat dissipation, and the spy decodes the message using its hardware's native heat sensors. In 2015, Masti et al [17] showed how to use the CPU core temperature measurements to build a covert channel. One very interesting property of this covert channel is that it can transmit information between applications running in two different cores because the temperature in one core (which runs the Trojan), affects the temperature of the cores close to it (which may be running the spy). Their channel implementation achieves a throughput of up to 12.5 bps. Their result was improved to 50 bps by Bartolini et al. [1], and latter by Long et al. [14] to 160 bps with close to 0% error rate, and 600 bps with around 15% error rate.

In 2018, Miedl and Thiele [20] presented the first construction of a covert channel based on the processor's power consumption. Their channel implementation is completely based on the CPU power consumption. They considered two platforms for their experiments: a Lenovo notebook with an Intel Core i7 quad-core processor, and server rack based on an Intel Xeon octa-core. Surprisingly, for their channel implementation, the power covert channel showed a lower error rate when running in the notebook than in the server. Using the notebook, the channel achieved 1000 bps with an error rate of a little less than 15%. Miedl and Thiele [20] did not consider the RAPL registers for the DRAM power when building the covert channels.

3 Threat model and experimental setup

3.1 Threat model

Our threat model is very similar to the one considered by Miedl and Thiele [20] when building covert channels based on RAPL CPU power measurements. The spy process can read the RAPL power consumption estimates on the PP0 and DRAM domains. In Linux, the spy can use the `powercap` module to read the power consumption provided by the RAPL registers. This does not require any privileged permission.

The spy runs in two phases. First it records the power consumption in a certain time interval, and then it analyses the data to decode the message. The Trojan is a simple, possibly multi-thread, C program that uses simple memory operations, and does not have access to the RAPL measurements. It does not explicitly communicate with the spy, neither has access to the Internet.

3.2 Experimental setup

In the following sections, we describe experiments and results which are highly dependent on the architecture of our experimental environments. We tested the covert channel in two environments, which are described next.

Notebook setup The notebook considered is an HP Pavilion 14-v064br. This notebook has 8GB of RAM and an Intel Haswell i5-4210U CPU with base frequency of 1.70GHz. This CPU has two cores and supports 4 threads. The sizes of the caches L1, L2, and L3, are, respectively, 32K, 256K, and 3MB.

Desktop setup This environment consists of a 16GB of RAM computer provided with an Intel Coffee Lake i7-8700 CPU with base frequency of 3.20GHz. This CPU has two 6 and supports 12 threads. The sizes of the caches L1, L2, and L3, are, respectively, 32K, 256K, and 12MB.

RAPL offers power measurements on four domains: Package, PP0, PP1, and DRAM. The Package domain corresponds the CPU package, which contains the cores, cache, memory controller and possibly processor graphics. PP0 (Power Plane 0) corresponds to the power consumption of the cores, and PP1 (Power Plane 1) contains the processor graphics component consumption. Not surprisingly, the DRAM domain measures the DRAM power consumption. The RAPL registers are updated at a frequency of about 1000Hz [10]. Figure 2 illustrates the RAPL domains for the notebook setup.

4 Power consumption profiles

In this section, we perform some experiments to get a grasp on how different processes affect power consumption. For now on, we are interested only in PP0 and DRAM domains.

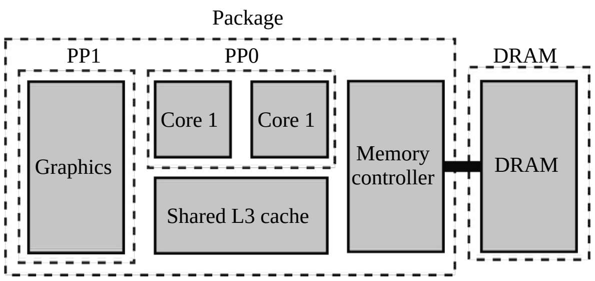

Fig. 2. The four domains for which one can measure power consumption using the RAPL interface on an Intel Haswell i5 4210U, which is our notebook's CPU.

4.1 Modulating CPU and DRAM power

To analyze the power consumption profile of CPU-bound tasks, we ran a simple loop computing a trigonometric function with different number of threads. The result can be seen in the upper part of Figure 3. We can see 4 distinguishable power consumption profiles (excluding the idle power consumption), corresponding to using 1 to 4 threads. The power consumption using 4 and 5 threads are somewhat indistinguishable, which is no surprise since our machine only supports 4 hardware hyperthreads.

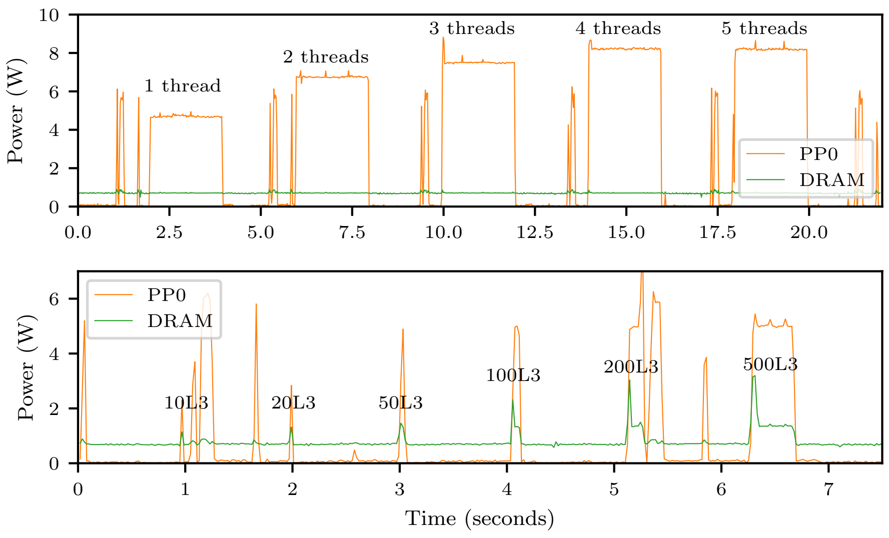

Fig. 3. Power consumption profiles of CPU-bound tasks with different number of threads (top) and of the `memset` operation when setting different sizes of memory chunks, with respect to L3 cache's size, considering the notebook setup.

For the power consumption profile of memory-bound tasks, we measured the DRAM power consumption when running `memset` to set different sizes of memory chunks. We chose to use `memset` in this work because, among other memory operations like `memcpy` and `memchr`, its impact on the DRAM power consumption was the easiest to control in our tests. The bottom part of Figure 3 shows our results. We can clearly see that power consumption and time to complete increase together with the size of the memory chunk used.

4.2 Benchmarks' profiles

When evaluating the covert channel with respect to the robustness against application interference, we consider three benchmarks: MRBench [11], Terasort [22], and LINPACK [6]. The parameters used can be found at https://www.ime.usp.br/~tpaiva/.

Figure 4 shows the power consumption of the first 30 seconds of each benchmark. We can see that MRBench and Terasort affect power consumption in a similar way, although MRBench appears to stress the DRAM power slightly more. LINPACK has a very interesting profile. In the first 5 seconds it uses all hardware threads consistently, affecting the CPU power, but without affecting much the DRAM power. After this period, it starts stressing the DRAM power, and the CPU power also increases by a considerable margin.

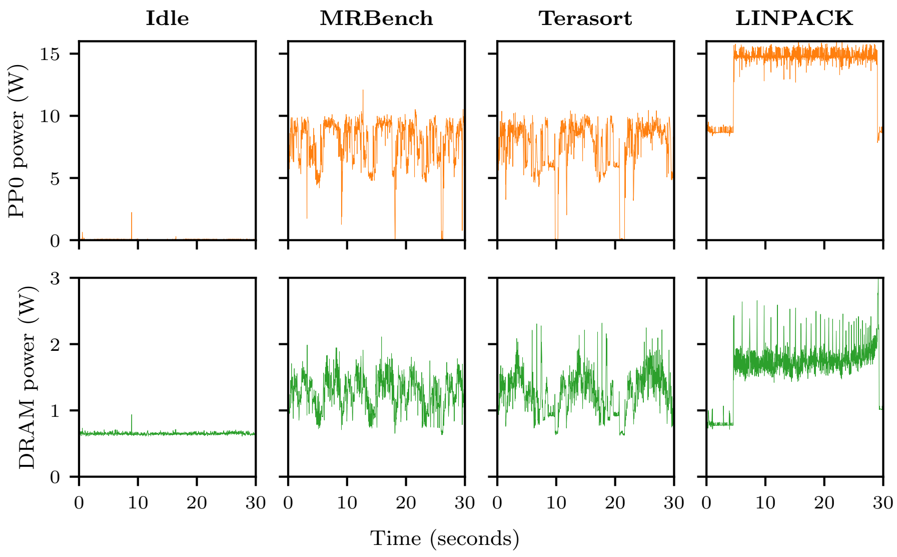

Fig. 4. Power consumption profiles of the benchmarks considering the notebook setup.

It is important to notice that DRAM power usage appears to imply CPU power usage, but not the opposite. This makes DRAM power, in theory, less prone to noise, making it useful for building robust covert channels.

5 Building the covert channels

This section describes the algorithms for building the covert channels based on power consumption. The algorithms do not depend on whether the channel is based on the power consumption by the CPU or by the DRAM. We begin by describing the behavior of the Trojan process, that modulates the power consumption. Then we discuss how a spy process can decode the message using the power measurement samples. We finish this section with a brief discussion on strategies to synchronize the two processes.

5.1 Trojan: modulating power consumption

Suppose we want to encode a binary message $M = (m_1, \ldots, m_n)$. Let A be the CPU or DRAM intensive task we want to use to encode the message. Suppose we want to transmit r bits per second. Let t_s be the time when the transmission is initiated. The algorithm for transmitting the message is given next.

1. Set $i \leftarrow 1$.
2. If $m_i = 1$, run the intensive task A, keeping control of the current time t_c, while
$$t_c - t_s < i \times 1/r. \qquad (1)$$
3. If $m_i = 0$, wait until $t_c - t_s < i \times 1/r$
4. Do $i \leftarrow i + 1$.
5. Go to step 2 if $i \leq n$, otherwise finish.

One of the critical steps in the modulating algorithm is given by Equation 1, which ensures that the transmission is always synchronized with the time when the transmission was initiated. This makes the decoding step easier.

Figure 5 shows the transmission of the same 20-bit message using CPU and DRAM-based Trojans, transmitting at 100 bps. One important observation with respect to the CPU power modulation shown in Figure 5 is that the processor needs some time to go from high to low power consumption. This behavior can be seen in the transmission of the 3rd, 8th, and 16th bits. Therefore when implementing the covert channel, one can consider to make the CPU-intensive task stop earlier when going from a 1 to a 0 bit to lower the probability of a decoding error.

5.2 Spy: demodulating power consumption

Let $T = (t_1, \ldots, t_N)$, be a sequence of time points where a power measurement occurs, and $P = (p_1, \ldots, p_N)$ be the sequence of corresponding observed values.

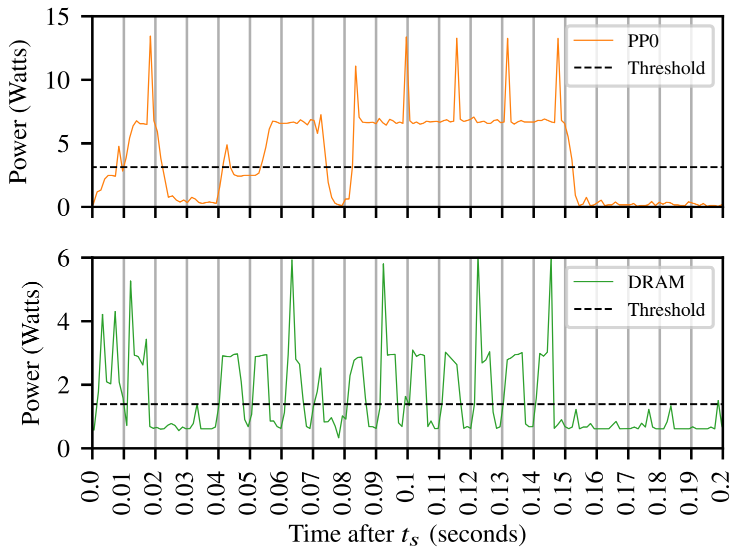

Fig. 5. Example of two Trojans, one using CPU-based modulation (top) and the other using memory-based modulation (bottom), transmitting at 100 bps.

That is, the i-th measurement was made at time t_i and the observed power consumption was p_i. Suppose that t_s, the time when the message starts being transmitted, and the number of bits to be transmitted n, are known. In Section 5.4, we discuss briefly how t_s and n can be obtained by the spy process. As in the previous section, let r be the known transmission rate, that is, the number of bits per second that the Trojan process is programmed to transmit. Then the spy process runs the following algorithm.

1. Set $i \leftarrow 1$.
2. Let
$$S_i \leftarrow \left\{ j \in \{1, \ldots, N\} : \frac{(i-1)}{r} \leq t_j - t_s < \frac{i}{r} \right\},$$
that is, S_i contains the indexes of the time measurements when, ideally, the power measurements samples correspond to the transmission of the i-th message bit.
3. Let
$$\overline{P_i} \leftarrow \frac{1}{|S_i|} \sum_{j \in S_i} p_j,$$
that is, $\overline{P_i}$ is the average of the power measurements that correspond to the i-th message bit.
4. If $\overline{P_i} \geq \overline{P}$, where $\overline{P}$ is some precomputed threshold, then set $\hat{m}_i \leftarrow 1$.
5. Else, set $\hat{m}_i \leftarrow 0$.

6. Do $i \leftarrow i + 1$.
7. If $i \leq n$, go to step 2. If not, return the decoded message $\hat{M} = (\hat{m}_1, \ldots, \hat{m}_n)$.

It is important to note that this algorithm stays valid even if n is not defined. This could be the case when the Trojan never stops streaming the user's private data. Figure 5 can help us visualize the decoding process, considering $t_s = 0, r = 100, N = 20$, and message $M = (11001110111111100000)$. The thresholds $\overline{P}$ shown in the figure depend on the type of Trojan used to encode the message.

5.3 Computing the threshold $\overline{P}$

A simple method for computing the threshold $\overline{P}$ is to run simulations on the target system to learn the distributions of the m_i's' when given P_i's. If the attacker does not have access to the target system beforehand to run the simulations, but he knows that each element of the message is independent and equally distributed over 0 and 1, he can compute the threshold $\overline{P}$ as the average of a large number of P_i's. If the attacker does not have access to the target system, nor does the message bits are independent and equally distributed, the Trojan and spy can use a protocol which starts by sending an encoding of a known message so that the spy can learn the threshold $\overline{P}$ specific for the target system.

5.4 Synchronizing the Trojan and the spy processes

It is important, for the decoding algorithm to work correctly, that the spy knows at what time the message starts being transmitted, denoted as t_s. If the Trojan and the spy share a clock, the instant t_s can be hardcoded in them. But in a more interesting setting, where, for example, the Trojan and spy are running in different Virtual Machines, the assumption of a shared clock is strong.

Let M_{sync} be a binary message of length n_{sync} hardcoded in both Trojan and spy. The Trojan and the spy can use the following protocol for synchronization.

Trojan:

1. To transmit a message M, the Trojan simply transmits (M_{sync}, M) using the encoding algorithm from Section 5.1.

Spy:

1. Read the power consumption registers continually at sample rate T.
2. For each time sample t, try to decode the first n_{sync} bits using $t_s = t$, and compare the result with M_{sync}, which is known a priori.
3. If the number of errors is above some threshold (e.g. 15%), discard t and try the next time sample.
4. Else, use $t_s = t$ to decode (M_{sync}, M) and obtain the message M.

To lower the error rates, when the spy finds a good value in step 4, it can try time samples close to t to find the one which gives the lower error rate with respect to M_{sync}. We call this variant the maximum likelihood syncing, and it is the one used in the experiments in Section 6.

It is also important, for finite n, for the spy to know the message length n. The simplest solution is to allow only messages of a fixed length, but it may be too restrictive. One solution is to use the first $\lceil \log n_{\max} \rceil$ bits to represent the length of the message that will be transmitted, where $n_{\max}$ is the maximum message length allowed.

6 Evaluation of the covert channels

We now evaluate the performance of our covert channels with respect to the transmission rate and robustness against application interference.

6.1 Bit error rate

In Section 4, we show that it is possible to distinguish between power consumption profiles of the `memset` when the size of the memory chunks use are sufficiently different. But, to implement a binary channel, we only need to distinguish between two DRAM power consumption states: doing nothing and `memset` some sufficiently large memory chunk. The size of this memory chunk must be small enough to enable high rate communication, but also large enough to be distinguishable from the baseline DRAM power consumption.

To find good sizes for the memory chunk, we ran Trojans simulations using memory chunks of different sizes considering the notebook and desktop setups. For the notebook, we considered 2, 5, and 10 times its L3 size. For the desktop, we considered two sequential `memset` operations on memory chunks of sizes 0.8, 1, and 1.5 times its L3 cache size.

The simulations consisted in a Trojan process transmitting 10 different 1000 bits random messages at increasing rates. The spy used the first 100 bits for synchronization using the maximum likelihood algorithm discussed in Section 5.4., and then decoded the remaining bits. The decoding threshold $\overline{P}$ is dynamically computed as the average of the power samples.[3] The first column of Figure 6 (Baseline) shows the results of our experiments.

We can see that the larger memory chunks are useful when the transmission rate is lower, because it makes it easier to distinguish between 0's and 1's. However, for higher transmission rates, larger sizes do not work very well because operations on them are slow, which cause synchronization problems.

This simple channel implementation is capable of achieving high transmission rates, without big differences between the two setups. For example, it can transmit at 400 bps with almost 0% error rate, and at 600 bps with less than 10% error rate.

[3] Notice that this works because the binary messages are random, thus we expected them to have a similar number of 0's and 1's.

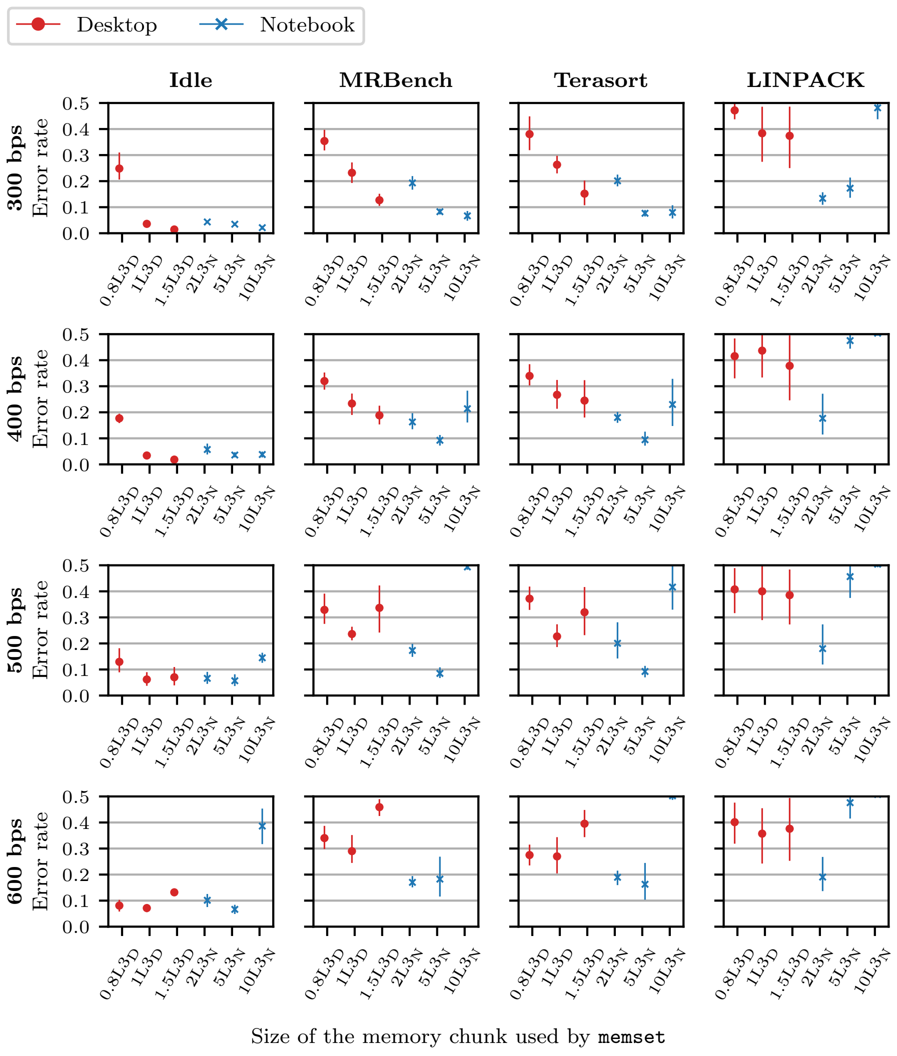

Fig. 6. Error rates for DRAM-based Trojans transmitting at increasing rates when the underlying system is under different workloads. $L3_D$ and $L3_N$ denote the L3 cache sizes of the desktop and notebook platforms, respectively.

6.2 Robustness of the channels

To assess the robustness of the covert channels with respect to application interference, we considered the channels' performance under 3 different workloads: MRBench [11], Terasort [22], and LINPACK [6]. For each workload, we tested the DRAM-based Trojans using the parameters considered in the previous section.

The results are shown in the corresponding columns of Figure 6. Under the desktop setup, the channel tolerates MRBench and Terasort with around 15% error rate at 300 bps, but LINPACK makes the channel very unreliable. With respect to the notebook setup, the DRAM-based Trojan appears to be fairly robust, achieving 500 bps at error rates close to 10% under MRBench and Terasort, and 300 bps under the LINPACK workload with less than 15% error.

It is interesting to see that LINPACK is the one to affect the channel the most, which is not surprising given its power consumption profile presented in Section 4.2.

7 Achieving higher transmission rates

7.1 Using better decoding algorithms

Since the maximum RAPL sample rate is currently fixed at $T = 1000$ samples per second, the number of samples per transmission gets lower as the transmission rate increases. Therefore, to achieve higher transmission rates, we have to extract as much information as we can from the samples. Unfortunately, using only the mean of the samples for each transmitted bit gives us too little information, which prevented us from reliably achieving transmission rates higher than 600 bits in the previous section.

One alternative to the simple decoder we have been using until now is to use clustering algorithms such as Support Vector Machines or Random Forests. We propose to use the whole set of samples corresponding to the transmission of each bit as features. More formally, let σ be a small integer, T be the RAPL sample rate, r be the Trojan's transmission rate, and c_i be the center of the time interval dedicated to the transmission of the i-th message bit. Then to decode the i-th message bit, the spy uses the σ power samples taken at time points which are the closest to c_i. In our experiments, we observed that $\sigma = \max(\lceil T/r \rceil, 4)$ is a good choice. With this setup, since we always feed at least 4 power samples to the classifier, it can perform much better at rates such as 1000 bits per second, where using $\lceil T/r \rceil$ would yield only 1 power sample.

The first row of Figure 7 (1 bit per symbol)[4] shows the error rates for the transmission of binary messages using DRAM based Trojans, under different workloads, when the spy uses the Random Forests classifier. Again, in our experiment, we ran 10 independent transmissions of a message of 1000 bits for each transmission rate and workload. To train the classifier, we used an initial message of 5000 bits with the correct labels. We consider that the synchronization step is already done, that is, the value of t_s is known a priori.

We can see that by using Random Forests we can achieve much higher transmission rates. Under both setups, the Trojan achieved 800 bps with error rates lower than 10%. When running on the notebook, the Trojan even achieved 1000 bps with around 10% error rate. Again, the channel is more robust when running in the notebook platform, where it is able to transmit at 600 bps under

[4] That is, the binary case.

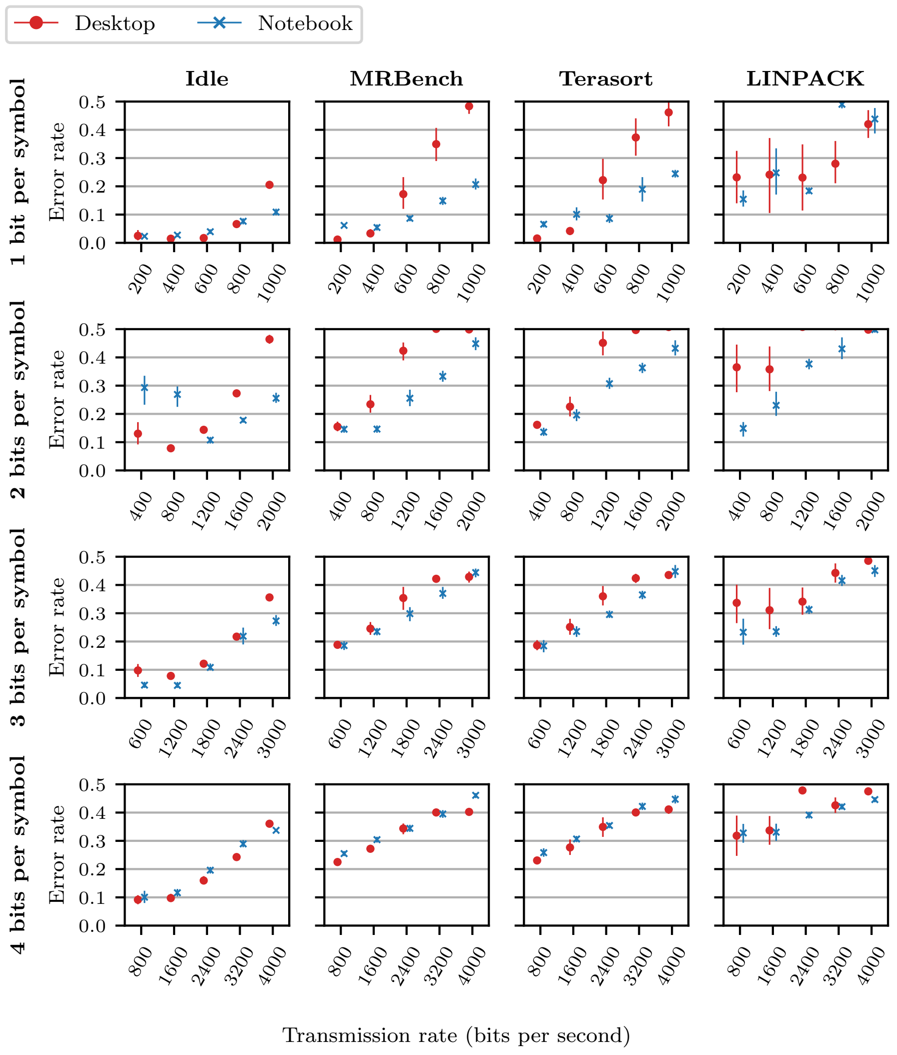

Fig. 7. Error rates for different transmission rates and workloads when transmitting 2, 3, and 4 bits at a time using DRAM-based Trojans combined with CPU-based power consumption states.

MRBench and Terasort with 10% error rate. This makes the Random Forests decoder a better choice than our previous decoding algorithm, at the expense of a more complex spy process, and the need of a possibly large initial predefined message to train the classifier.

7.2 Encoding multiple bits

In this section we investigate the possibility of transmitting non-binary symbols per power modulation. Our main observation is that, from DRAM power consumption samples, it is possible to distinguish, although with some error, the size of the memory chunk set by `memset`.

Figure 8 illustrates four memory power consumption states representing symbols a, b, c, and d. Under the notebook setup, these symbols correspond to the `memset` operation on memory chunks of size 0, 1.5, 3, and 6 times the platform's L3 size, respectively. Similarly, under the desktop setup, they correspond to two sequential `memset` operations using 0, 0.8, 1.2, and 1.4 times the corresponding L3 size. Notice that, if we want to transmit at high rates, we cannot use `memset` over too large chunks of memory because the operation cannot take more time than the inverse of the transmission rate, otherwise we get synchronization problems.

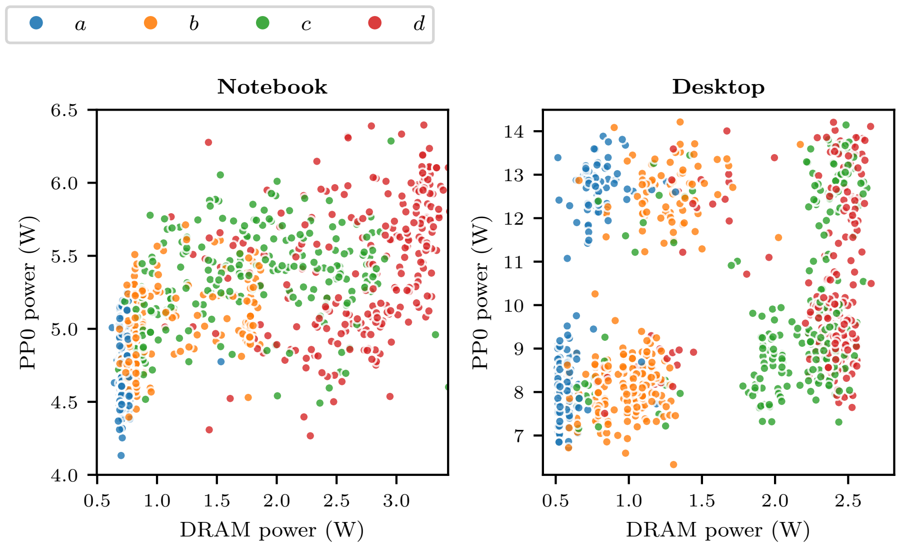

Fig. 8. Power consumption states representing 4 different symbols (2 bits), considering a random message of 1000 symbols transmitted at 200 symbols per second.

Recall Figure 3, where we can see distinct power consumption profiles corresponding to different operations. One straightforward algorithm to encode 4 bits at once, that is, 4 bits per symbol, when modulating one symbol message is to use 4 threads: 1 thread is responsible for modulating the DRAM power consumption, and the other 3 modulate only the PP0 power consumption. The behavior of the encoding algorithm is shown in Table 1. For example, to transmit the bits 1010, one thread does a `memset` operation on a memory chunk 3 times the size of L3 cache, and 2 threads perform a CPU-bound task for a predefined time interval. Notice that in a system with more cores, we expect the number of symbols we can distinguish to be larger, since it has more power consumption states.

Table 1. Encoding 4 bits per transmitted symbol combining CPU and DRAM power consumption states under the notebook setup. For the desktop, we used 0.8, 1.2, and 1.4 times L3, correspondingly.

		DRAM power states			
		Do nothing	`memset` 1.5L3	`memset` 3L3	`memset` 6L3
CPU power states	Do nothing	0000	0001	0010	0011
	1 thread	0100	0101	0110	0111
	2 threads	1000	1001	1010	1011
	3 threads	1100	1101	1110	1111

Figure 7 shows the error rates observed for the DRAM modulation combined with the CPU modulation for different workloads. Notice that for 2 bits per symbol, there is no need for a CPU power modulation thread, since there are 4 distinguishable memory power consumption states. We can see that, under the desktop setup, it was possible to transmit at 2400 bps using 4 bits per symbol at error rates close to 15%. Furthermore, under both setups, it is possible to transmit at 1800 bits per second with error rates around 10%.

It is important to note that the robustness of the channel is better when using 1 and 2 bits per symbol, since there is no need for CPU-power modulation threads, which are less robust with respect to application interference. For more than 2 bits per symbol, the robustness of the channel under both setups is very similar.

8 Discussion

Using the DRAM power modulation alone enables us to build a covert channel implementation with the following very desirable properties.

1. It can can transmit at high rates with low error rates, as defined by TC-SEC [23].
2. It uses simple algorithms for encoding and decoding.
3. It is robust against application interference.

One important feature of DRAM power is that, by manipulating memory chunks of different sizes, we get different power consumption profiles. This can yield a potentially large number of distinguishable states. Further study using other classifiers can help us estimate the number of useful states.

Combining the DRAM and CPU power states, we achieved the throughput of 1800 bps with 10% error, and 2400 with 15% error under the notebook and desktop setups, respectively. This significantly improves upon Miedl and Thiele's results [20], in which only CPU power modulation is considered, as shown in Table 2.

Table 2. Comparison between our covert channel and the one by Miedl and Thiele [20].

Miedl and Thiele [20]			**Our work**		
Processor	Transfer rate	Error rate	Processor	Transfer rate	Error rate
Intel Xeon E5-2690 (octa-core)	200 bps	≈15%	Intel Core i7-8700 (hexa-core)	2400 bps	≈15%
Intel Core i7-4710MQ (quad-core)	1000 bps	≈15%	Intel Core i5-4210U (dual-core)	1800 bps	≈10%

It is important to emphasize that both our platforms have lower numbers of cores than the corresponding ones used by Miedl and Thiele's. It is reasonable to expect the throughput to be even higher in a machine with more cores, since it has more power states. However our results suggest that in a system with more cores, the covert channel robustness against application interference is significantly compromised, in particular under workloads similar to LINPACK.

Our implementation also improves upon Miedl and Thiele's work with respect to the robustness against application interference, mainly because the way `memset` with large memory chunks affects the DRAM power profile appears to be very different than other common applications. Furthermore, understanding which benchmarks affect the quality of this covert channel can shed some light on countermeasures against side-channel attacks.

The data and source code are publicly available at https://www.ime.usp.br/~tpaiva/.

9 Conclusion and future work

In this paper, we introduce a new method for building covert channels which uses the DRAM power consumption estimates given by the processor's internal power measuring registers. We evaluate in detail the performance of the covert channel with respect to throughput, error rate, and robustness against application interference.

We show how to combine DRAM and CPU power consumption states to achieve higher transmission rates. Using this method, our channel implementation improves upon similar proposals, with respect to bandwidth and robustness against application interference. Using more complex encoding and decoding techniques, higher transmission rates with lower error rates may be achieved.

A simple countermeasure to these covert channels is to prohibit applications from reading the power consumption, but this may be to restrictive. Another possible solution would be to allow applications to read the consumption only at low rates. However, further research is needed to find which sampling rates are secure but also give useful information to applications.

For future work, it is interesting to derive a capacity bound for the DRAM-based covert channel as Miedl and Thiele [20] consider for the CPU-based one. It would also be interesting to consider error correction codes to lower the error rates. Another possible extension of this work would be to compare the bandwidth and error rates of the channel when running in different environments.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9. Dr. Navaridas is supported by EuroEXA, funded by the European Union's Horizon 2020 programme under Grant Agreement 754337. Part of this paper was written while T. B. Paiva was attending the Escuela de Ciencias Informáticas 2018 supported by a travel grant given by the Computer Science Department of Universidad de Buenos Aires.

References

1. Bartolini, D.B., Miedl, P., Thiele, L.: On the capacity of thermal covert channels in multicores. In: Proceedings of the Eleventh European Conference on Computer Systems. p. 24. ACM (2016)
2. Bedard, D., Fowler, R., Linn, M., Porterfield, A.: Powermon 2: Fine-grained, integrated power measurement. Renaissance Computing Institute, Tech. Rep. TR-09-04 (2009)
3. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 292–302. Springer (1999)
4. Deshotels, L.: Inaudible sound as a covert channel in mobile devices. In: WOOT (2014)
5. Desrochers, S., Paradis, C., Weaver, V.M.: A validation of DRAM RAPL power measurements. In: Proceedings of the Second International Symposium on Memory Systems. pp. 455–470. ACM (2016)

6. Dongarra, J.J.: Performance of various computers using standard linear equations software (2018), http://www.netlib.org/benchmark/performance.ps
7. Evtyushkin, D., Ponomarev, D., Abu-Ghazaleh, N.: Covert channels through branch predictors: a feasibility study. In: Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy. p. 5. ACM (2015)
8. Gruss, D., Maurice, C., Wagner, K., Mangard, S.: FLUSH+FLUSH: A fast and stealthy cache attack. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 279–299. Springer (2016)
9. Guri, M., Monitz, M., Mirski, Y., Elovici, Y.: Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations. In: 2015 IEEE 28th Computer Security Foundations Symposium. pp. 276–289 (July 2015). https://doi.org/10.1109/CSF.2015.26
10. Intel: Intel® 64 and IA-32 architectures software developer's manual. Intel (2019), https://software.intel.com/sites/default/files/managed/22/0d/335592-sdm-vol-4.pdf
11. Kim, K., Jeon, K., Han, H., Kim, S.g., Jung, H., Yeom, H.Y.: MrBench: A benchmark for MapReduce framework. In: Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on. pp. 11–18. IEEE (2008)
12. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference. pp. 388–397. Springer (1999)
13. Lampson, B.W.: A note on the confinement problem. Communications of the ACM **16**(10), 613–615 (1973)
14. Long, Z., Wang, X., Jiang, Y., Cui, G., Zhang, L., Mak, T.: Improving the efficiency of thermal covert channels in multi-/many-core systems. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1459–1464. IEEE (2018)
15. Mangard, S.: A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: International Conference on Information Security and Cryptology. pp. 343–358. Springer (2002)
16. Mantel, H., Schickel, J., Weber, A., Weber, F.: How Secure Is Green IT? The Case of Software-Based Energy Side Channels. In: European Symposium on Research in Computer Security. pp. 218–239. Springer (2018)
17. Masti, R.J., Rai, D., Ranganathan, A., Müller, C., Thiele, L., Capkun, S.: Thermal covert channels on multi-core platforms. In: USENIX Security Symposium. pp. 865–880 (2015)
18. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 144–157. Springer (1999)
19. Michalevsky, Y., Schulman, A., Veerapandian, G.A., Boneh, D., Nakibly, G.: Powerspy: Location tracking using mobile device power analysis. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 785–800 (2015)
20. Miedl, P., Thiele, L.: The security risks of power measurements in multicores. In: 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018). pp. 1585–1592. ETH Zurich (2018)
21. Murdoch, S.J.: Hot or not: Revealing hidden services by their clock skew. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 27–36. ACM (2006)
22. O'Malley, O.: Terabyte sort on apache Hadoop. Yahoo, available online at: http://sortbenchmark.org/Yahoo-Hadoop.pdf, (May) pp. 1–3 (2008)
23. United States Department of Defense: Trusted Computer System Evaluation Criteria (Orange Book). Tech. rep., National Computer Security Center (1985)

24. Weaver, V.M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S.: Measuring energy and power with PAPI. In: Parallel Processing Workshops (ICPPW), 2012 41st International Conference on. pp. 262–268. IEEE (2012)
25. Yan, L., Guo, Y., Chen, X., Mei, H.: A study on power side channels on mobile devices. In: Proceedings of the 7th Asia-Pacific Symposium on Internetware. pp. 30–38. ACM (2015)
26. Yao, F., Doroslovacki, M., Venkataramani, G.: Are coherence protocol states vulnerable to information leakage? In: High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on. pp. 168–179. IEEE (2018)
27. Yarom, Y., Falkner, K.: FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 719–732 (2014)
28. Zander, S., Branch, P., Armitage, G.: Capacity of temperature-based covert channels. IEEE Communications Letters **15**(1), 82–84 (2011)