Comprehensive Open-Source SCA Course Modules for Hands-On IoT Security Education

Mateus Augusto Fernandes Amador, Brooks Olney, Srinivas Katkoori, Robert Karam

Comprehensive Open-Source SCA Course Modules for Hands-On IoT Security Education *

Mateus A. Fernandes A.[0000−0003−2109−3683], Brooks Olney[0000−0001−9178−0783], Srinivas Katkoori[0000−0002−7589−5836], and Robert Karam[0000−0002−2713−029X]

University of South Florida, Tampa FL 33620, USA
{mateusf1,brooksolney,katkoori,rkaram}@usf.edu

Abstract. With the rapid growth of the Internet of Things (IoT) and increasing reliance on network-connected devices, IoT security, which integrates components of hardware and cybersecurity, is more important than ever. Hence, we must improve and expand training opportunities for students in IoT security. Experiential learning is an essential component of education for engineering and cybersecurity in particular. In this work, we describe three comprehensive hands-on IoT security experiments built using off-the-shelf development boards which can provide a low-cost and accessible experiential learning opportunity for students in this area.

Keywords: Internet of Things (IoT) · Security · Side-Channel Analysis (SCA) · Education · Experiential Learning

1 Introduction

The Internet of Things (IoT) is an ongoing technology transition with the goal of connecting the unconnected. IoT application domains are varied and diverse, including transportation, healthcare, consumer electronics, public services, defense, and more [7, 14, 15, 23]. These devices are regularly exposed to potential attacks with significant economical losses and public safety risks [1, 18]. Due to the cyberphysical nature of IoT devices, their security necessarily integrates components of hardware and cybersecurity [5, 25]. In particular, hardware security has often been lacking in devices and overlooked by researchers in contrast to software security [24].

To ensure that current and future IoT devices are secure from malicious attacks, we must focus not only on changing development methodologies and prioritizing security as a design metric, but also on addressing deficiencies in training the next generation of IoT engineers [2, 21]. In particular, providing students with interactive practical experiences to efficiently motivate and instill long-term knowledge about IoT security [11]. To this end, we have developed and contextualized several security course modules focusing on IoT security that

* This material is based upon work supported by the National Science Foundation under Grant No. DGE-1954259. Contact: rkaram@usf.edu

can be integrated into upper-level courses to provide students with *hands-on, experiential learning* for IoT security topics.

In this paper, we describe three of these course modules which focus on a wide range of side-channel analysis (SCA) attacks to identify vulnerabilities in IoT devices. In particular, one performs a password checker, another performs a symmetric key encryption (SKE), and another performs a public key encryption (PKE). Students learn the motivation behind these cryptosystems, go through the mathematics of one particular example for each (i.e., simple loop password checker, AES, and RSA respectively), and gain an understanding for how naïve implementations may be vulnerable to various SCA attacks. They then attempt the attacks themselves (password or key recovery), integrate countermeasures with the implementations, and show the efficacy of the countermeasures. These modules, and others in the course, run on inexpensive, commercial/off-the-shelf hardware, and use only open-source tools and languages in an effort to minimize the barrier to entry / integration into existing courses at other universities.

The rest of the paper is organized as follows: Section 2 provides a background on experiential learning, SCA Attacks, IoT security, and the basics for the course modules that students learn. Section 3 describes each of the course modules, student tasks, and expected learning outcomes. Section 4 provides key takeaways from a pilot offering of the course in Fall 2021. Finally, Section 5 concludes with future directions for the research.

2 Background

In this section, we provide a brief overview of experiential learning, which is central to the curriculum design, and summarize the relevant IoT security topics explored by students in these course modules.

2.1 Experiential Learning and Hardware Security

Broadly, experiential learning is the process of "learning by doing". Integrating experiential learning methods into course modules can benefit engineering education by helping students connect theories and knowledge learned during lectures to real world situations through hands-on experiences [8, 11, 16]. Therefore, the theory is *contextualized* in IoT. Most models of experiential learning are based on Kolb's Experiential Learning Theory (ELT) [8], which operates by creating knowledge through experience [**?**]. ELT can be modeled using Kolb's Experiential Learning Cycle, which has the following stages: 1) active experimentation, concrete experience, reflective observation, and abstract conceptualization, after which the steps are repeated until students attain the desired learning outcome.

These practices allow students to gain experience from real-world examples and develop many skills, including critical thinking, research, meta-cognitive thinking, epistemic cognition, scientific inquiry, engineering innovation and problem solving [**?**, 8]. Because hardware security vulnerabilities and their exploits

are inherently complex – requiring background knowledge in areas such as cryptography, statistics, and electrical circuits – experiential learning is well suited to training students in IoT security [11]. A mostly theory-based course covering IoT security is well suited to discussing topics such as supply chain security, testing and verification, and intellectual property piracy. Students can mount attacks using pre-recorded data, but without access to the original hardware, they cannot *actively experiment* with different implementations and countermeasures, reacquire data, form hypotheses, or reflect on their concrete experiences after the experiment. Putting the experiment itself in context – in this case, IoT – is also important to motivate students and connect the experiment to the real world. In summary, access to a hardware platform is essential to perform these experiments.

2.2 Side-Channel Analysis (SCA) Attacks

As computer hardware devices perform various computations, their physical properties can be measured and used to identify what *functions* are being performed and even the contents of the underlying *data* [22]. These physical properties are referred to as side-channel leakages, and come in many forms. For example, as transistors switch on and off, the effects can be seen on the power consumption on the chip, as well as in the electromagnetic and thermal emanations measured externally.

Simple Power Analysis (SPA) Attacks An SPA attack involves visual inspection of the power waveforms measured from the device during operation. This can be accomplished by looking at peaks within the waveform at specific locations. For example, a basic "if statement" may produce a higher spike when its condition evaluates as `true`. Moreover, the time it takes to perform certain operations can be used as a way to determine what the inputs are to the algorithm [13].

Differential Power Analysis (DPA) Attacks A DPA attack can exploit *data-dependent leakages*, where the power consumption of a crucial operation is dependent on the data inputs [12]. Essentially, the idea of DPA is that small differences in power consumption can be measured for the same operation performed across different inputs – one where a targeted bit in the result is a 1, and the other a 0. This difference can be exploited to deduce the secret key during operation.

Template-Matching Attack To process more complex power traces, a sum of absolute differences (SAD) template-matching approach may be used. If an algorithm is repeating the same instructions, the power is expected to match very closely between iterations (assuming that there are no countermeasures in place). The attacker begins by defining a *template* signal with the region of interest in

Algorithm 1: Naïve Password Checker v1.0

```
Input: input_pw: string with password given by the user to check
Output: Indicate access granted or not based on the input_pw
1  secret_pw ← "RealPassword"; /* Real password stored in the system */
2  wrong_pw ← False;
3  for i ← 0 to Size(secret_pw)–1 do
4      if secret_pw[i] != input_pw[i] then
5          wrong_pw ← True;
6          break;
7
8  if wrong_pw then
9      Print("INCORRECT PASSWORD!");
10 else
11     Print("ACCESS GRANTED!");
```

the power trace. Then, sweep this template along every point in the signal under attack, subtracting the two, taking the absolute value, and adding the result. If the two regions match very closely, then its SAD output is close to 0. Otherwise, the output will be higher [6]. In contrast to SPA, SAD template-matching allows one to identify similarities between power traces within a moving window instead of a sample-wise comparison (i.e., one power sample at a time).

2.3 IoT Security

Generally, an IoT system consists of edge sensors that send data to a central unit for processing. The central unit uses software applications to process the collected data for intelligent decision making. As an edge node cannot be physically protected or continuously monitored, it can be easily attacked [10]. For this reason, edge sensors often rely on password checkers for user credentials and access verification, SKE cryptosystems (e.g., AES) for secure communication and data transfer, and PKE cryptosystems (e.g., RSA) for secure connections and distribution of SKE keys. The attacker can gain access to the edge node and interfere with the legitimate operation. For example, they can modify the transmitted data. Typically, the transmitted data is encrypted. In these course modules, students observe first-hand how vulnerable unprotected systems are.

Password Checkers: Periodically, edge nodes may be accessed by users and/or administrators. However, a naïve password checker implementation may be exploited by an attacker through SCA. For example, a simple power analysis (SPA), such as timing analysis, can be used to break a basic loop design in which the input and secret password are compared character-by-character, and then break as soon as it finds an incorrect character (Algorithm 1). The attacker can simply brute force each character at a time instead of the whole password at once.

Algorithm 2: Naïve Password Checker v2.0 (w/Random Delay)

```
   Input: input_pw: string with password given by the user to check
   Output: Indicate access granted or not based on the input_pw
 1 secret_pw ← "RealPassword"; /* Real password stored in the system */
 2 wrong_pw ← False;
 3 for i ← 0 to Size(secret_pw)–1 do
 4 |   if secret_pw[i] != input_pw[i] then
 5 |   |   wrong_pw ← True;
 6 |   |   break;
 7
 8 if wrong_pw then
 9 |   /* Countermeasure Attempt: insert a random delay            */
10 |   wait ← Rand() %12345;
11 |   for delay ← 0 to wait do
12 |   |   ; /* Do Nothing */
13 |   Print("INCORRECT PASSWORD!");
14 else
15 |   Print("ACCESS GRANTED!");
```

This reduces the complexity from $O(n^m)$ to $O(n * m)$, where n is the number of possible valid characters that the password may have and m is the number of characters in the secret password. Therefore, the designer should implement the respective countermeasures to prevent similar security vulnerabilities.

However, identifying the appropriate countermeasure requires an understanding of the trade-offs and threat model, including the access and capabilities of the attacker. The countermeasure needs to be simple enough to not produce high power, performance, and resources overheads, but not too unsophisticated that it does not protect the system as desired. For instance, a random delay could be inserted before letting the user know the password in an attempt to confuse the attacker (Algorithm 2). However, the attacker can still identify relevant power spikes (e.g., at the break statement) before the random delay begins. Thus, designers need to be more vigilant to truly protect their system.

SKE and AES: Once the users have been verified safely and allowed access to their systems, they may need to communicate and transfer confidential information between them. However, since this access is typically facilitated through public infrastructure, we need to encrypt the data to prevent data theft or manipulation. Hence, a symmetric key encryption (SKE) such as AES is used to protect the content transferred. Just as before, for a given threat model, making certain assumptions about the knowledge and capabilities of the attacker, the implementation of the encryption may be broken, leading to key recovery [9,20]. Attacks such as differential or correlation power analysis (DPA or CPA) have been successfully used against implementations of AES and other SKEs [3,17].

Algorithm 3: Square-and-Multiply (SAM)

Input: b: base
Input: m: modulo
Input: exp_bin: exponent represented as an array of n bits
Output: r: result from the modular exponentiation

```
1 r ← b;
2 i = n − 1;
3 while i > 0 do
4 |   r ← (r ∗ r) mod m;
5 |   if exp_bin[--i] == 1 then
6 |   |   r ← (r ∗ b) mod m;
7 return r;
```

For example, this issue may be present in the AES algorithm during the substitution bytes (S-box) step, as we will demonstrate later in Section 3.3.

PKE and RSA: On the other hand, even if the SKE implementation has been adequately hardened, these depend on a secret key that both edge nodes need beforehand through a secure channel. However, these systems or users may be too far away for them to transfer the key offline. Moreover, an SKE system becomes less and less secure as more users require access (and the secret key). Therefore, we also need to encrypt and communicate keys through efficient, reliable, and secure mechanisms. Otherwise, an eavesdropper can decrypt the ciphertext generated by the SKE, defeating any countermeasure implemented. Currently, most IoTs uses PKEs to safely authenticate, generate, and distribute the secret key for the respective SKE.

An example of PKE that students can learn is RSA. Mathematically, the security of RSA is derived from the computational difficulty of factoring the product of two large prime numbers. In a naïve implementation, an attacker can exploit the fact that certain operations depend on the present value of a key bit and can be observed from side channels [4]. Encrypting a plaintext p requires first representing it as an integer, then computing the ciphertext as $c = p^e \bmod m$, where e is a public key exponent, and n is a public key modulo. To decrypt, the plaintext is computed as $p = c^d \bmod m$, where $d = e^{-1} \bmod \phi(m)$ and is kept secret. The value of $\phi(m)$ (Euler's totient function) depends on two large prime factors of m, which is difficult to extract.

Alternatively, the secret key d may be detected using power SCA. Since the RSA decryption involves modular exponentiation, which is commonly performed using the square-and-multiply (SAM) algorithm (Algorithm 3) for resource-constrained microcontrollers. This algorithm takes as arguments the base b, modulus N, and exponent in binary exp_bin – which for each bit of the exponent, performs one or two operations. To begin with, the base b is copied into a temporary variable r. If the current key bit is 0, $r = (r * r) \bmod N$, and the loop iterates to the next key bit. If the current key bit is 1, the same function

$r = (r * r) \bmod N$ is performed, but then it is multiplied once more by the original base value, $r = (r * b) \bmod N$. Hence, the number of operations inside the loop depends on the current key bit. The attacker can exploit this factor to effectively read the secret key d from the power trace.

3 Description of Course Modules

In this section, we explain the general setups for the experiments and a description for each of the course modules provided to the students, including their tasks and expected learning outcomes.

3.1 Hardware and Software Setup

The students are provided with a ChipWhisperer (CW) Nano [19]. This board is small (about 60 x 30 x 3 mm) and powered by a micro-USB cable. The CW Nano has two onboard processors, one that is controlled through a Python API and Jupyter Notebooks, and a second *victim* microcontroller programmed in C.

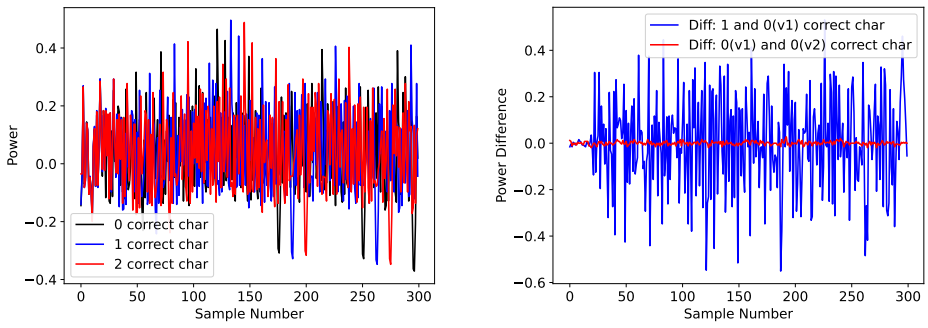
The CW software is open-source and deployed as a virtual machine (VM) image, which greatly simplifies deployment on students' systems. The Python-controlled CW acquisition board can be used as a kind of USB oscilloscope which records the power consumed by the *victim* in real-time while it executes various functions. The *victim* can be controlled using a serial protocol, which allows students to write, flash, and execute different firmware on the *victim*. Simultaneously, they can record, observe, analyze, and plot the results in real time from within the Jupyter Notebook environment.

3.2 Password Checker Module

Module Description: In this course module, teams of students attack two implementations of example naïve password checkers (Algorithms 1-2). Students are given a training password to test, practice, and become familiar with ChipWhisperer and password checkers. Once the students gain sufficient experience and implement a plausible attack, they will repeat the procedure with a secret password. Students learn to perform SPA, perform power timing analysis attacks on relevant spikes, and improve countermeasures.

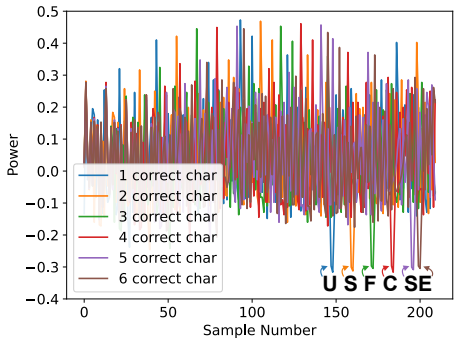
Student Tasks: Students must complete the following tasks in this module:

1. Implement the training Password Checkers v1.0 and v2.0 (Algorithms 1-2) in C and flash the compiled binary file to the *victim*.
2. Send sample input passwords ($input\_pw$) to the *victim* with a varying number of correct characters and collect power traces.
3. Plot and analyze the average power trace (P_{avg}) as in Fig. 1(a).
4. Plot and analyze the power difference (P_{diff}) for the input passwords $input\_pw$ with a varying number of correct characters as in Fig. 1(b).



(a) (b)

Fig. 1. Password Checker Module: Plots of example (a) average power traces for password inputs with 0,1,2 correct characters; and (b) power difference between 0 vs 0 correct (red) and 0 vs 1 correct (blue) characters.



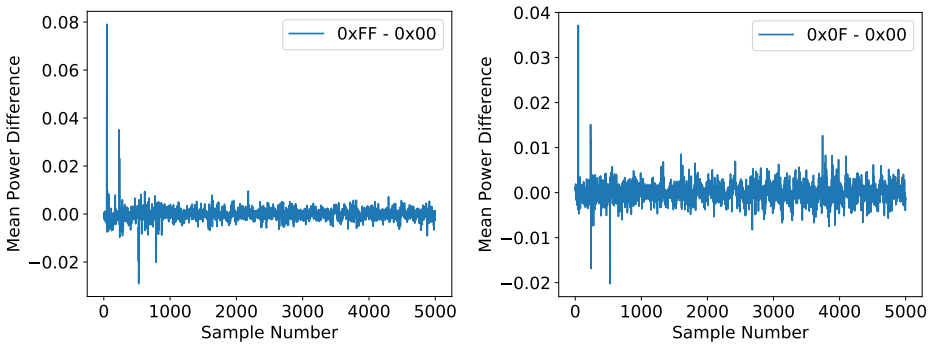
(a)

```
Char Found: U
Char Found: S
Char Found: F
Char Found: C
Char Found: S
Char Found: E
Password: USFCSE
Confirm USFCSE: ACCESS GRANTED
```

(b)

Fig. 2. Password Checker Module: (a) Plot of example average power traces for respective correct character found during the SPA attack. (b) Example confirmation and attack output when finding the password for the training Naïve Password Checkers v1.0 and v2.0.

5. Identify a relevant spike in P_{diff} where it could indicate the end of the loop or break statement.
6. Implement an SPA attack to find a single correct character. If given a password starting with i correct characters, then the algorithm should return the $i+1$ correct character and plot the respective power trace as in Fig. 2(a).
7. Iterate the single-character SPA attack to find all the characters in the real secret password.
8. Verify the recovered password (Fig. 2(b)).
9. Repeat these procedures for a secret password ($secret\_pw$) stored in a given binary firmware to flash to the *victim*.
10. Once successful, implement in C a working countermeasure on the password checker v1.0 (Algorithm 1) that will defeat the SPA attack implemented



(a) (b)

Fig. 3. AES Module (Part A): Plots of example mean power difference at byte 0 between AES inputs (a) 0xFF vs 0x00; and (b) 0x0F vs 0x00.

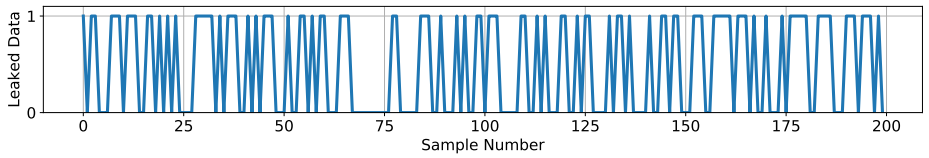


Fig. 4. AES Module (Part B): LSB output data from random sample inputs into an AES S-Box model.

in previous steps. Flash the compiled binary file to the *victim* and test the implemented SPA attack. Verify that the countermeasure thwarts the attack.

11. Report the findings, discuss the results, and draw meaningful conclusions.

Expected Learning Outcomes: By the end of the experiment, students will become familiar with basic SCA, SPA, timing attacks, and respective countermeasures in password checkers. They will gain a better understanding on the related topics and importance through their experiences they gathered through this module. This module should give them strong foundations for the following modules on AES and RSA (Sections 3.3-3.4).

3.3 AES Module

Module Description: In this course module, students build on their knowledge of SPA and experiment with more sophisticated SCA attacks on an implementation of AES. In particular, the students investigate the usage of DPA and CPA on the AES S-box to gain an understanding of how the attacks work. The students work through three Jupyter Notebooks (A-C) that have been modified from existing experiments provided in the CW software suite [19].

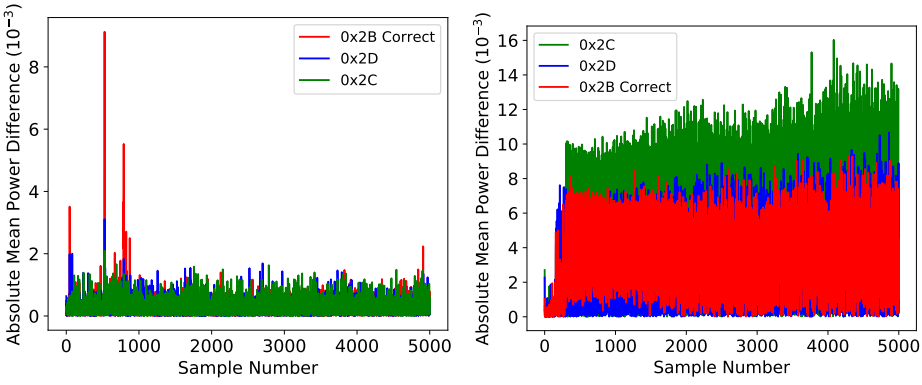


Fig. 5. AES Module (Part C): Plots of example absolute mean power difference between 0/1 AES S-Box LSB outputs for different key guesses on (a) the original AES; and (b) the AES with additional internal random delay countermeasure.

Student Tasks: Students must complete the following tasks in this module:

1. Program the CW with the AES implementation from the provided library.
2. Work through notebook A to capture many traces at a time for two inputs - 0xFF and 0x00.
3. Divide the traces into two groups, one for each input
4. Average the traces in each group to filter out noise from the measurements.
5. Compute the differential between the two traces, plot using matplotlib and inspect the peak as in Fig. 3(a).
6. Repeat the same process but with inputs that have a HW difference of 4 (e.g., 0x0F and 0x00) and plot as in Fig. 3(b).
7. Work through notebook B to conceptualize the process of recovering a byte of the AES key.
8. Complete the steps to build the foundation of the DPA attack on AES.
9. Plot the LSB of the results from random sample inputs to the AES S-Box as in Fig. 4.
10. Work through notebook C to apply the attack from B to an actual hardware implementation of AES.
11. Automate the attack over the entire AES key, plot a few example CPA outputs as illustrated in Fig. 5(a), and compare the recovered results from the hardware to the actual key as shown in Table 1.
12. Modify the provided implementation of the AES in C code to include a random delay as a countermeasure to the attack.
13. Repeat the attack above, plot a few example CPA outputs as shown in Fig. 5(b), and compare the recovered key with the actual key, the attack should be thwarted by the countermeasure as given in Table 2.
14. Report the findings, discuss the results, and draw meaningful conclusions.

Table 1. AES Module (Part C): Example of key recovered output and actual key values for the original AES.

Key	Hex Bytes	Accuracy
Recovery Key	[2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C]	16/16 Bytes
Actual Key	[2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C]	(100%)

Table 2. AES Module (Part C): Example of key recovered output and actual key values for the AES with additional internal random delay countermeasure.

Key	Hex Bytes	Accuracy
Recovery Key	[B3,A2,90,79,46,EF,9B,23,FA,5A,59,B7,7C,4D,A4,D2]	0/16 Bytes
Actual Key	[2B,7E,15,16,28,AE,D2,A6,AB,F7,15,88,09,CF,4F,3C]	(0%)

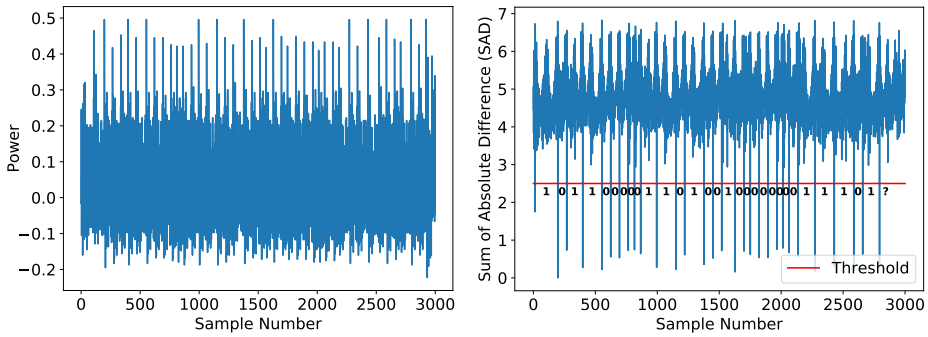
Expected Learning Outcomes: Through this experiment, the students gain an understanding of the basics of the AES algorithm, and the S-box in particular. They observe the leakage of the S-box, and learn to exploit it by using classic DPA. Finally, they use this knowledge to mount the attack and reveal the whole key.

3.4 RSA Module

Module Description: In this course module, teams of students attack an implementation of the square-and-multiply (SAM) algorithm – a key component of RSA that enables modular exponentiation – on a resource-constrained microcontroller development board. Students are encouraged to apply critical thinking to identify vulnerabilities in the design to extract the secret key through SCA attacks. The module has a tutorial that introduces the topic and basic attack setup, enabling students to familiarize themselves with the attack. In particular, students learn about improving Signal-to-Noise Ratio (SNR), template-matching attacks such as the sum of absolute differences (SAD), and their countermeasures.

Student Tasks: Students must complete the following tasks in this module:

1. Implement SAM in C and flash the compiled binary file to the *victim*.
2. Send sample exponent keys to the *victim* and collect power traces.
3. Plot and analyze the average power trace (P_{avg}) as in Fig. 6(a).
4. Identify a suitable portion of P_{avg} to serve as the template T for SAD.
5. Implement SAD to compare T against P_{avg} and plot results as in Fig. 6(b).
6. Determine a threshold for the SAM results to discern matches (values under the threshold) as in Fig. 6(b) and collect this in a binary match vector M_{bin}.
7. Examine the spacing between each match and determine when the *victim* is processing 1 or 0 key-bit, a larger space indicates a 1 key-bit (Fig. 6(b)).



(a) (b)

Fig. 6. RSA Module: Plots of example (a) average power trace and (b) SAD algorithm output.

Table 3. RSA Module: Example key recovered output and actual key values from Fig. 6(b).

Key	Hex	Binary	Accuracy
Recovery Key	6C1A403A	01101100000110100100000000111101**?**	31/32 bits
Actual Key	6C1A403B	01101100000110100100000000111011	(96.9%)

8. Read off the key values from the plot and automate this process using M_{bin}.
9. Compare the recovered and actual keys (Table 3). Note that the SAD attack is unable to get the LSB because there is no next match to compare to.
10. Report the findings, discuss the results, and draw meaningful conclusions.

Expected Learning Outcomes: By the end of the experiment, the students will become familiar with complex SCA, template-matching attacks using SAD on a SAM algorithm. The student will learn how the SAM algorithm works and its potential security vulnerabilities on RSA for IoT devices. This module will close many gaps in the students' understanding on SCA and the importance of hardware security design on IoT through hands-on experiences.

4 Results

Feedback from students in the Fall 2021 course was generally positive, based on interactions with students during the semester. Some critical feedback of the course stemmed from a lack of relevant and accessible resources for Python development. Many of the students felt it was difficult to translate the high level topics and requirements of the modules to producible Python code. To improve upon this, we have developed more supplementary material to teach students

relevant techniques in Python for analyzing raw binary, using data processing tools such as Numpy and Matplotlib, interfacing with hardware, and reading official documentation.

The students also fulfilled the expected learning outcomes outlined in the prior section. The password checker module taught the students about the basics of SCA and how different results from functions can cause measurable spikes in power. The AES module reinforced their understanding of these concepts by first having them compare the power consumption of AES with different inputs. Then, the module guided them through preliminary steps for the attack to teach them the unfamiliar concepts of DPA. After these two module, the students were much more familiar with the CW platform and had a strong basis to work from for the following RSA module. In the RSA module, the students were provided with less resources to start from than in the previous labs, so they had to rely on the lessons and concepts learned earlier. Most students were able to derive the correct RSA key, and analyses in their reports demonstrated a link between their conceptual understanding of the attack and the successful outcomes of their experiments. This was a common theme across all the experiments.

5 Conclusion

In this paper, we presented a comprehensive set of course modules that provide students with hands-on, experiential learning with IoT security experiments. It uses off-the-shelf, low-cost development boards and open-source tools to make integration into existing university courses feasible. These modules teach students about SCA attacks to identify and exploit vulnerabilities in IoT devices for simple password checkers through SPA timing attacks, key recovery for AES encryption using DPA, and in RSA during the SAM algorithm using SAD template-matching attacks. Students learn the motivation behind SKE and PKE, the mathematics of SAM and RSA, and gain an understanding of how naïve implementations may be vulnerable to SCA attacks. Through this, students gain real-world experience and develop critical thinking, research, and engineering problem solving skills. Furthermore, we will consider future incremental improvements and other practical projects based on feedback from students.

References

1. Abdullah, A., Hamad, R., Abdulrahman, M., Moala, H., Elkhediri, S.: CyberSecurity: a review of internet of things (IoT) security issues, challenges and techniques. In: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS). pp. 1–6. IEEE (2019)
2. Al-Emran, M., Malik, S.I., Al-Kabi, M.N.: A Survey of Internet of Things (IoT) in Education: Opportunities and Challenges, pp. 197–209. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-24513-9_12
3. Albiol, P., Manich, S., Arumí, D., Rodríguez-Montañés, R., Gómez-Pau, A.: Low Cost AES Protection Against DPA Using Rolling Codes. In: 2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS). pp. 1–6 (2021). https://doi.org/10.1109/DCIS53048.2021.9666192

4. Aljuffri, A., Reinbrecht, C., Hamdioui, S., Taouil, M.: Multi-Bit Blinding: A Countermeasure for RSA Against Side Channel Attacks. In: IEEE 39th VLSI Test Symp. (VTS). pp. 1–6 (2021). https://doi.org/10.1109/VTS50974.2021.9441035
5. Arias, O., Wurm, J., Hoang, K., Jin, Y.: Privacy and Security in Internet of Things and Wearable Devices. IEEE Trans. Multi-Scale Comput. Syst. **1**(2), 99–109 (2015). https://doi.org/10.1109/TMSCS.2015.2498605
6. Atallah, M.: Faster image template matching in the sum of the absolute value of differences measure. IEEE Transactions on Image Processing **10**(4), 659–663 (2001). https://doi.org/10.1109/83.913600
7. Baker, S.B., Xiang, W., Atkinson, I.: Internet of Things for Smart Healthcare: Technologies, Challenges, and Opportunities. IEEE Access **5**, 26521–26544 (2017). https://doi.org/10.1109/ACCESS.2017.2775180
8. Desai, P., Bhandiwad, A., Shettar, A.S.: Impact of Experiential Learning on Students' Success in Undergraduate Engineering. In: 2018 IEEE 18th Conf. Adv. Learn. Technol. (ICALT). pp. 46–50 (2018). https://doi.org/10.1109/ICALT.2018.00018
9. Dinesh Kumar, S., Thapliyal, H., Mohammad, A.: FinSAL: FinFET-Based Secure Adiabatic Logic for Energy-Efficient and DPA Resistant IoT Devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(1), 110–122 (2018). https://doi.org/10.1109/TCAD.2017.2685588
10. Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., Sikdar, B.: A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures. IEEE Access **7**, 82721–82743 (2019). https://doi.org/10.1109/ACCESS.2019.2924045
11. Kaneko, K., Ban, Y., Okamura, K.: A Study on Effective Instructional Design for IoT Security Education Focusing on Experiential Learning. International Journal of Learning Technologies and Learning Environments **2**(1), 1–18 (2019)
12. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. p. 388–397. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
13. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
14. Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., Zhao, W.: A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. IEEE Internet of Things Journal **4**(5), 1125–1142 (2017). https://doi.org/10.1109/JIOT.2017.2683200
15. Lv, Z., Qiao, L., Kumar Singh, A., Wang, Q.: AI-Empowered IoT Security for Smart Cities. ACM Trans. Internet Technol. **21**(4) (jul 2021). https://doi.org/10.1145/3406115
16. Mantwill, F., Multhauf, V.: Application of Agile Experiential Learning Based on Reverse Engineering as Support in Product Development, pp. 65–81. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-78368-6_4
17. Mazumdar, B., Mukhopadhyay, D.: Construction of Rotation Symmetric S-Boxes with High Nonlinearity and Improved DPA Resistivity. IEEE Transactions on Computers **66**(1), 59–72 (2017). https://doi.org/10.1109/TC.2016.2569410

18. Mohamad Noor, M.b., Hassan, W.H.: Current research on Internet of Things (IoT) security: A survey. Computer Networks **148**, 283–294 (2019). https://doi.org/10.1016/j.comnet.2018.11.025
19. O'Flynn, C., Chen, Z.D.: ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In: Prouff, E. (ed.) Constructive Side-Channel Analysis and Secure Design. pp. 243–260. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10175-0_17
20. Ramesh, C., Patil, S.B., Dhanuskodi, S.N., Provelengios, G., Pillement, S., Holcomb, D., Tessier, R.: FPGA Side Channel Attacks without Physical Access. In: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). pp. 45–52 (2018). https://doi.org/10.1109/FCCM.2018.00016
21. Ramlowat, D.D., Pattanayak, B.K.: Exploring the Internet of Things (IoT) in Education: A Review. In: Satapathy, S.C., Bhateja, V., Somanah, R., Yang, X.S., Senkerik, R. (eds.) Information Systems Design and Intelligent Applications. pp. 245–255. Springer, Singapore (2019)
22. Rostami, M., Koushanfar, F., Karri, R.: A Primer on Hardware Security: Models, Methods, and Metrics. Proceedings of the IEEE **102**(8), 1283–1295 (Aug 2014). https://doi.org/10.1109/JPROC.2014.2335155
23. Sarker, I.H., Khan, A.I., Abushark, Y.B., Alsolami, F.: Internet of things (iot) security intelligence: a comprehensive overview, machine learning solutions and research directions. Mobile Networks and Applications pp. 1–17 (2022)
24. Sidhu, S., Mohd, B.J., Hayajneh, T.: Hardware Security in IoT Devices with Emphasis on Hardware Trojans. Journal of Sensor and Actuator Networks **8**(3) (2019). https://doi.org/10.3390/jsan8030042
25. Xu, T., Wendt, J.B., Potkonjak, M.: Security of IoT systems: Design challenges and opportunities. In: 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). p. 417–423 (Nov 2014). https://doi.org/10.1109/ICCAD.2014.7001385