# Power-Based Side-Channel Attack for AES Key Extraction on a ATMega328 controller

Utsav Banerjee, Lisa Ho, and Skanda Koppula

***Abstract*—We demonstrate extraction of a private-key from Flash program memory on the ATMega328 microcontroller (the controller used on the popular Arduino Genuino/Uno board). We loaded a standard AVR-architecture AES implementation onto the chip and ran this implementation to encrypt 500 randomly chosen plaintexts. By carefully measuring the chips power consumption, we were able to correlate the consumed power with the input plaintexts and key values that might be used to encrypt each AES block, and back-derive the hard-coded key used for encryption. We describe here our test infrastructure for automated power trace collection, an overview of our correlation attack, sanitization of the traces and interesting stumbling blocks encountered during data collection and analysis, and the results of our attack.**

***Index Terms*—AES, side-channel, power consumption, ATMega328, Correlation Power Analysis**

## I. Introduction

Recent concerns about data privacy has brought attention to the necessity of encryption algorithms. One of the more popular symmetric-key algorithms, Advanced Encryption Standard (AES), has been the U.S. government standard since 2002 (ISO/IEC 18033-3), and is used in a multitude of applications: SSL/TLS protocols [1], Kerberos [2], and demonstrably secure embedded devices [3]. This last application in particular, embedded devices, has seen much growth in recent years, given the advantages of computation on smaller embedded devices: low power, lower system latency, and generally smaller device size.

Small hardware implementations, however, are notoriously vulnerable to a range of side-channel attacks [4]. Timing, electromagnetic radiation, and power consumption are just three commonly exploited vectors used to leak information about ongoing computations and data on the chip. Knowing that a device architecture is amiable to side-channel exploitation is useful in deciding whether to execute unprotected sensitive computations or store data on devices with similar memory and processor characteristics.

We aim to demonstrate a reasonably realistic side-channel attack on AES on one such embedded device: the ATMega328 microcontroller produced by Atmel. The ATMega328 is the basis for the widely popular development board, Arduino Uno [1]

In section two, we review for the read the theoretical ideas underpinning our attack. In section three, we describe our implementations our hardware setup, power measurement infrastructure, correlation methodologies, instructive problems that we encountered, and overview the structure of our source code. In section four, we describe quantitatively describe the results of our attack.

## II. Preliminaries

### A. Controller Specifications

The ATMega328 family of chips is an 8-bit microcontroller series with 32 KB of NAND-type flash and 2KB of SRAM. The controller runs off a 16 MHz external clock on the Arduino board. Typical power consumption of the chip ranges from 7 to 12V, with a 20mA current draw, depending on the peripheral and I/O pin usage [7]. Out attack exploits the NAND-type flash memory architecture that consumes marginally more power when accessing addresses that store value-zero (discharge) bits [2] [8].

The encryption program running on our ATMega328 uses an Arduino-specific port of the `avr-crypto-lib` by Davy Landman and Bochum Hackerspace [5] [6]. `AESLib` is one of the more widely-used AES implementations for Arduino, and includes support for ECB and CBC-modes of AES. Our team decided that ECB-mode would be more amiable to a power correlation attack, and correspondingly chose to exploit the library's AES-ECB implementation. We discuss ECB in further depth in section 3.D.

### B. Correlation Power Analysis

## III. Protocols and Procedure

### A. Data Collection Infrastructure

A central piece of our work was developing the serial-connection based data collection framework to feed plaintexts for encryption to the Arduino, and read the resulting power trace from the oscilloscope. An outline is given in Figure 1.

All authors are with the Department of Electrical and Computer Engineering, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA

To contact the authors: `utsav@mit.edu`, `lisaho@mit.edu`, and `skandak@mit.edu`

Manuscript completed for 6.858 Computer Systems Security; completed on December 5, 2015

[1]Other models of the Arduino, such as the Arduino Mega and Arduino Genuino Micro use ATMega chips as well, that have a similar architecture to the ATMega328. It is possible that this attack could be adapted to those chips as well.

[2]In a highly simplified power consumption model, NAND-flash charges a central power line connected to a series of memory cells. Depending on the value stored in the accessed memory cell, the line is discharged or not. Thus, line requires data-dependent recharging.

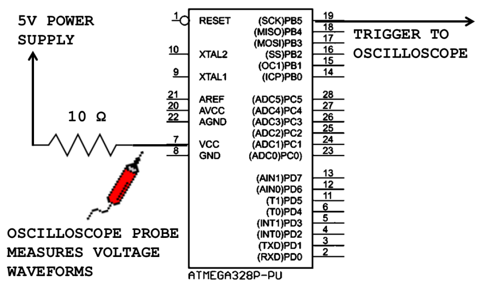

Fig. 1: The oscilloscope probes voltages on the chip's power line, starting automated power trace on a trigger that signals the start of AES

We needed a subset of the power trace that correlated strongly with the input plaintext and key. Specifically, we were interested in finding the portion of the trace that corresponded to the `xor(plaintext,key)` operation, and the the SBOX permutation taken over the `xor` result. In order to automate slicing to this part of the trace, we modified the `AESLib` SBOX assembly code to insert a flag a memory-mapped register that pulled up pin 13. Our oscilloscope triggered on this output, and automatically captured the part of the trace immediately after the pin 13 rising edge.

Central to our collection was a Tektronix 5054B computing oscilloscope that collected samples at 4GHz with internal memory bank of at most 16 million points. The sampling rate limited the resolution of traces that we were able to capture; we originally thought this physical cap on data quality was causing correlation issues we ran into, until we discovered arbitrary DC shifts in the traces. We discuss this challenge in more detail in Section 3.C. We used the oscilloscope's GPIB query interface to automate configuration and downloading of traces.

In the final iteration of our system, we have a orchestrating computer $C$ send plaintexts to the Arduino for encryption every 2 seconds over the Arduino's serial port. The pin 13 trigger resulting in a oscilloscope trace capture, which was sent back to $C$.

More recently, we have been attempting to use the Tektronix $FastFrames$ feature to take capture batches of 2,500-point traces at once, and use one memory-read, GPIB-write operation to transfer these traces to the computer. This would allow us much faster trace measurement, which is currently bottlenecked by the GPIB-write operation. As we will discuss in Section 4, the number of key bytes we can recover is directly correlated with the number of plaintexts we can capture and average.

Photos of our collection framework can be found at `https://www.dropbox.com/sh/usialgelvfqlsdr/AACvqOHKEWoumWNYi2WRIebCa?dl=0`. Of perhaps small interest is the grounded aluminum food box encapsulating the ATMega328 setup, which reduced the EM noise and interference which we noticed in our power traces.

### B. *Implementation of CPA and Power Model*

- discuss different power models we've tried, faster correlation

### C. *Instructive Problems Encountered (and Panaceas)*

-

- remove difAmp -¿ increase resistor value -¿ increase bandwidth?//used to measure current before
- serial print adds noise
- averaging to solve dc shift
- interrupt introduces clock jitter
- adding nops to prevent asyncronous / delay
- original hypothesis about sbox: SBox bad for correlation? the flash architecture -¿ bit block bit block bar
- FUNDAMENTALLY A PROBLEM WITH DATA

### D. *Caveats*

We address concerns and discuss the drawbacks for our chosen method of attack:

– We chose to attack the Arduino library's implementation AES-ECB mode. Althought ECB is not semantically secure (e.g. you can derive information about the plaintext from the ciphertext), ECB is still (unfortunately) used as the default option in a number of crypto-suites, `avr-crypto-lib` included. This is because of it's relatively simple implementation, compared to other more sophisticated modes of AES. Furthermore, our attack does not exploit the plaintext-ciphertext correlations in ECB to derive the key; rather, it uses our power sidechannel. For our team's first power-trace based attack, we chose a mode that we were confident that might have some correlation with the plaintext-key-guess XOR (the first computation in the first step of ECB). It might be possible to adapt our attack to CBC-mode as well. - presence of trigger - more - go over the graphs, and timing/accuracy results of tests

### E. *Overview of Source Code*

- overview everything in dropbox

## IV. Results

2.5 hours for 500 plaintexts 5 minutes to process

Most of our data and results can be found at `https://www.dropbox.com/sh/07xni6s4tu4klme/AABnrBK-QZCVO1tMK4GFeQ5ta?dl=0`.

## Acknowledgment

The authors would like to extend our deepest thanks to Chiraag Juvekar of the Energy Efficient Digital Circuits and Systems group for the time he spent with us aiding our debugging of data collection and analysis problems. We would also like to thank Albert Kwon, our TA, for his insightful advice over the course of the project.

## References

[1] Lee, Homin K and Malkin, Tal and Nahum, Erich, *Cryptographic strength of ssl/tls servers: current and recent practices*. Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, 2007.

[2] Rathore, Romendrapal Singh and Pal, BL and Kumar, Shiv. *Analysis and Improvement in Kerberos 5*. 2015.

[3] Altera Corporation, *FPGAs with built-in AES: The key to secure system designs*. Embedded Computing Design, July 15, 2008.

[4] Oswald, Elisabeth, et al. *Side-Channel Analysis Resistant Description of the AES S-Box*. 2005.

[5] https://github.com/DavyLandman/AESLib. *Arduino AESLib*. 2015.

[6] http://www.das-labor.org/wiki/AVR-Crypto-Lib/en. *AVR-Crypto-Lib*. 2013.

[7] http://goo.gl/hhZF2z. *Atmel 8-bit Microcontroller Datasheet*. 2014.

[8] Handy, Jim. http://goo.gl/5QZHhK *3D NAND: Making a Vertical String*. 2013.