ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research

Colin O'Flynn and Zhizhang (David) Chen

Dalhousie University, Halifax, Canada
{coflynn, z.chen}@dal.ca

Abstract. This paper introduces a complete side channel analysis toolbox, inclusive of the analog capture hardware, target device, capture software, and analysis software. The highly modular design allows use of the hardware and software with a variety of existing systems. The hardware uses a synchronous capture method which greatly reduces the required sample rate, while also reducing the data storage requirement, and improving synchronization of traces. The synchronous nature of the hardware lends itself to fault injection, and a module to generate glitches of programmable width is also provided. The entire design (hardware and software) is open-source, and maintained in a publicly available repository. Several long example capture traces are provided for researchers looking to evaluate standard cryptographic implementations.

Keywords: side-channel analysis, acquisition, synchronization, FPGA

1 Introduction

The introduction of Differential Power Analysis (DPA)[1] spurned interest in the vulnerabilities of embedded systems previously thought to be secure. The difficulty in comparing results of attacks on different platforms was realized early on, and the SASEBO board aimed to provide a standard platform for attacking [2]. Likewise it was realized that for new entrants into the world of side-channel attacks, having available code and algorithms was a useful starting point such as the OpenSCA toolbox [3], and the DPA Book[4]. Despite this, there is still considerable progress to be made. A researcher looking to replicate existing work, even if that work uses a board such as the SASEBO/SAKURA board, needs to purchase an oscilloscope, interface the oscilloscope to the computer, and (re)implement the attack.

Work into making a complete platform has already been presented, for example the GIAnT system, which even uses the same FPGA board as this work [5][6] (and with additional details in [7]). This work was developed in parallel to these systems, and the two systems use different architectures and design languages. The ChipWhisperer system presented in this work is a more modular design, and has more extensive publicly available code for the computer control. Certain features do differ between them: the GIAnT system has a high-speed

Published in Proceedings of COSADE 2014. Author Copy, Updated 10-MAR-2014.

Digital-to-Analog Converter (DAC) for fault injections of adjustable magnitude, something missing on the current ChipWhisperer hardware.

This work presents a side-channel attack platform which integrates all required elements: target device, measurement equipment, capture software, and attack software. This work has benefits for almost any user: students have a low-cost laboratory, researchers have an environment which can be duplicated around the world, and embedded engineers have a method of easily testing published research on their own systems. The entire design (both hardware and software) is open-source, encouraging future development from users. Versions of the project are designed to work with existing hardware, such as the SAKURA-G and SASEBO-W board which researchers may already have access to. Modules to control standard oscilloscopes such as PicScopes and VISA-connected devices are also present, encouraging the use of the ChipWhisperer software with existing measurement labs.

Beyond side-channel attacks, the hardware lends itself to glitch and fault attacks. The device runs synchronous to the device under test (DUT), greatly simplifying the introduction of faults on certain clock cycles. A simple glitch generation module is included for inserting glitches at specific offsets from the clock edge.

2 Hardware

The hardware consists of both the hardware design and the FPGA code. The system is designed to work with several different FPGA boards, all based on the Spartan 6 FPGA. A 'reference' FPGA board is also provided based on a commercially-available FPGA module, shown in Fig. 1. This has a ZTEX FPGA Module with a Spartan 6 LX25 FPGA, however these modules are available in sizes from the LX9 – LX150. Researchers interested in implementing more logic inside the control FPGA may simply switch the ZTEX module for a larger one.

This board provides several features specific to side-channel analysis: two headers for mounting ADC or DAC boards, an AVR programmer, voltage-level translators for the target device, clock inputs, power for a differential probe and Low Noise Amplifier (LNA), external Phase Locked Loop (PLL) for clock recovery, and extension connectors for future improvements such as fault injection hardware. This board will be referred to as the *ChipWhisperer Capture Rev2*.

2.1 Modular FPGA Design

The blocks within the FPGA are designed around a central 'register control' module, as shown in Fig. 3. The design greatly simplifies the addition of new modules: only one small section of the design needs to be modified to insert the bus connections, otherwise the new module can live independently of the rest of the system.

The modular design allows customizing of which modules to include of interest to the researcher; including for example only the clock glitching module if it is desired to work with fault attacks and use a smaller FPGA.

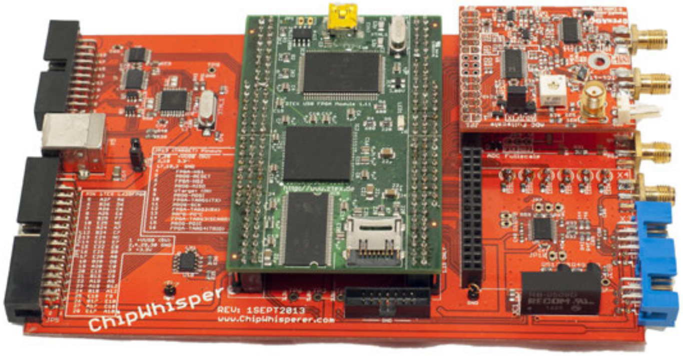


Fig. 1. The reference implementation runs on a ZTEX Spartan 6 LX25 FPGA Module, with an OpenADC as the analog front-end. The completed board is referred to as the ChipWhisperer Capture Rev2.

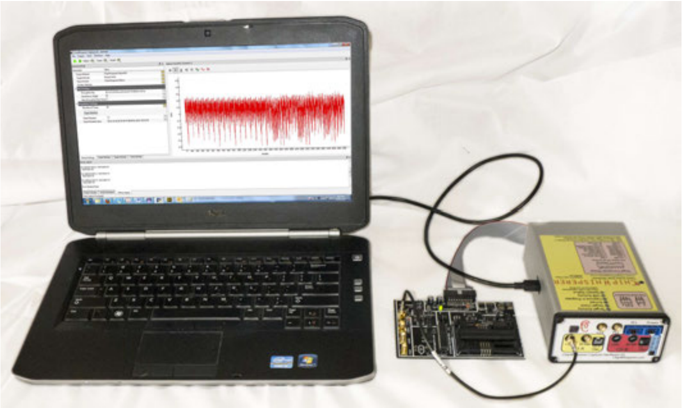


Fig. 2. The complete system, including the FPGA board from Fig. 1 which is mounted in an enclosure, a laptop computer, and the example capture board from Fig. 8. The system can also use a breakout board to connect other embedded hardware targets.

2.2 Capture and Clock Control

If the underlying objective is to measure data on the clock edges of the system clock, sampling at the clock rate of the system is sufficient, provided such samples occur at the correct moment (i.e. on the clock edge). This sampling technique is called synchronous sampling, where the sample clock is synchronized to the device clock. The application of this to side-channel analysis was described and demonstrated in [8], where the SASEBO-GII board was attacked, and this demonstrated how sampling at 96 MS/s synchronously achieved similar results to sampling at 2 GS/s asynchronously.

The analog front-end is based on the OpenADC [8], which provides a -5 dB – 55 dB gain, simplifying measurement of low-level signals. Additionally designs

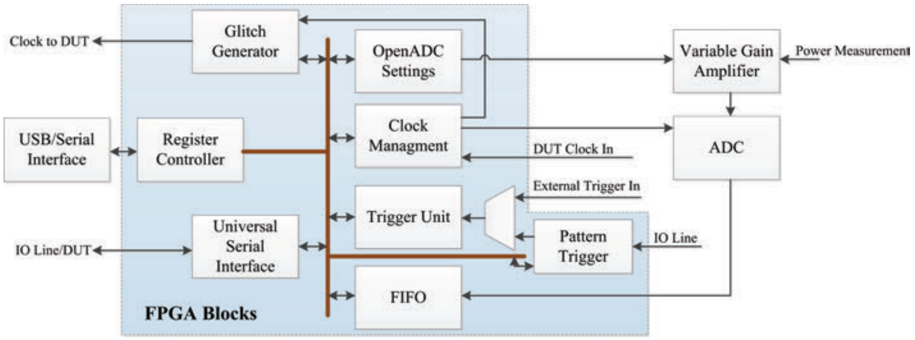


Fig. 3. The base design consists of several blocks connected via a internal bus in the FPGA, shown in the blue box.

for a differential probe and Low Noise Amplifier (LNA) are provided, an example shown in Fig. 4.

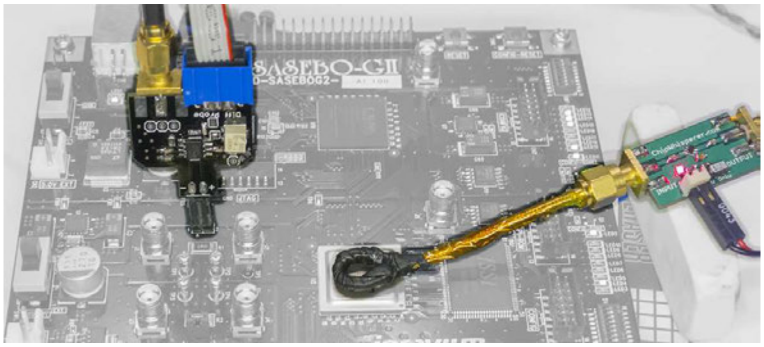


Fig. 4. Beyond capture hardware, design for a differential probe and H–Field probe with LNA are available.

For the synchronous sampling to work, the device must be able to lock onto the system clock. If the clock is readily available as a digital signal (e.g. from the crystal oscillator on the DUT), it can be fed into the FPGA directly, where internally it can be multiplied if desired. If the clock is not available, such as in the case of internal RC oscillators, an external PLL can be used with clock recovery logic to recover the clock [9]. Finally an asynchronous clock is available, although due to the limited sample-rate in this platform will have very poor performance compared to synchronous capture [8].

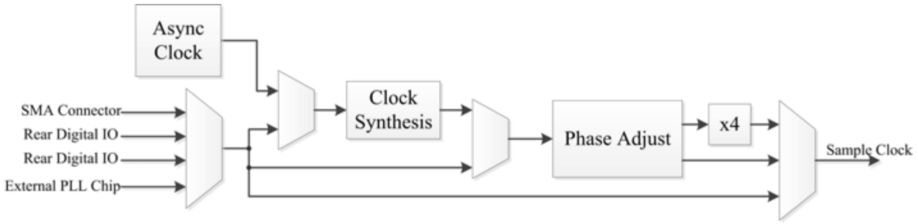


Fig. 5. Clock Routing in ChipWhisperer capture hardware.

2.3 Target Control and Triggering

The FPGA provides some basic IO blocks for driving standard devices. This includes a UART, a Smart Card interface, and Universal Serial IO device, which can be controlled from the computer. Note the target device can be driven by an existing connection instead; the FPGA-based IO blocks are provided simply as a convenience to allow a single USB connection to provide both communications and target control.

Several triggering options are provided. The most basic allows standard triggering: triggering on the rising edge of a digital line for example. This is suitable when analyzing devices which the researcher has programmed, and is able to insert a suitable trigger event into. For more realistic examples, two additional triggering blocks are provided. The first is a digital pattern match, which looks for a specific sequence of transitions on an IO line. This is implemented as a state machine, where it moves through to the next state only if the IO line remains in the expected state for the 'allowed' amount of time. If the IO line fails to match the expected state transition, the state machine resets. This system is specifically designed for triggering on communications protocols, for example by waiting on a response byte. The final triggering system looks for an analog pattern in the waveform, using a Sum of Absolute Difference (SAD) criteria, which is frequently used in video compression, and fast FPGA implementions exist for [10]. Here the system continuously compares the incoming waveform to a known pattern: when the SAD criteria falls below some threshold, the system triggers the capture.

All of these triggering options feature a pretrigger ability. The capture buffer is continuously filled, meaning that the trigger can occur after the actual cryptographic operation has occurred. The limit is simply the size of the capture buffer, which is primarily dependent on the size of the chosen FPGA.

The trigger out signal can be dynamically routed to an external pin. This allows triggering of external equipment with this advanced trigger source.

2.4 Glitch Generation

A clock glitch module is also present in the system. Using two adjustable delay lines built into the FPGA, it can insert glitches into a 'target clock': the 'target clock' either coming from the device under test or generated by the FPGA itself.

The glitch width can be adjusted from about 3 nS to 100 nS (maximum width limited to 50% of clock period or 100 nS, whichever is smaller) and the offset from the clock edge is adjustable from -50% to +50% of the clock period. The specific resolution of the glitch and offset varies for the target clock frequency, but is always smaller than 100 pS. Fig. 6 shows an example of a glitch inserted into the output clock, although the glitch itself can be output separately from the clock for driving modules such as optical or electromagnetic fault injection. This method is the same as described in [11], although with improved resolution on the glitch width and location. The minimum glitch width is limited by the FPGA IO-pin speed: using a faster IO-standard (e.g. LVDS) allows a smaller minimum glitch width if required.

The use of the special Digital Clock Manager (DCM) blocks along with partial reconfiguration (discussed next) allow this extremely fine-grained control over glitch width. This is an improvement on previously proposed glitch generation methods where the 'coarse' width control comes from a higher-speed clock, which generally limits glitch 'coarse' control to a multiple of this clock speed (often 4–10 nS)[7].

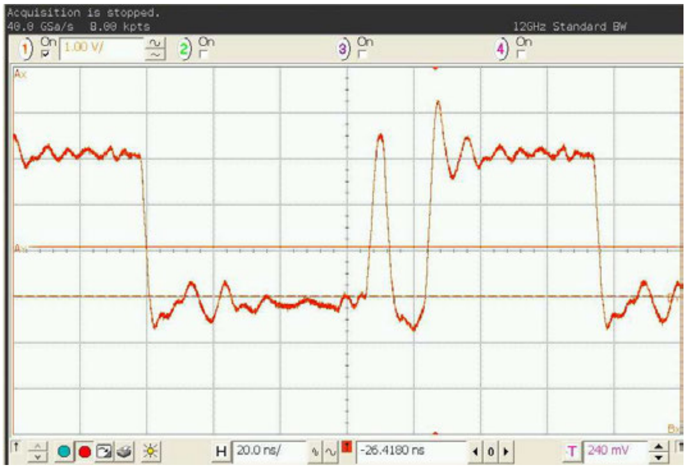


Fig. 6. Inserting a glitch into a 7.37 MHz clock coming from a target device.

2.5 Partial Reconfiguration

The clock generation and glitch generation modules both use the Digital Clock Manager (DCM) blocks within the FPGA. These blocks have limited support for run-time configuration of parameters such as phase delay and frequency generation, and for maximum performance the configuration must be fixed at design time. The Xilinx-provided *run-time* adjustment can shift the phase only by about ±5 nS in 30 pS increments (exact values vary with operating conditions).

To allow adjustments over a wider phase range, a partial reconfiguration interface is provided. This interface allows changes to the FPGA configuration while the system is operating. This is specifically used to reconfigure the DCM blocks for a variety of parameters which are fixed at the implementation stage. This partial reconfiguration requires that appropriate 'bitstream difference' files are generated by the FPGA tools for every possible setting of the desired parameter, e.g. the DCM phase delay attribute. Due to the opaque nature of the FPGA tools there is no simple mapping between parameter changes and a specific portion of the bitstream.

To generate these bitstream difference files, Xilinx's *FPGA Editor* tool is used to modify the Native Circuit Description (NCD) file for the design. A script generates versions of the NCD file with every possible setting of the desired attribute, e.g. for the DCM block with a fixed phase value of -255 to $+255$. These NCD files are converted into bitstream difference files with the Xilinx *bitgen* utility. Setting the desired DCM fixed phase offset means loading the appropriate bitstream difference file[1].

2.6 Implementation on Other Boards

This entire system is implemented as generic FPGA blocks. Whilst a reference platform is provided, it can be used on any FPGA platform. For example this system can be programmed into the control FPGA provided in the SAKURA-G or SASEBO-W platform. Fig. 7 shows a photo of the SAKURA-G board with an OpenADC mounted. This system allows implementation of a cryptographic algorithm in the main FPGA on the SAKURA-G, while the control FPGA serves to actually perform the measurements. Any of the available blocks can be inserted into this system, for example adding clock glitch generation into the control FPGA for the SASEBO-W.

2.7 Generic Device Under Test Board

For demonstration of basic attacks, a generic target board is provided. This board provides several target options: a 28-pin AVR socket, an XMEGA device, and a Smart Card socket. The board also has two LNAs built onto it, along with several clock options. Jumpers can select which target is connected, measurement mode (high-side or low-side shunt), connect the AVR programmer from the ChipWhisperer Capture Rev2, and allows an external Smart Card reader to be connected to the smart card socket by way of a feed-through smart card PCB. The target board is shown in Fig. 8.

The choice of a 28-pin AVR socket allows the board to accept many similar AVR devices. Attacks targeting recent processes can use the AtMega328P or At-

[1] See the ChipWhisperer sources for details, with additional information in the June 2014 issue of Circuit Cellar and at `programmablelogicinpractice.com/?p=143`

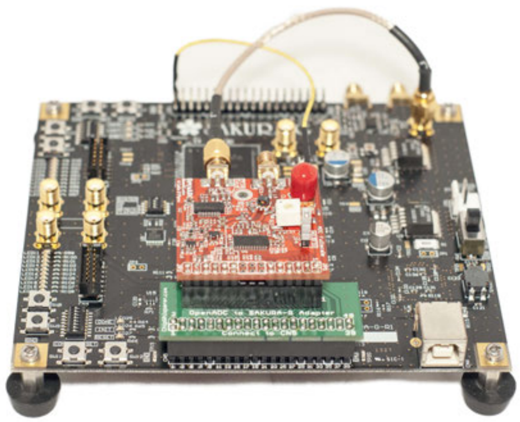


Fig. 7. The modular design allows easy implementation on other hardware, such as the SAKURA-G board. A Spartan 6 LX75 FPGA is used for a cryptographic algorithm, and the LX9 FPGA is used for control of the FPGA along with capturing of power traces. This replaces both the 'ChipWhisperer Capture Rev2' hardware box and the 'Target Device', meaning the entire side-channel analysis system is present on the SAKURA-G board.

Mega48A. Attacks looking to test older devices may use an older Mega8 device[2]. Note that many 'Smart Card' attacks are tested on a AtMega163 card, which contains an AtMega163 die from Atmel. Existing code targeting the AtMega163 can be ported to work on a 28-pin AVR, avoiding the need to find outdated Mega163 cards, and also using a more recent semiconductor process

3 Software Architecture

The software is implemented in entirely in Python. Python was chosen for a variety of reasons: it is natively cross-platform, provides a simple GUI through PySide, can easily interface to other languages including C/C++ and MATLAB, provides high performance using Cython, and has a large collection of modules which provide functionality such as cryptographic functions, plotting, numeric computations, low-level IO, and smart card interfaces. In addition the choice of an interpreted language such as Python enables considerably more advanced scripting options. This section will only briefly outline the software architecture, full documentation is kept using the Sphinx system[3], which combines documentation built into the Python source with additional files. This

[2] While the Mega8 is an older device, recently bought ones may be produced on newer processes. If looking for a device produced on older IC process, one will need to confirm the production date via the date code

[3] `sphinx-doc.org`

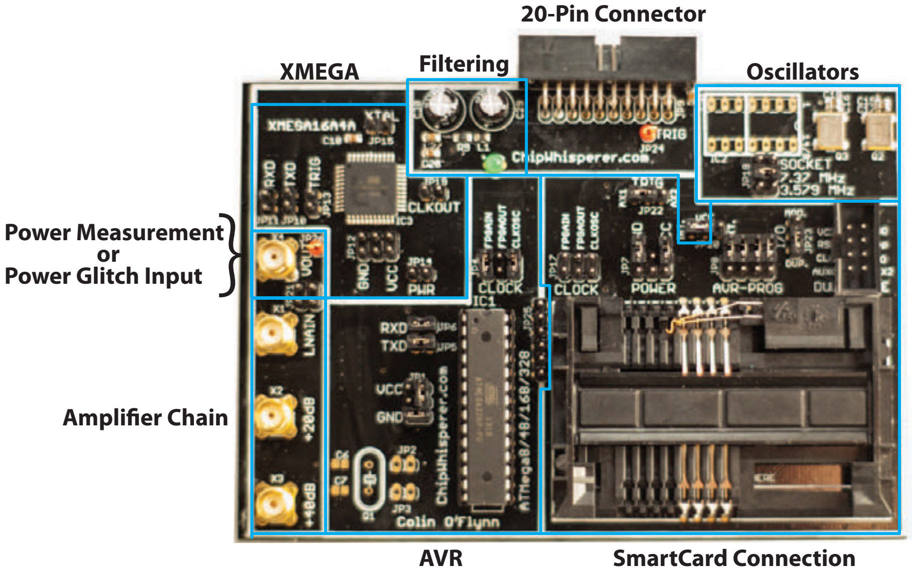


Fig. 8. The Multi-Target board provides a simple platform for testing various attacks and cryptographic implementations.

documentation is linked from www.ChipWhisperer.com, or available directly at www.newae.com/sidechannel/cwdocs/.

The project is split into two programs. One program captures the power traces and saves them, the other perform side-channel analysis algorithms (e.g. CPA). The decision to split the project into two programs was done to allow use of only part of this project by other researchers. For example researchers with existing attack code can still use the capture program, for example saving the data to a MATLAB workspace. Or researchers with existing traces can load them into the analysis program without using the capture portion.

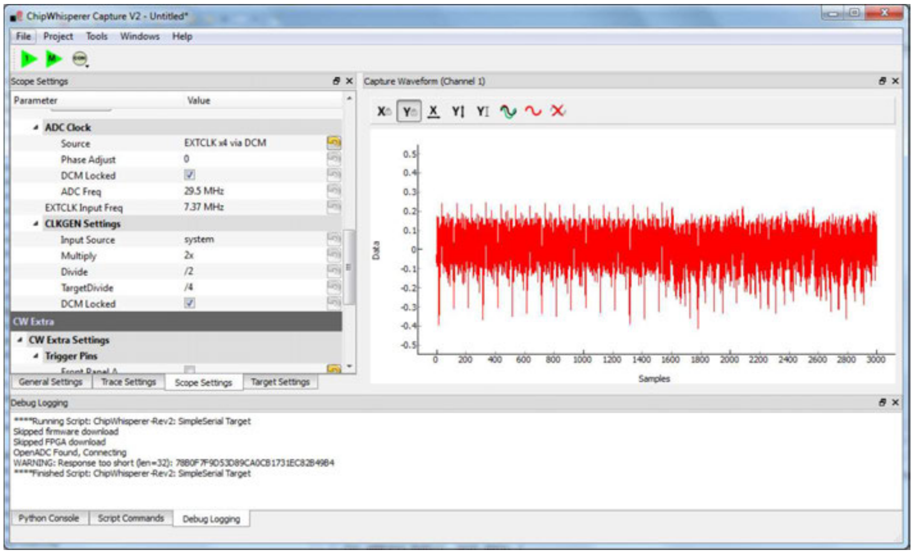
Both capture and analysis software share a common base class; this class defines methods of modifying parameters, saving and restoring projects, using trace files, and plotting data. In addition support for a special 'scripting' system is provided. This 'scripting' language allows execution of either the capture or analysis software from another Python application, which forms the 'script'. This design is especially powerful since it allows the script to call any function within the entire program, and does not require the definition of specific scripting commands. As an example, when adding a new FPGA module, it requires the addition of the appropriate driver module to the capture software. However by using the scripting interface, it is possible to simply send raw commands to this module, which allow testing and debugging it before the complete driver has been written. When options are changed using the GUI, the GUI also shows how to accomplish the same operation with a script. Thus a user can simply setup appropriate options from the GUI, and then save these script commands to recreate the configuration. Listing 1.1 in Appendix A shows an example of such a script.

The graph windows allow transformations to be performed on the data, such as switching from time-domain to frequency-domain through an FFT, filtering and smoothing, and exporting data in the graph.

3.1 Trace Management

The trace management module is common to both the capture and analysis software, which provides a method of mapping traces from different formats into a continuous block of trace data. The default storage method used by the software uses the Python NumPY library's native save and load commands. Alternatives which store the traces to a MySQL server, and saving the traces to the same format used by the DPAContestv3 tools are also provided.

4 Capture Software



Parameter	Value
ADC Clock	
Source	EXTCLK x4 via DCM
Phase Adjust	0
DCM Locked	☑
ADC Freq	29.5 MHz
EXTCLK Input Freq	7.37 MHz
CLKGEN Settings	
Input Source	system
Multiply	2x
Divide	/2
TargetDivide	/4
DCM Locked	☑
CW Extra	
CW Extra Settings	
Trigger Pins	
Front Panel A	□

```
****Running Script: ChipWhisperer-Rev2: SimpleSerial Target
Skipped firmware download
Skipped FPGA download
OpenADC Found, Connecting
WARNING: Response too short (len=32): 7880F7F9D53D89CA0CB1731EC82B49B4
****Finished Script: ChipWhisperer-Rev2: SimpleSerial Target
```

Fig. 9. The Capture GUI provides an interface to the capture hardware, target device, and storage media.

The capture GUI is ideal for initial experimentations with a new system, such as trying different ADC settings or different trigger settings. For repeatable captures it is desirable to instead script the setup, which sets appropriate values to various parameters. In addition this script can perform actions such as saving the target data to a different format (e.g. a MATLAB workspace), or used for automatically performing captures under different target conditions (e.g. selecting an algorithm with and without countermeasures).

The script can either run the entire capture program, or simply configure part of the window. Several example scripts are provided, which can be loaded in an already-running ChipWhisperer-Capture application. See Listing 1.1 in Appendix A for an example script.

For debug purposes, a monitor window shows the input and output results of the cryptographic operation being performed on the target. This window also displays what the expected result would be for a known key. This can be quickly used to confirm a device under test is operating as expected, and the encryption key was correctly loaded into the device.

4.1 Capture Performance

The capture performance demonstrates how quickly traces can be captured with the default system. The measurements are taken on two different computers: a Windows 7 based Intel i5-2540M laptop, and a Linux based AMD A10-5800K desktop. Captures are averaged over 10,000 traces. The 'Target Connection' indicates how the device under test (e.g. the cryptographic target, *not* the ChipWhisperer capture) connects to the computer. The 'FPGA-x' mode means one of the ChipWhisperer IO blocks are being used.

For high-speed USB targets (ChipWhisperer Rev2, SAKURA-G, SASEBO-W) the capture speed is primarily limited by USB latency in the host computer stack. Note the AVR target shows both 3000 and 20000 points per trace; the resulting speed change is much less than the 6x increase in data size would suggest. Targets connected to the computer directly run much faster, as the IO blocks in the ChipWhisperer tend to required several USB transactions, each transaction adding latency from the USB stack. This suggests that there is considerable room for speed improvements by streamlining transactions to reduce this latency.

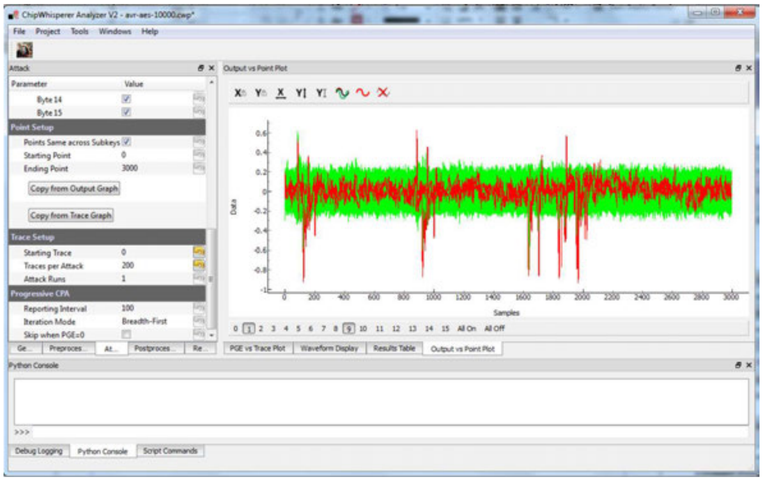
In addition to the OpenADC capture hardware, three standard oscilloscopes are shown for comparison. The capture software can be programmed to support other oscilloscopes with minimal changes.

5 Analysis Software

The analysis software uses the same project and trace management system used by the capture software. Moving a project over simply means saving the project in the capture software, and opening it in the analysis software. Alternatively, traces can be manually imported, either if they come from an external source or if you wish to combine traces from several different captures into a single analysis run. In addition this manual mode is used for configuring database operation; in this mode the analysis software can read traces from a MySQL database, which allows analysis to occur while the capture is still ongoing.

Table 1. Traces/Second for various targets and capture hardware. Points/Trace varies by target, and indicates number of points stored in each trace to attack the target. The '—' indicates lack of supported driver for the host OS.

Capture Hardware	Attack Target	Target Connection	Points /Trace	Traces/Second Win7	Linux
ChipWhisperer Rev2	SASEBO-GII	USB	100	14.8	28.3
ChipWhisperer Rev2	AVR, 38400 Baud	FPGA-Serial	3000	11.3	3.91
ChipWhisperer Rev2	AVR, 38400 Baud	FPGA-Serial	20000	7.04	3.78
ChipWhisperer Rev2	AVR, 38400 Baud	USB-Serial	3000	18.2	18.9
ChipWhisperer Rev2	SmartCard	PS/SC Reader	3000	7.40	6.62
SAKURA-G	SAKURA-G	Integrated	400	6.67	7.18
SASEBO-W	SmartCard	FPGA-USI	3000	0.271	0.279
SASEBO-W	SmartCard	FPGA-SmartCard	3000	1.49	1.52
Agilent MSO54831D	SASEBO-GII	USB	1500	8.01	—
PicoScope 6403D	SASEBO-GII	USB	1500	12.1	43.6
PicoScope 6403D	SAKURA-G	USB	1500	15.4	29.6
PicoScope 5444B	AVR, 38400 Baud	USB-Serial	12000	16.4	5.63



Parameter	Value
Byte 14	☑
Byte 15	☑
Point Setup	
Points Same across Subkeys	☑
Starting Point	0
Ending Point	3000
Trace Setup	
Starting Trace	0
Traces per Attack	200
Attack Runs	1
Progressive CPA	
Reporting Interval	100
Iteration Mode	Breadth-First
Skip when PGE=0	□

Fig. 10. The Analyzer GUI runs a given attack on the stored traces.

When traces are loaded, a single trace is plotted in time. A number of traces can be overlaid on each other, which is useful to confirm the synchronization of traces. If traces are incorrectly synchronized, one of the preprocessing modules (described next) can be used to resynchronize the traces.

5.1 Preprocessing

Several basic preprocessing modules are provided, which operate on the data before passing through to the attack. Three types of resynchronization are implemented: a sum-of-errors minimizer, peak detect, and cross-correlation. These methods provide very simple sliding resynchronization, which works well with the synchronous capture methodology of the ChipWhisperer Capture hardware. In addition a simple low-pass filter is also provided. Any of the preprocessing modules can be chained together in an arbitrary order, and additional modules can trivially be added to the system.

The waveform display window shows the results after the preprocessing chain. This could be used to confirm that traces are properly resynchronized in the time domain before continuing on to the attack.

5.2 Attack Implementation

The attack module is designed to simplify how new attacks are added to the system. The cryptographic model, leakage model, and attack algorithm are all separate modules. This greatly simplifies changes and increases reuse: if a new attack is added, it can pull in the existing cryptographic model and leakage model modules to automatically work with both software and hardware AES implementations. If a new cryptographic model is added such as DES for example, it should work with the existing CPA attack.

The main attack implemented currently is a CPA attack. Fig. 11 shows the data flow within the attack system. Data coming from the 'Trace Container' may also have had preprocessing applied, or a certain window of data may be selected instead of the entire trace range. The correlation calculation has several modules that can be loaded, for example selecting between a version that takes advantage of the fast NumPy library, and a version compiled in C using Cython. The 'correlation calculation' may actually implement more advanced algorithms, an example is the Bayesian calculation given in [12] is also implemented. The results are stored in the 'Attack Statistics', which stores results after a given number of traces, and also calculates metrics such as Partial Guessing Entropy (PGE) at the current state [13].

The number of traces to use in the attack is also configurable, and allows for looping the attack several times over different sections of the traces. If 100,000 traces are present in the project for example, one could perform the attack with 1000 traces, and repeat it 100 times. When final metrics are calculated, the system can average the results of all 100 attacks.

Correlation Power Analysis The basic equation for a Correlation Power Analysis (CPA) attack, where $r_{i,j}$ is the correlation coefficient at point j for hypothesis i, $t_{d,j}$ is power measurement of trace number d at point j, and $h_{d,i}$ is the hypothetical power consumption of hypothesis i for trace number d, with a total of D traces is given in equation 1. This basic formula does not allow online calculation, where new traces are easily added without recalculation of the entire sum. Instead the form shown in equation 2 is used [14]. This form lends itself to online calculation, where when a new trace is added the sums are updated and the new correlation coefficient calculated.

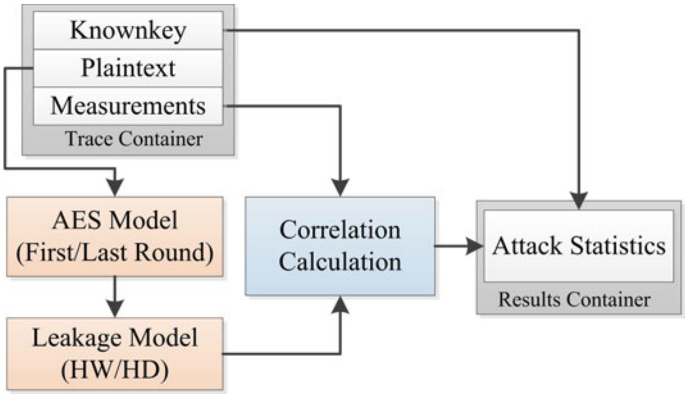


Fig. 11. In the CPA attack module, the cryptographic model, hardware model, and statistics update are all separate. This allows simple selecting of Hamming-Weight (HW) or Hamming-Distance (HD) models for example, or selecting the AES round to attack.

The online update is used during calculation of attack statistics, where it is desired to track the attack results as new traces are added. As a practical matter, note the denominator of either equation 1 or equation 2 may have numeric stability problems due to cancellation in either of the two terms. Forms given in [15] may result in more stable calculations, however experimental results shown that for measurements of real systems the numerical stability is not an issue.

$$r_{i,j} = \frac{\sum_{d=1}^{D}\left[\left(h_{d,i}-\overline{h_i}\right)\left(t_{d,j}-\overline{t_j}\right)\right]}{\sqrt{\sum_{d=1}^{D}\left(h_{d,i}-\overline{h_i}\right)^2 \sum_{d=1}^{D}\left(t_{d,j}-\overline{t_j}\right)^2}} \tag{1}$$

$$r_{i,j} = \frac{D\sum_{d=1}^{D} h_{d,i}t_{d,j} - \sum_{d=1}^{D} h_{d,i}\sum_{d=1}^{D} t_{d,j}}{\sqrt{\left(\left(\sum_{d=1}^{D} h_{d,i}\right)^2 - D\sum_{d=1}^{D} h_{d,i}^2\right)\left(\left(\sum_{d=1}^{D} t_{d,j}\right)^2 - D\sum_{d=1}^{D} t_{d,j}^2\right)}} \tag{2}$$

5.3 Results Display

The output of the attack is stored in a 'Results Container' type. Generally the assumption is that the output of the attack container will contain a metric for each hypothesis of the subkey, at each point in time. The metric can be sorted to give the most likely subkey hypothesis.

The results display module can plot the value of the attack output for each point in time, for each of the hypothetical keys. If the correct key is known, this is plotted in a separate color as in Fig. 10. In addition if the correct key is known additional metrics can be calculated, such as the Partial Guessing Entropy (PGE). As previous mentioned this PGE will be averaged over many trials when available.

6 Example Results

This section demonstrates some example results of the platform. Two example devices will be tested: the first is the AtMega328P microcontroller, which is a reasonably recent AVR microcontroller from Atmel. The system is loaded with basic code which performs encryptions when requested over a serial protocol, and returns the encryption result. The Partial Guessing Entropy (PGE) of the CPA attack is shown in Fig. 13 using the *ChipWhisperer Capture Rev2* hardware (i.e. as in Fig. 2). As a comparison Fig. 14 shows the PGE of the same attack where traces have been recorded with a normal oscilloscope.

This device can also be targeted for glitch attacks. When the AVR was running at 7.3728 MHz, glitches were inserted into the clock with the following specifications: glitch width of 15.2% (20.6 nS), glitch offset of -17.0% (-23.1 nS). The glitch was XOR'd with the clock, and repeated on 200 consecutive clock edges. The objective was simply to cause the embedded system to skip authentication code, which was successfully accomplished.

A second example uses the SASEBO-GII board running at 24 MHz, with the AES core loaded from the 'DPA Contest V3', and the results plotted in Fig. 12. Comparison to previously published results from the SASEBO-GII board indicate the ChipWhisperer system is performing as expected[8].

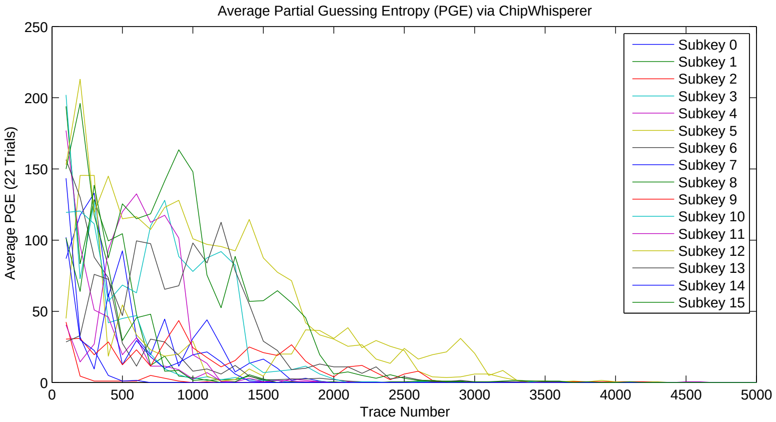


Fig. 12. Partial Guessing Entropy (PGE) of SASEBO-GII running at 24 MHz. No smoothing has been applied.

7 Conclusion and Future Work

This work has demonstrated a embedded security analysis platform, which is completely self-contained and requires no additional hardware or software besides a standard computer. The design is extremely modular and allows users

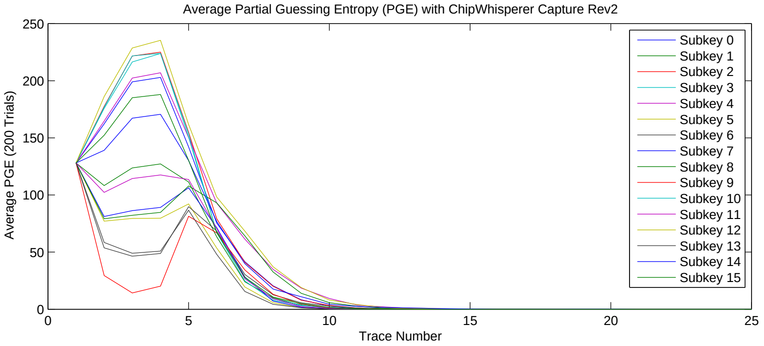


Fig. 13. Partial Guessing Entropy (PGE) of CPA attack against AES-128 running on AVR Microcontroller. Traces recorded with ChipWhisperer Capture Rev2 hardware at 29.5 MS/s synchronous to device clock. No smoothing has been applied, graph comes from ChipWhisperer Analyzer software.

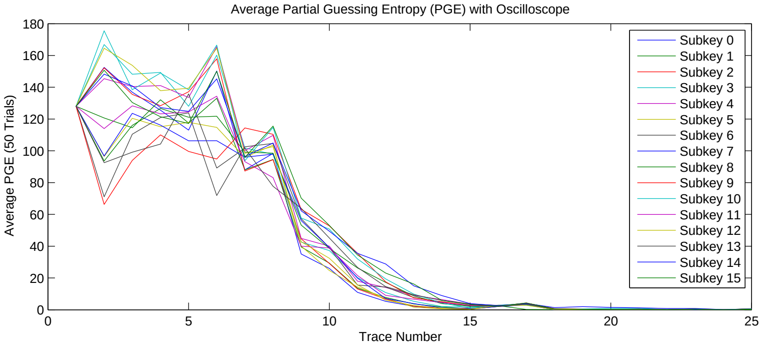


Fig. 14. Partial Guessing Entropy (PGE) of CPA attack against AES-128 running on AVR Microcontroller. Traces recorded with PicoScope 6403D at 312 MS/s (as with any oscilloscope, this sampling is done asynchronous to device clock). No smoothing has been applied, graph comes from ChipWhisperer Analyzer software.

to use only a portion of the design; for example using a normal oscilloscope with this system, taking advantage of the advanced triggering mechanisms or the clock glitching capability without using the analog capture hardware.

All design material including source code and hardware design files are maintained in a GIT repository at `www.ChipWhisperer.com`. A wiki is used to maintain documentation, and contributions to either documentation or design are welcome. For users interested in the analysis algorithms, large example captures are available as well: a set of 500,000 traces of AES-128 executed on an AtMega328P microcontroller along with 500,000 traces from the SASEBO-GII. Example traces from other hardware is also available, and community submissions are welcomed.

Acknowledgments Colin O'Flynn's funding is provided by NSERC Canada Graduate Scholarship. Thanks to Akashi Satoh for donation of the SAKURA-G used in this work, and Akashi Satoh and Pankaj Rohatgi for donation of the SASEBO-GII and SASEBO-W also used in this work. Thanks to COSADE 2014 reviewers for many insightful comments on initial revision of this papers.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO' 99, Springer-Verlag (1999) 388–397
2. Satoh, A.: Side-channel Attack Standard Evaluation Board (SASEBO). `http://www.morita-tech.co.jp/SASEBO/en/index.html` (2011)
3. Oswald, E.: OpenSCA: A Matlab-based open source framework for side-channel attacks (2009) `http://opensca.sourceforge.net/`.
4. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Advances in information security. Springer (2008)
5. Oswald, D., Kasper, T., Markhoff, S., Paar, C.: FPGA-based Implementation Attacks with GIAnT (November 2011) 9th CrypArchi Workschop, Bochum.
6. Oswald, D.: Implementation Attacks: From Theory to Practice. PhD thesis, Ruhr University Bochum (September 2013)
7. Kasper, T., Oswald, D., Paar, C.: A Versatile Framework for Implementation Attacks on Cryptographic RFIDs and Embedded Devices. In Gavrilova, M.L., Tan, C.J.K., Moreno, E.D., eds.: Transactions on Computational Science X. Springer-Verlag, Berlin, Heidelberg (2010) 100–130
8. O'Flynn, C., Chen, Z.D.: A case study of Side-Channel Analysis using Decoupling Capacitor Power Measurement with the OpenADC. Lecture Notes in Computer Science **7743** (2013) 328–344
9. O'Flynn, C., Chen, Z.D.: Synchronous Sampling and Clock Recovery of Internal Oscillators for Side Channel Analysis. Cryptology ePrint Archive, Report 2013/294 (2013)
10. Olivares, J., Hormigo, J., Villalba, J., Benavides, I.: Minimum Sum of Absolute Differences Implementation in a Single FPGA Device. In Becker, J., Platzner, M., Vernalde, S., eds.: Field Programmable Logic and Application. Volume 3203 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2004) 986–990
11. Balasch, J., Gierlichs, B., Verbauwhede, I.: An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In: Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography. FDTC '11, Washington, DC, USA, IEEE Computer Society (2011) 105–114
12. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.X.: An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks. In Knudsen, L., Wu, H., eds.: Selected Areas in Cryptography. Volume 7707 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 390–406
13. Standaert, F., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. Advances in Cryptology-Eurocrypt 2009 (2009) 443–461
14. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. Cryptographic Hardware and Embedded Systems - CHES 2004 (2004) 135–152
15. Chan, T.F., Golub, G.H., Leveque, R.J.: Algorithms for Computing the Sample Variance: Analysis and Recommendations. The American Statistician **37**(3) (1983) 242–247

8 Appendix A: Script Example

Listing 1.1. Example user script for running automated capture and saving traces to MATLAB file

```

# Example of using a list to set parameters.
lst = [
# Clock Routing Setup
 ['CW Extra', 'CW Extra Settings', 'Clock Source',
                                   'Target IO-IN'],
 ['OpenADC', 'Clock Setup', 'ADC Clock', 'Source',
                                   'EXTCLK x4 via DCM'],
# Sample Length/Offset Setup
 ['OpenADC', 'Trigger Setup', 'Total Samples', 3000],
 ['OpenADC', 'Trigger Setup', 'Offset'       , 1500],
# Low Noise Amplifier Gain Setting
 ['OpenADC', 'Gain Setting' , 'Setting'      , 45],
# Rising Edge Trigger
 ['OpenADC', 'Trigger Setup', 'Mode'         , 'rising edge'],
# Final step: make DCMs relock in case they lost sync
 ['OpenADC', 'Clock Setup'  , 'Relock DCMs'  , None],
]

# cap variable contains instance of ChipWhispererCapture() and
# was configured in the main script

# Download all hardware setup parameters
for cmd in lstexample: cap.setParameter(cmd)

# Set number of traces
cap.setParameter(['Generic Settings', 'Acquisition Settings',
                  'Number of Traces', 75])

# Capture a few traces initially (not saved)
cap.capture1()
# pe() is a macro which processes any queued events it must
# be called when interacting with the low-level API directly.
pe()
cap.capture1()
pe()

# Start capture process of 75 traces, save to memory
writer = cap.captureM()

# Save files to MATLAB workspace file instead of native format
sio.savemat('sca_data.mat', {'powertrace':writer.traces,
                             'textin'    :writer.textins,
                             'textout'   :writer.textouts,
                             'knownkey'  :writer.knownkey})
```