Final Projects

EEE5716/EEE4714: Introduction to Hardware Security and Trust

Fall 2024

Deadlines

- December 8, 2024, 11:59 pm (Subject to change): Final project report and other deliverables are due.
- During finals week, TAs may reach out for a project demonstration by appointment, depending on the project and our ability to replicate your results.

Important Points

- If your project requires any additional hardware, your team is responsible for purchasing the hardware and splitting costs.
- Source files (programming scripts, RTL, etc.) must be commented on and cleanly written. They are graded as part of your report.
- The ECE Linux server can provide licensed software tools (Synopsys Design Compiler, IFV, Jasper). Xilinx Vivado is free to download on your personal computer. Refer to vnc_setup_and_tools_access.pdf in Canvas for help accessing the ECE Linux server.
- **Start early.** Except in extenuating circumstances, extensions will not be granted. TAs may help debug tooling problems during office hours, but we will certainly be flooded with such requests close to the project deadlines and unable to help everyone.
- If your project is not working or your results are not as clean as you had hoped, this does not mean you will get a bad grade. In this scenario, focus on writing a detailed, high-quality report on what works. Explain where you suspect problems occurred and how you would try to fix them.

1 Weak PUF (ring oscillator) implementation

Main Objective: In this project, you are exposed to the challenges of implementing reliable, high-quality PUFs.

Background Physical unclonable functions (PUFs) are a security primitive that binds some secret value(s) to a physical piece of silicon. PUFs are supposedly unclonable because their response depends on manufacturing process variations that are believed to be uncontrollable. PUFs are issued a challenge and (ideally) produce a unique and reliable response in return. Since this response is unique to the device, it can be used as a device ID or key. Weak PUFs, such as ring-oscillator PUFs, are often used for generating cryptographic keys.

However, designing and implementing a PUF must be done carefully to avoid skewing the response. For example, implementing a PUF in FPGA fabric requires explicitly specifying which hardware primitives the PUF will occupy. Otherwise, an RTL-based design without any placement/routing constraints allows the synthesis and layout tool to choose an optimized structure that is not typically symmetric and, therefore, will produce a skewed PUF response.

Project Goal: You are expected to design and implement 64-bit RO-PUF architecture and evaluate the quality of the PUF using the same metrics used in hardware security research: reliability, uniqueness, and uniformity/randomness.

Software Requirements

- Xilinx Vivado. Consult freely available online resources to learn how to create constraints in Vivado to map PUF primitives to FPGA hardware. E.g., https://youtu.be/hJ1LaSKYYow
- Scripting language for analyzing/visualizing your results. Python, MATLAB, or others.

Hardware Requirements

- Basys 3 : https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Report to submit.

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
  1. Top Sheet with group members' names and ID
  2. Introduction, motivation, and problem statement. (10%)
  3. RO-PUFs Implementations (40%)
     - PUF architecture and operating principle
     - Hardware mapping
     - Experimental setup
     - Algorithms for post-processing if necessary
  4. Results (35%)
     - Did it work? Why or why not? Discuss challenges faced, how those challenges were overcome, etc.
     - Reliability, uniqueness, and uniformity measurements for the implemented PUF
     - Any tables, plots, etc., you produce when analyzing your PUF.
  5. Conclusion and personal comments (10%)
- Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all the following:
  - Report.pdf
  - Working directory for your analysis
    * readme.txt (provide necessary instructions for running your code)
    * Any RTL source files and TCL scripts used to build the PUF.
    * All raw data collected from the PUF.
    * All scripts used to analyze/plot/tabulate your results.

References

1. M. Majzoobi, A. Kharaya, F. Koushanfar and S. Devadas , “Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines”, 2014.
2. A. Maiti et al., A systematic method to evaluate and compare the performance of physical unclonable functions, In Embedded Systems Design with FPGAs, P. Athanas, D. Pnevmatikatos, and N. Sklavos (Eds.). Springer, New York, 245267.

2 Strong PUF (arbiter) implementation

Main Objective: In this project, you are exposed to the challenges of implementing high-quality PUFs, and graduate students will additionally compare two implementations of delay based strong PUFs.

Background Physical unclonable functions (PUFs) are a security primitive that binds some secret value(s) to a physical piece of silicon. PUFs are supposedly unclonable because their response depends on manufacturing process variations that are believed to be uncontrollable. PUFs are issued a challenge and (ideally) produce a unique and reliable response in return. Since this response is unique to the device, it can therefore be used as a device ID or key. Strong PUFs, such as the delay-based arbiter PUF, are particularly useful for device authentication: if a manufacturer can collect a number of challenge-response pairs for each IC as they are fabricated, they can store these and query devices in the field to verify their authenticity.

FPGA implementation of delay based PUFs must be performed carefully. The primary requirement is that all delay paths must be as carefully matched as possible so that there is no systematic (and predictable) variation in the PUF response due to the path mismatch. This is crucial for FPGA since an RTL-based design without any routing constraints allows the synthesis and layout tool to provide an optimized structure that is typically not symmetric and identical. Hence, for most FPGAs, one must incorporate all possible routing constraints and design and place the hard macro for symmetric design.

Project Goal: You will design a 32-bit arbiter PUF architectures. You will evaluate the quality of your PUF using the same metrics used in hardware security research.

Software Requirements

- Xilinx Vivado. Consult freely available online resources to learn how to create constraints in Vivado to map PUF primitives to FPGA hardware. E.g.: https://youtu.be/hJ1LaSKYYow
- MATLAB, Python, C/C++, R, Octave, etc. for performance analysis.

Hardware Requirements

- Basys 3 : https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Report to submit.

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
  1. Top Sheet with group members' names and ID
  2. Introduction, motivation, and problem statement. (10%)
  3. PUF Implementations (40%)
     - PUF Architecture(s) and operating principle
     - Experimental setup for evaluation
     - Any post-processing algorithms/hardware
     - Hardware mapping
  4. Result and Effectiveness (35%)
     - Did the PUF(s) work? Why or why not? Discuss challenges faced, how those challenges were overcome, etc.
     - Reliability, uniqueness, and uniformity measurements for the implemented PUF(s)
     - Any tables, plots, etc. that you produce when analyzing your PUF.

5. Conclusion and personal comments (10%)
6. Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all of the following:
  - Report.pdf
  - Working directory for your analysis
    * readme.txt (provide necessary instructions for running your code)
    * Any RTL source files and TCL scripts used to build the PUF.
    * All raw data collected from the PUF.
    * All scripts used to analyze/plot/tabulate your results.

References

1. M. Majzoobi, A. Kharaya, F. Koushanfar and S. Devadas, “Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines”, 2014.
2. A. Maiti et al., A systematic method to evaluate and compare the performance of physical unclonable functions, In Embedded Systems Design with FPGAs, P. Athanas, D. Pnevmatikatos, and N. Sklavos (Eds.). Springer, New York, 245267.

3 Machine Learning Attack on Arbiter PUF

Main Objective In this project, you will learn that PUFs are not perfect! In particular, you will perform a machine learning (ML) based attack on arbiter PUFs.

Background Research shows that arbiter PUFs can be modeled with certain machine learning techniques [1], [2]. If adversaries can accurately model a PUF, they can predict its challenge-response pairs and impersonate the PUF, thereby breaking the security of protocols that rely on the response is unpredictable.

In the simplest form, a machine learning-based modeling attack proceeds as follows:

1. The attacker learns some number of challenge-response pairs for the PUF. This could happen if, for example, a manufacturer remotely authenticates its chips on an unencrypted channel.
2. The attacker must know the PUF architecture.
3. A mathematical model is constructed of the PUF, and parameters of the model are derived (or, equivalently, "the model is trained") using known challenge-response pairs.
4. New challenges can now be supplied to the trained model (which now acts as the software clone of the original PUF) to predict the real PUF's responses.

More efficient modeling attacks accurately model a PUF's responses given a small number of challenge-response pairs.

Project Goal

1. For the given CRP dataset (CRPSets.zip), implement two machine learning techniques for predicting PUF responses given previously unseen inputs (SVM: [1,3], regression: [2]). In other words, train your model using the different training/testing splits as described in "Results to submit".
2. Evaluate your model. Use CRPs that were not in your training set to determine how well your model predicts the PUF output for inputs that it has not previously seen.

Software Requirements

- Any software and scripting language capable of performing numerical analysis and building machine learning-based trainers and classifiers (MATLAB, R, Python, C/C++).
- CRPSets.zip contains (12000x65) data for a 64-stage arbiter-PUF with 12000 challenges, as rows indicate. Each challenge is 64-bit long (Columns 1-64), and the corresponding response is one bit, 0/1, given in column 65.

Report to submit.

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
    1. Top Sheet with group members' names and ID
    2. Introduction, motivation, and problem statement. (10%)
    3. Prior techniques (5%)
    4. Methods: model selection and design (40%)
        - Implementation of PUF modeling attack
        - Reason of choice, brief description, pros/Cons
        - Working flow-chart/algorithm
        - Used boundary conditions, parameters, etc. as necessary.

5. Results (35%)
   - Evaluate the impact of training set size on the accuracy of your PUF model. What is the smallest number of CRPs you could observe that would allow your model to predict PUF responses with 80% accuracy?
     * 2000, 6000, and 10000 samples (total of 12000 challenges/samples available)
     * All remaining samples are used for evaluation.
   - Compare different attack models (Grad only)
   - Tables, plots, etc., summarizing the accuracy of your modeling attacks.
   - Compare your modeling results with those from [1,2].
6. Conclusion and personal comments (10%)
7. Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all of the following:
  - Report.pdf
  - Working directory for your analysis
    * readme.txt (provide necessary instruction for running your code)
    * All source files.
    * Data set

References

1. Rhrmair, Ulrich, et al. "PUF modeling attacks on simulated and silicon data." IEEE Transactions on Information Forensics and Security 8.11 (2013): 1876-1891.
2. Rhrmair, Ulrich, et al. "Modeling attacks on physical unclonable functions." Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010.
3. Lim, Daihyun. Extracting secret keys from integrated circuits. Diss. Massachusetts Institute of Technology, 2004.

4 Machine Learning-based Classifier for Hardware Trojan Detection

Main Objective In this project, you must design machine learning (ML) based classifiers to detect a hardware Trojan.

Background Detecting malicious modifications ("Trojans") in a fabricated chip is difficult. The designer does not know the size, type, location, trigger, or payload of a potential Trojan, and there is a lot of complex logic on a chip that must be analyzed before the chip can be declared Trojan-free. One class of Trojan detection mechanisms analyzes variations in power consumed by a chip. Since any chip activity consumes power, unexpected activity or variations may indicate the presence of Trojans.

In [1], the authors put identical ring oscillators in different locations of a fabricated chip to monitor the power supply network at runtime. The key idea behind this is that RO frequencies are affected by the power supply and temperature variations (runtime activity), capacitance, threshold voltage, etc. variations (from the manufacturing process). Since wires are not perfect superconductors (they have finite resistance and inductance), any Trojan switching will draw power in one part of the on-chip power network and also cause a small, temporary voltage dip in a different part of the circuit. This voltage drop causes a temporary drop in RO frequency when the Trojan is active. If the resulting drop in RO frequency can be detected, then the Trojan can be detected.

However, the real-life scenario is much more complicated. There is no single frequency that works as a pass/fail threshold for detection: ROs experience random process variations and, therefore, produce slightly different frequencies even with similar design and operating conditions (recall the RO frequency data in homework 2), so it is difficult to compare RO frequencies from different chips even if they are both Trojan-free. Also, normal switching activity can cause RO frequency variations as large as, or even larger than, Trojans. Thus, smart classifiers are required. This is why you will design a machine learning classifier to distinguish trojan-infected from trojan-free chips.

Project Goal

1. Choose two cases from the following and implement them:
    - Case 1. You have some known samples of both Golden chips and Trojan-inserted chips. (i.e., RO data from both types of chips are known), and both can be used for training the classifier. (However, the type of Trojan is unknown.)
    - Case 2. You only have some known samples of Golden chips, i.e., only golden data can be used for training the classifier.
    - Case 3. You have completely unidentified samples (no knowledge about the samples, whether they are golden, Trojan-inserted, or a mixture of both) to train the classifiers.
2. Implement two different classification techniques of your choice.
3. Evaluate your classifier accuracy with the given sample size (33 total samples available):
    - 6 samples (Case 1: 3TF and 3TI; Case 2: All 6TF; Case 3: All 6 Unknown)
    - 12 samples (Case 1: 6TF and 6TI; Case 2: All 12TF; Case 3: All 12 Unknown)
    - 24 samples (Case 1: 12TF and 12TI; Case 2: All 24TF; Case 3: All 24 Unknown)
    - All remaining samples are used for evaluation.

    For each case, choose the samples for the training set randomly (20 trials) and use the remaining chips/samples for evaluation. Run your classification/detection and collect results (20 trials) for each Trojan type (23 total), and report the average accuracy result (i.e., % of successful detection).

Software Requirements

- Scripting language for analyzing/visualizing your results. Python, MATLAB, or others.

- Dataset: ROFreq.zip (on Canvas) contains 33 CSVs (Chip1, Chip2, Chip33) that refer to data from 33 chips. Each spreadsheet has the following specifications:
  - Each row indicates Trojan Free/ Trojan-inserted data.
  - Trojan Free (golden) data in row 1 and row 25
  - Trojan-inserted data in rows 2-24
  - Each column indicates the frequency of RO# in the network (RO1-RO8). For details, check Ref. [1].

Report to submit.

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
  1. Top Sheet with group members names and ID
  2. Introduction, motivation, and problem statement. (10%)
  3. Classifiers selection and design (40%)
     - Reason of choice, Brief description, Pros/Cons
     - Short workflow/algorithm, model architecture, and parameters, etc.
  4. Classifier evaluation Results (35%)
     - For each classifier, provide the following: Working flow-chart/algorithm and used boundary conditions, parameters, etc., as necessary.
     - For each problem case, provide the following:
       * Average true positive and false positive rates over the 20 trials (figures, tables, etc.).
       * Average training time (if appropriate) and evaluation time (if appropriate).
       * Comparative results
         · Among different sample sizes
         · Among different cases (if applicable)
       * You can also provide any other necessary data to justify your implementation. Some sample tables/plots are given for your convenience.
  5. Conclusion and personal comments (10%)
  6. Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all of the following:
  - Report.pdf
  - Working directory for your analysis
    * readme.txt (provide necessary instruction for running your code)
    * All source files
    * Data set

References

1. Shane Kelly, Xuehui Zhang, Mohammed Tehranipoor, and Andrew Ferraiuolo, Detecting Hardware Trojans using On-chip Sensors in an ASIC Design. Journal of Electronic Testing 31, no. 1 (2015): 11-26.

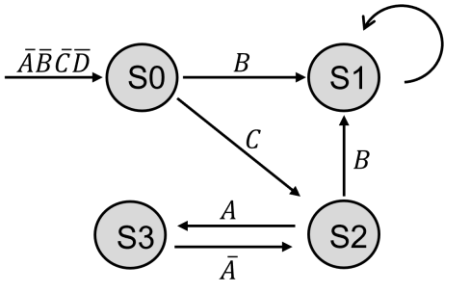
5 Design and Security verification of a Finite State Machine

Main objective The reason that cybersecurity is such an important concern is that *implementing a system correctly is very difficult.* In real hardware and firmware development, companies pay lots of money for engineers who can help verify system correctness. This project gives you a taste of hardware validation and verification.

Background Security requirements can be viewed as a special subset of correctness requirements. In other words, it makes no sense to talk about the security of a hardware IP unless you can first guarantee that the IP performs its function correctly. Moreover, the same tests used to verify that an IP is implemented correctly might be useful for finding trojans.

Functional testing is the simplest way to guarantee that a design implements its specification correctly, but functional testing can only guarantee the design's behavior for a specific (sequence) of inputs. Formal modeling and verification can provide stronger guarantees about the design's behavior when it is supplied with inputs that have not been functionally tested.

Consider the following Finite State Diagram that has 4 inputs and 4 states. Assume that only one input can change per clock cycle. Note that all state transitions are not explicitly indicated; if an input changes that does not cause a state transition, the FSM remains in its current state. You are required to implement the RTL design and perform functional and property-based security verification of this FSM circuit.



Project Goal: Design the FSM in RTL code and verify it using both conventional tests (a "test bench") and formal approaches. Students should attempt to implement as rigorous a test bench as possible. For the formal verification portion, students will implement ten assertions to be checked in Jaspergold. You may find errors or bugs in your initial design. If you do, fix your FSM implementation and re-run the tests.

Software Requirements

- Any Verilog/VHDL/System Verilog simulator (Vivado, Synopsys VCS, Modelsim)
- Cadence JasperGold for formal verification (Available on the ECE server. You can connect to the Linux server by following the tutorial Tutorial_RemoteAccess.pdf uploaded in canvas)

Hardware Requirements None

Report to submit

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
  1. Top Sheet with group members names and ID
  2. Introduction, motivation, and problem statement. (10%)
  3. Methods (40%)
     - Technical description of your design implementation. (20%)

- How did you test your FSM design? Describe the verification methodology. How confident are you that your test bench achieves good "coverage" of your FSM's operation? (20%)
- What properties of the FSM are formally checked? Explain how you chose formal assertions to be checked, and reason about what aspects of your design are verified by your assertions.

4. Result and discussion (35%)
   - Were there bugs in your initial implementation(s)? If so, how did you change your design to resolve bugs? Were your bugs identified via formal analysis or through functional testing?
   - This is a very simple system. Discuss how your testing methodologies would scale to larger systems.
   - What are the trade-offs between formal analysis and functional testing?
5. Conclusion and personal comments (10%)
6. Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all of the following:
  - Report.pdf
  - Working directory for your analysis. **Provide a different "version" of the working directory for each iteration of the analysis.** E.g., if your first implementation has bugs, include all of the following for your first iteration and call it "foldername_v1", then fix the problem and call it "foldername_v2" ...
    * readme.txt (provide necessary instructions for running your code)
    * FSM design implementation source
    * Functional test bench
    * FSM formal properties
    * Bind files and TCL script (to run automatically in Jaspergold)
    * Traces screenshots and VCD dump files of counterexamples

References *References 1 and 2 describe implementation and verification of a FIFO. While they are obviously not verifying the state machine specified above, you may find it helpful to refer to these works to learn about the verification process, generally.*

1. Cummings, Clifford E. "Simulation and synthesis techniques for asynchronous FIFO design." SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers. 2002.
2. Rizvi, N. Z., Arora, R., & Agrawal, N. (2015). Implementation and Verification of Synchronous FIFO using System Verilog Verification Methodology. Journal of Communications Technology, Electronics and Computer Science, 2, 18-23.
3. https://www.doulos.com/knowhow/sysverilog/tutorial/assertions/

6 Side-Channel Attacks on Advanced Encryption Standard (AES)

Main Objective In this project, you will extract the key to an Advanced Encryption Standard (AES) cryptographic algorithm through a side-channel attack using power analysis, with as few as possible collected power traces.

Background In cryptography, a side-channel attack is based on information gained from the physical implementation of a cryptosystem rather than brute force or theoretical weaknesses in the algorithms (in contrast with cryptanalysis). For example, timing information, power consumption, electromagnetic emanations, or even acoustic sound can provide an extra source of leaked information, which can be exploited to break the system. Some side-channel attacks require technical knowledge of the internal operation of the system on which cryptography is implemented. However, others, such as differential power analyses, are effective as black-box attacks. Many powerful side-channel attacks are based on statistical methods pioneered by Paul Kocher.

Power analysis is a form of side-channel attack in which the attacker studies the power consumption of a cryptographic hardware device (e.g., smart card, tamper-resistant "black box", or integrated circuit). The attack can non-invasively extract the device's cryptographic keys and other secret information. Simple power analysis (SPA) involves visually interpreting power traces or graphs of electrical activity over time. Differential and Correlation power analysis (DPA/CPA) are more advanced forms of power analysis that can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations.

Project Goal: The students must mount a DPA/CPA attack on the public power traces provided in the DPA contest v2 (https://www.dpacontest.org/v2/download.php). Students should optimize their algorithm to obtain the key with as few power traces as possible.

Overall Requirements: For this project, you need to implement a DPA/CPA on power traces that were collected from an AES module (https://www.dpacontest.org/v2/data/aes_dpa_contest_v2.tar.gz). You can download these traces from the public database: https://cloud.telecom-paris.fr/s/N5qgyMdxEcqipN2.

1. There are 640,000 AES traces of 32 different keys, as presented in the public database. Each key has 20,000 encryption traces with random plaintexts. Keys and plaintexts can be found in the index file[1].
2. For four keys in the data set, target the corresponding traces using your CPA/DPA attack scripts and report your results.

Keys	082efa98ec4e6c89452821e638d01377
	be5466cf34e90c6cc0ac29b7c97c50dd
	00000000000000003243f6a8885a308d3
	13198a2e03707344a4093822299f31d0

Table 1: Specified keys

3. Randomly select two additional keys and report whether your scripts successfully extract those keys.
4. Attempt to optimize your code to find the key with as few power traces as possible.

Note that you can find useful materials like reference tools and templates to help you build the attack scripts on this page (https://www.dpacontest.org/v2/download.php).

Software Requirements Scripting language for analyzing/visualizing your results. Python, MATLAB, or others.

Hardware Requirements None

[1] https://www.dpacontest.org/v2/data/keymsg_public_base_dpacontest2.bz2

Report to submit

- IEEE conference format. (4-6 pages)
- The report should include (tentative marks dist.):
  1. Top Sheet with group members' names and ID
  2. Introduction, motivation, and problem statement. (10%)
  3. Methods (40%)
     - Describe why you chose either DPA or CPA
     - Describe how your chosen method of power analysis works.
     - Describe how your scripts work.
  4. Results and discussion (35%)
     - Do your scripts work for all of the required keys? How did you tweak your algorithms to work for your assigned keys?
     - How many traces do your scripts need to process before they start to reveal correct key byte guesses?
     - Include sample plots for different guesses at round-key bytes and compare the figures of the correct guess with the wrong guesses. The correct guess should be obviously different from the incorrect guesses.
     - *After* tweaking your algorithms, randomly select (UG:1 and G:2) keys other than your assigned ones and report whether your scripts can successfully deduce the key. Include plots as appropriate.
  5. Conclusion & Personal comments (10%)
  6. Clarity, organization, etc. (5%)

Submission guidelines

- Submit a .zip file with name: Project#Group#
- Include all of the following:
  - Report.pdf
  - Working directory for your analysis
    * readme.txt (provide necessary instructions for running your code)
    * All source files.

References

1. https://www.dpacontest.org/v2/index.php
2. Kocher, P., Jaffe, J., Jun, B., & Rohatgi, P. (2011). Introduction to differential power analysis. Journal of Cryptographic Engineering, 1(1), 5-27.
3. Brier, Eric, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model." International workshop on cryptographic hardware and embedded systems. Springer, Berlin, Heidelberg, 2004.
4. Clavier, Christophe, et al. "Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest." Journal of Cryptographic Engineering 4.4 (2014): 259-274.
5. Lo, O., Buchanan, W. J., & Carson, D. (2017). Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA). Journal of Cyber Security Technology, 1(2), 88-107.
6. Randolph, M., & Diehl, W. (2020). Power side-channel attack analysis: A review of 20 years of study for the layman. Cryptography, 4(2), 15. https://doi.org/10.3390/cryptography4020015.

7. Maghrebi, H., Portigliatti, T., & Prouff, E. (2016, December). Breaking cryptographic implementations using deep learning techniques. In International Conference on Security, Privacy, and Applied Cryptography Engineering (pp. 3-26). Springer, Cham

8. Hnath, William. Differential power analysis side-channel attacks in cryptography. Diss. Worcester Polytechnic Institute, 2010.

7 RowHammer Testing

Main Objective: In this project, you will learn about the Rowhammer vulnerability and test the vulnerability of a real DRAM chip.

Background Computer systems use privilege and permissions to protect system integrity. Without privilege and permission, malicious users could take complete control of the computer and steal data or harm other users. In 2014, it was discovered that the physical weaknesses of modern DRAM (dynamic random-access memory) could enable adversaries to flip bits in memory cells they do not have permission to access. This is a hardware-based attack that undermines software protection.

Project Goal

- Implement an open-source DRAM testing framework on an FPGA.
- Determine the logical -> physical address mapping for a DRAM chip.
- Use a framework to characterize the Rowhammer vulnerability of a real DDR3 chip. For a 512 Mb memory range:
  - Test the vulnerability of each row to a single-sided Rowhammer attack
  - Test the vulnerability of each row to a double-sided Rowhammer attack
  - Analyze your results
    * Find the minimum number of hammers required to flip a bit.
    * Find the average number of bit flips across all rows.
    * Do the same bits get flipped in every row if you repeat your test twice? Do different bits get flipped?

Hardware Requirements

- Basys 3 : https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Software Requirements

- AntMicro DDR Rowhammer testing framework: https://rowhammer-tester.readthedocs.io/en/latest/index.html
- Xilinx Vivado
- Scripting language for analyzing/visualizing your results. Python, MATLAB, or others.

Report to submit

- IEEE conference format. (4-6 pages)
- The report should include (approximate marks dist.):
  1. Top Sheet with group members names and ID
  2. Introduction: explain the Rowhammer bug, its causes, and its implications for digital computers. (10%)
  3. Methods:
     (a) Explain your experimental setup (15%)

i. How does the AntMicro rowhammer tester work?

ii. What setup are you using to test a generic processor?

(b) Explain the procedure for conducting your tests. (20%)

4. Results: Present bit flip statistics, numbers, plots, etc. (20%)

5. Discussion: (20%)

(a) Did your results match your expectations?

(b) Are there any anomalies?

(c) Comparison between FPGA setup and software setup.

6. Conclusion and personal comments (10%)

7. Clarity, organization, etc. (5%)

Submission guidelines

- Submit .zip file with name: Project#Group#
- Include all the following:
  1. Report.pdf
  2. `build`folder produced by the Rowhammer testing framework for your Arty A7
  3. Software directory containing code run on a commercial microprocessor to test Rowhammer vulnerability.
  4. Working directory for your analysis
  5. A `readme.txt`file that explains anything we need to know to reproduce your results.
     - All data produced by your Rowhammer analysis.
     - All scripts are written to produce your experimental results.
     - Raw data files (plots, tables, etc.) exported from your analysis scripts. These don't have to be in any particular directory structure; we'd like to see them if you have them.

References

1. Y. Kim et al., Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors, in 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), Jun. 2014, pp. 361372. doi: 10.1109/ISCA.2014.6853210.
2. M. Seaborn and T. Dullien, Exploiting the DRAM rowhammer bug to gain kernel privileges, Google Project Zero Blog, March 9, 2015.
3. S. Qazi, Y. Kim, N. Boichat, E. Shiu, and M. Nissler, Introducing Half-Double: New hammering technique for DRAM Rowhammer bug, Google Security Blog, May 25, 2021.