ECE 5770 : Resilient Computer Systems Modeling On-Die Termination Power Overhead of Encrypted Data

Udit Gupta, Monica J. Lin

School of Electrical and Computer Engineering, Cornell University, Ithaca, NY

ug28@cornell.edu, mjl256@cornell.edu

Abstract

[0.5pgs]

1. Introduction

Since the advent of the Internet, computing devices have become not only more prolific but also more varied - encompassing not only high performance clusters but also general purpose computers and mobile systems. Moreover, the assumptions made about the security model of a system have changed drastically; users can no longer be trusted, and devices have the potential to interact with unknown and possibly untrustworthy hosts while communicating sensitive information. For example, the boom in smartphones has caused a secondary explosion in the mobile application industry, allowing users to log in to various trusted systems (e.g. bank accounts, medical records and shopping accounts) remotely. The sudden increase in connectivity exposes users to adversaries that may attempt to leverage these interactions in order to obtain confidential information or disrupt the use of services or applications.

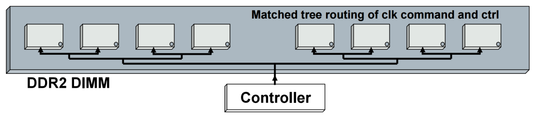
In response to these changes, there has been an increasing effort in developing trusted computing platforms augmented with specialized hardware modules that provide security features such as authentication and decryption/encryption. One such security feature that remains a focus in trusted computing research is protecting off-chip memory. One key component of protecting off-chip memory is maintaining confidentiality of private data. Challenges in designing memory protection mechanisms involves providing encryption primitives at a low cost (money, area, power and design complexity), high throughput and low latency without compromising security. Current work also focuses on memory protection schemes for general purpose and high performance computing systems. The domain of memory protection in the mobile and embedded computing domains is relatively less studied and poses interesting challenges. Mobile and embedded devices often operate under strict power and area constraints. Providing secure memory protection while following the prescribed power and area constraints as well as maintaining performance of the computing devices is an ongoing challenge.

According to Hollis [4] there are two main source of power consumption when driving I/O:

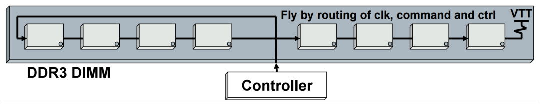
Dynamic A/C Power Dynamic A/C power consumption from charging and discharging capacitances (e.g. electrostatic discharge devices, driver output capacitance, receiver input capacitance and the channel itself), is possibly the dominant and most universal form of I/O power consumption. However, once the capacitances have reached steady state, the dynamic a/c power consumption is negligible. For the purposes of our study we assume that all device capacitances are already in steady state - simplifying our model as we no longer have to consider dynamic a/c power consumption.

Link Termination The second most common source of I/O power consumption is link termination schemes. Intuitively, when the driver outputs a signal which is then received by the I/O device, a significant portion of the signal is reflected back to the driver itself. Consequently, the driver has to not only power the desired signal but also overcome the signal reflections. This incurs a significant power overhead. Past research in the context of link termination schemes has focused on developing more efficient DDR chip topologies to reduce signal reflections. For example, in Figure 1 illustrates the DDR2 and DDR3 DDR chip topologies. As driving frequencies for DDR3 increased, memory architects adopted the fly-by network topology which greatly reduces signal reflections and improves On-Die Termination (ODT).

Though existing DDR3 memory technologies implement more efficient DRAM chip topologies to reduce signal reflections, ODT is still a prominent source of power consumption. Memory architects are looking at more efficient read/write schemes such as Data Bus Inversion (DBI) for further improving ODT power consumption for DDR4. Our study focuses on using first order models for DBI developed by Hollis [4] in the context of memory encryption to simu-



(a) Example DDR2 DRAM chip topology



(b) Example DDR3 DRAM chip topology

Figure 1: Representative DRAM chip topologies for DDR2 and DDR3 memory technologies taken from [2]

late the power overhead of secure memory encryption. Generally, our results so that memory encryption does indeed incur a significant power overhead and that DBI-DC and AC values for is relatively consistent across MiBench applications.

The rest of this paper is organized as follows: Section 2 outlines our general project goals and outlines the memory encryption model used, Section 3 outlines the high level model and experimental setup for our study, Section 4 analyzes our results and data, Section 5 reviews our group dynamic's and work distribution, Section 6 reviews related work in relation to power consumption of secure memory encryption and Section 7 concludes the paper with final remarks and future work.

2. Problem Formulation

Figure 2 illustrates the general secure memory encryption architecture used in this study. The threat model assumes that the on-chip memory, `data-cache`, is secure and tamper-proof whereas the off-chip memory is not. The `encryption core` between the data-cache is supposed to provide cryptographic confidentiality properties that prevent unauthorized principals from reading sensitive information that user's wish to keep secret.

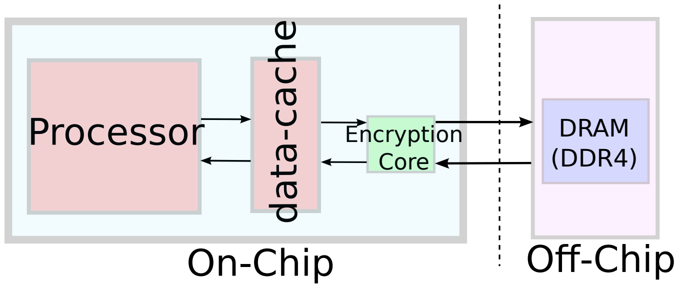


Figure 2: General Memory Encryption Architecture: All on-chip memory accesses, between the processor and data-cache, are performed in plaintext whereas all off-chip, between the data-cache and DRAM, are encrypted.

Typically, the encryption core seen in Figure 2 implements the `AES-CTR` mode scheme. Where previous studies have focused on the performance and area overhead of memory encryption, we wish to explore the impact of memory encryption on the power consumption of off-chip memory storage systems. Specifically we want use various computer architecture analysis tools to simulate memory access traces and model the power overhead of encryption on DDR4 memory technology. Our first order model characterizes the `ODT` power overhead when using `Data-Bus Inversion` for realisitic benchmarks targetting mobile systems using DDR4 memory technology.

3. Methodology

As mentioned earlier, DRAM power consumption is dominated by charging/discharging chip capacitances and signal reflections caused by link terminations. Once the capacitors have reached steady state, their power consumption is negligible in comparison to the system as a whole and thus can be neglected from our power models. Instead we focus on link termination and signal reflections. Specifically, DDR4 memory technology leverages `DBI` to reduce the signal reflections and power consumption. Intuitively, DBI aims to reduce the number of bit flips (DBI-AC) and binary signal probabilities (DBI-DC) to improve power consumption of read and write operations. For reducing bit flips, take for example the case where memory location M previously had the value 00000000 that was being overwritten by the value 11111111. Instead of flipping every bit at address M, a DBI-AC enabled memory system would retain the previous state of 00000000 and set the DBI flag that signifies that all the bits are flipped. This would incur the power consumption of 1 bit flip, not the entire data byte being flipped. Since the power consumption of 0 and 1 is not the same, depending on memory technology constraints, DBI-DC aims to reduce the prevalence of the higher power consuming bit. In the case of DDR4, 0 consumes more power and thus DBI aims to reduce the number of 0's and increase the number of 1's [6]. Figure 3 illustrates the impact of DBI-DC on improving signal quality and consequently power consumption of read and write operations. The larger open data eye illustrates that there is less interference between the driver and reflected signals when using DBI-DC and the smaller V_{pp} suggests that the driver has to work less with DBI-DC enabled.

The following sections outline our model, and experimental setup for measuring the ODT power overhead of memory encryption on DBI-enabled DDR4 memory technology.

3.1 Model

In order to model the power overhead of memory encryption on DBI-enabled memory technologies, we use the same model described by Hollis [4], shown below:

$$P_t = A \times P_{dc} + B \times P_{ac}$$

Where P_t is the ODT power consumption, P_{dc} is the intrinsic DC-power consumed and P_{ac} is the intrinsic AC-

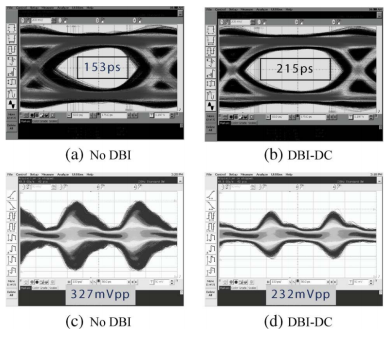


Figure 3: Using DBI-DC increases the open data eye and reduces the required V_{pp} for effectively driving the DRAM chip. Taken from [4]

power consumed by bit flips. The ratios A and B determine the DC and AC factors of P_t respectively. That is, P_{dc} and P_{ac} are properties of the memory technology and A and B are determined by the data being read/written. DBI-DC aims to exploit program structure and reduce A whereas DBI-AC aims to reduce B. Together they can significantly decrease the ODT power consumption. Hollis also describes - under the assumption that data written to memory or read from memory are completely random [4]:

$$A = 0.5$$

$$B = 0.5$$

Intuitively, truly random data would have equal probability of each binary digit making $A = 0.5$. Similarly, truly random data would have equal probability of flipping or not flipping and thus, $B = 0.5$. This will be used later for modeling the AC and DC factors for encrypted data. In order to model ODT power consumption, we must determine the values of A and B for secure memory systems with encryption and insecure memory systems without encryption. The following sections outline our experimental setup for analyzing the DBI enabled power model described.

3.2 Experimental Setup

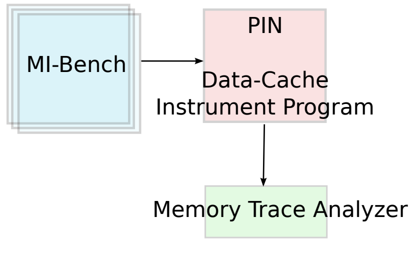


Figure 4: General system architecture for the experimental setup for analyzing the DBI enabled power model

Figure 4 illustrates the general experimental setup we used to implement the DBI model described earlier. First, we need a set of representative benchmarks to generate realistic memory traces that can be used to analyze the effect of memory encryption on power consumption. We chose MiBench [3] for our benchmark suite. MiBench consists of applications representative of embedded and mobile workloads. MiBench applications can be easily compiled to binary executables which integrate well with the `PIN` architectural simulator we used. We chose not to use the `SPEC2006` benchmark as the applications targeted workloads handled traditionally by general purpose processors. We used 28 applications in the MiBench suite (the following 14 applications on big and small data-sets):

1. `basicmath`
2. `bitcount`
3. `qsort`
4. `susan`
5. `jpeg`
6. `dijkstra`
7. `patricia`
8. `stringsearch`
9. `blowfish`
10. `sha`
11. `adpcm`
12. `CRC32`
13. `FFT`
14. `gsm`

The chosen set of MiBench applications spans all six application classes provided by the benchmark: `automotive`, `consumer`, `network`, `office`, `security`, and `telecomm`.

3.2.1 Architectural Simulator

In order to analyze the impact of memory encryption on DBI-enabled DDR4 memory systems, we need to generate memory traces from the MiBench applications using a computer architectural simulator. Traditional simulators such as `Gem5` [1] and `ESESC` [5] did not meet our constraints. Gem5 and ESESC for the purposes of this study required significant overhead in terms of establishing an environment and integrating the simulators with MiBench applications to generate usable memory traces. Instead we chose to use `Pin` [7] - a dynamic binary instrumentation tool developed by Intel as a lean, easy to use computer architectural simulator. Since Pin performs run-time program instrumentation analysis on compiled binary files, integrating Pin with MiBench required little effort. Once we generated the specific `Pintool` to generate memory traces, we could simply run the Pin simulator providing the Pintool and compiled Mibench application binaries as inputs.

Leveraging the IA-32 instruction set architecture framework provided by Pin, we created a `Pintool`, to output the memory trace of arbitrary compiled binaries. That is, we used the `dcache` and `safecopy` examples provided to create a single Pintool that provided a trace of all first-level cache misses. On every miss, the Pintool would output whether the operation was a `load` or `store` operation, the address and the value being read or written to memory. Unlike Gem5 and ESESC, Pin does not provide a cycle-by-cycle timing trace for specified architectural events occur. Though this may be necessary for more accurate system-wide power analysis, our model is not parametrized on time but rather on specific bit values of data being read and written. In our experience, Pin worked well for generating such a coarse, event based memory trace.

For the purposes of our study we only simulated the data-cache with : 32KB total size, 8-way associativity, and 128B cache line size. The Pintool we generated allows the user to input a defined size, associativity and line size. The specific parameters chosen were used in previous Pin exercises as the default configuration and seemed reasonable for our study.

3.2.2 Memory Trace Analysis

Once the memory trace of data-cache misses was generated, the traces needed to be analyzed under our model of ODT power consumption. To the best of our knowledge Pin itself does not offer off-chip DDR memory power consumption tools so we chose to study other simulators that could easily integrate with the Pin memory traces. We focused on using `DRAMSim` [8] however ran into many integration issues. Though DRAMSim takes as input a memory trace, it produced power traces that suggested that for each operation the power consumed was `nan`. In addition, the memory trace that DRAMSim used needed timing information which we could not accurately produce using Pin. Looking at the source code, we could not figure out an easy way to output the power consumed for each operation and decided instead to write our own `Python` based analysis script.

Our Python script reads the memory trace generated by Pin and tracks the sequential load and store misses. For each miss, it counts the total number of 0's and bit-flips assuming DBI is enabled. In order to account for DBI we use the following rules:

DBI-DC If the number of 0's in a given byte is greater than 4, then flip all the bits and set the DBI-DC flag. Using this method, the maximum number of 0's in any given data byte is 4.

DBI-AC If the number of bit-flips between the previous and new data value is greater than half the word length (32-bits in our system), flip all the bits and set the DBI-AC flag. Using this method, the maximum number of bit flips for a given word is 16. For store operations, the previous state is determined by tracking all values previously written to memory. If the store operation is the first to write to the specific memory location then we assume the previous state in memory initialized to a series of 0's. For load operations, we keep track of the previous load operation and compare the new read value to the previous load operation's read value. This assumes that there is a single read bus from the DDR4 to the processor on which we can characterize the DBI-AC factor.

At the end of the memory trace simulation, the Python script calculates the DBI-DC and DBI-AC ratios as follows:

$$DBI - DCRatio = \frac{\texttt{Number of 0's}}{\texttt{Total number of bits}}$$

$$DBI - ACRatio = \frac{\texttt{Number of Bit Flips}}{\texttt{Total number possible of bits}}$$

Finally, we wrote `bash` scripts to fully integrate the simulation flow from the Pin data-cache simulation to the Python script and generating graphs for the produced DBI-DC and DBI-AC ratios.

4. Evaluation

As described in the previous section, our study focuses on analyzing the AC and DC power consumption ratios. Intuitively we would expect that DBI-AC and DBI-DC help the AC and DC power consumption factors respectively. Table 1 illustrates the AC and DC ratio reductions for load and store operations separately. The average AC and DC ratio improvements were computed using the following formula:

$$\texttt{\% Improvement} = \left|\frac{\texttt{DbiRatio} - \texttt{nonDbiRatio}}{\texttt{nonDbiRatio}}\right| \times 100\%$$

DBI Ratio	% Improvement
Load DC	68.59
Load AC	8.62
Store DC	75.04
Store AC	3.38

Table 1: Average AC/DC ratio reduction for load and store operations using DBI across all 28 MiBench applications

We can see that DBI-DC reduces the DC factor by 68.6 % and 75.0 % for load and store operations whereas DBI-AC has a smaller reduction of 8.6 % and 3.4 % respectively. Since the DC factor optimized the data values to reduce the number of binary 0's [6], we can conclude that the values on cache misses for both load and store operations were predominantly 0's in the MiBench suite. This suggests that DBI, especially DBI-DC, does indeed improve the AC and DC power consumption ratios for unencrypted, DBI-enabled memory systems. Next we compare DBI's impact on unencrypted versus encrypted memory systems.

Figure 5 illustrates the DBI-AC and DBI-DC ratios for select MiBench applications and our encrypted system model. The MiBench applications chosen: `basicmath`, `jpeg`, `djikstra`, `fft`, `bitcount`, `blowfish`, and `strsearch` span all six application classes of MiBench: `automotive`, `consumer`, `security`, `office`, `telecomm` and `network`. We chose these specific applications as they span all six application classes of MiBench and are representative of the memory access patterns for each of their domains.

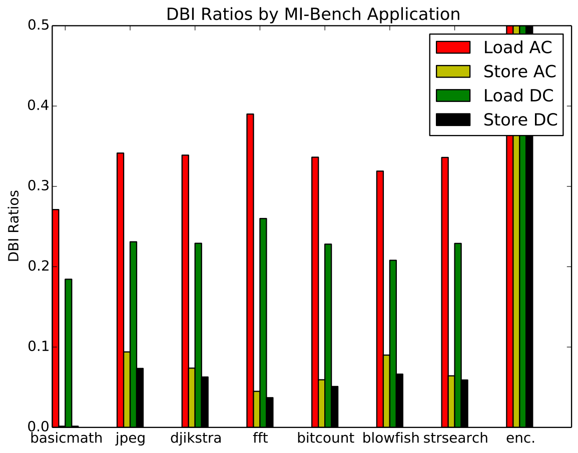


Figure 5: Comparing load/store DBI ratios for representative MiBench applications

Encryption Model First, the encryption model exhibited DBI-AC and DBI-DC ratios of 0.5 for both loads and stores. As mentioned in Section 3, truly random data exhibits DBI-AC and DBI-DC ratios of 0.5 [4] since truly random data has equal signal probabilities of 0 and 1, and equal switching probabilities (flip or not flip). According to the Avalanche effect, changing even a single bit in the unencrypted plain-text should cause the encrypted ciphertext to appear completely random [9]. Under the avalanche effect, we assume encrypted data is truly random for the purposes of this study. Furthermore, as shown in Figure 2, all cache misses are first encrypted with the `Encryption Core` before being processed by the off-chip DRAM based memory system. This means that for the encrypted model, all data seen by the DDR4 DRAM memory system will be truly random and DBI-AC and DBI-DC ratios will be 0.5. Though the true DBI-AC and DBI-DC ratios may be less than 0.5, we believe that the difference would not be significant enough to change the underlying results of this study.

As seen in Figure 5, the DBI-AC and DBI-DC ratios for load and store operations for the MiBench applications is significantly lower than that of encrypted model. This suggests that for unencrypted data stores and loads that follow the MiBench memory access pattern, DBI-AC and DBI-DC can significantly reduce the AC and DC power factors to 0.33 for Load-AC, 0.07 for Store-AC, 0.22 for Load-DC and 0.06 for Store-DC on average across all MiBench applications. If the encryption scheme seen in Figure 2 were used, the DBI-AC and DBI-DC ratios would be 0.5. This suggests that for load and store operations, encryption would incur a significant power overhead.

Interestingly, we see that for all MiBench applications — especially the select ones shown in Figure 5 — the load and store ratios are relatively consistent. This can be explained by the fact that for the given cache architecture, the MiBench applications may have similar miss rates per 1000 instructions [3]. From the select applications, `rijindael` and `ispell`, and the given cache configuration : 32KB size, 8 way associative and 128B cache lines both applications have around 3 misses per 1000 instructions. Assuming that the trend continues for the all applications in the MiBench suite, the DBI ratio similarity across the applications is expected. Furthermore, it should be noted that MiBench is organized such that each application has two version: `small-input` and `large-input`. We noticed that for all applications, the small and large input version exhibited virtually the same DBI ratios which can also be explained by the fact that despite the varying input data sets, the memory access pattern should be the same across both versions resulting in similar DBI ratios.

5. Group Dynamics

Generally, we worked together for all parts of the project. To start with, we came up with the design and overall experiment together. This included investigating Gem5, SESC, and Pin. Since Monica had more experience regarding the computer architectural simulators she was able to better understand the benefits of using Pin versus Gem5 and SESC. Next we split up the work for getting Pin setup, acquiring benchmarks and investigating DRAM simulators. Monica worked on acquiring the benchmarks and finding DRAM simulators - namely DRAMSim. Udit worked on organizing, understanding and implementing a working Pintool for generating the memory trace. Once the initial Pintool for generating a memory trace using the `pinatrace` and `safecopy` examples, (without a data-cache) was completed, we worked together to test the Pintool on small generic `bash` commands and sample programs we wrote.

The two of us then worked together on understanding DRAMSim and understanding how to run the infrastructure using user inputted memory traces. After a few days of analyzing the tool flow, the generated power traces, and the source code the two of us decided that DRAMSim would not be a viable option for our project. At this point we went back to existing literature to find simple DBI models that we could use for power modeling. This is when we found Hollis' model [4] and agreed to use it for our study. Concurrently we worked on implementing a data-cache instrumentation tool using Pin to isolate the cache misses - a more realistic configuration for our project. Finally, once the DBI model was decided, Udit focused on writing the simple Python script to analyze the memory traces and Monica focused on organizing all of our sources, background research and began working on the power-point.

Needless to say, most of the project was done together since much of the work was understanding existing models and tools.

6. Related Work

[0.5pgs]

7. Conclusion

[0.5pgs]

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug 2011.

[2] J. Burnett. DDR3 Design Considerations for PCB Applications. Jul 2009.

[3] R. M. Guthaus, S. J. Ringenberg, D. Ernst, M. T. Austin, T. Mudge, and B. R. Brown. MiBench: A free, commercially representative embedded benchmark suite. *Int'l Symp. on Workload Characterization (IISWC)*, 2001.

[4] T. M. Hollis. Data Bus Inversion in High-Speed Memory Applications. *IEEE Transactions on Circuits and Systems*, 2009.

[5] E. K. Ardestani and J. Renau. ESESC: A Fast Multicore Simulator Using Time-Based Sampling. In *International Symposium on High Performance Computer Architecture*, HPCA'19, 2013.

[6] S.-W. Kwack and K.-D. Kwack. A high speed graphics dram with low power and low noise data bus inversion in 54nm cmos. IEICE Electronics Express, 2009.

[7] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, Jun 2005.

[8] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMSim: A memory-system simulator. 2005.

[9] A. Webster and T. S. On the design of s-boxes. *Advances in Cyptology CRYPTO*, 1986.