IoT Goes Nuclear: Creating a ZigBee Chain Reaction

Eyal Ronen(✉)*, Colin O'Flynn†, Adi Shamir* and Achi-Or Weingarten*
PRELIMINARY DRAFT, VERSION 0.93
*Weizmann Institute of Science, Rehovot, Israel
{eyal.ronen,adi.shamir}@weizmann.ac.il
†Dalhousie University, Halifax, Canada
coflynn@dal.ca

***Abstract*—Within the next few years, billions of IoT devices will densely populate our cities. In this paper we describe a new type of threat in which adjacent IoT devices will infect each other with a worm that will spread explosively over large areas in a kind of nuclear chain reaction, provided that the density of compatible IoT devices exceeds a certain critical mass. In particular, we developed and verified such an infection using the popular Philips Hue smart lamps as a platform. The worm spreads by jumping directly from one lamp to its neighbors, using only their built-in ZigBee wireless connectivity and their physical proximity. The attack can start by plugging in a single infected bulb anywhere in the city, and then catastrophically spread everywhere within minutes. It enables the attacker to turn all the city lights on or off, permanently brick them, or exploit them in a massive DDOS attack. To demonstrate the risks involved, we use results from percolation theory to estimate the critical mass of installed devices for a typical city such as Paris whose area is about 105 square kilometers: The chain reaction will fizzle if there are fewer than about 15,000 randomly located smart lights in the whole city, but will spread everywhere when the number exceeds this critical mass (which had almost certainly been surpassed already).**

To make such an attack possible, we had to find a way to remotely yank already installed lamps from their current networks, and to perform over-the-air firmware updates. We overcame the first problem by discovering and exploiting a major bug in the implementation of the Touchlink part of the ZigBee Light Link protocol, which is supposed to stop such attempts with a proximity test. To solve the second problem, we developed a new version of a side channel attack to extract the global AES-CCM key (for each device type) that Philips uses to encrypt and authenticate new firmware. We used only readily available equipment costing a few hundred dollars, and managed to find this key without seeing any actual updates. This demonstrates once again how difficult it is to get security right even for a large company that uses standard cryptographic techniques to protect a major product.

1. Introduction

The Internet of Things (IoT) is currently going through exponential growth, and some experts estimate that within the next five years more than fifty billion "things" will be connected to the internet. Most of them will be cheaply made sensors and actuators which are likely to be very insecure. The potential dangers of the proliferation of vulnerable IoT devices had just been demonstrated by the massive distributed denial of service (DDoS) attack on the Dyn DNS company, which exploited well known attack vectors such as default passwords and the outdated TELNET service to take control of millions of web cameras made by a single Chinese manufacturer [1].

In this paper we describe a much more worrying situation: We show that without giving it much thought, we are going to populate our homes, offices, and neighborhoods with a dense network of billions of tiny transmitters and receivers that have ad-hoc networking capabilities. These IoT devices can directly talk to each other, creating a new unintended communication medium that completely bypasses the traditional forms of communication such as telephony and the internet. What we demonstrate in this paper is that even IoT devices made by big companies with deep knowledge of security, which are protected by industry-standard cryptographic techniques, can be misused by hackers to create a new kind of attack: By using this new communication medium to spread infectious malware from one IoT device to all its physically adjacent neighbors in a process resembling a nuclear chain reaction, hackers can rapidly cause city-wide disruptions which are very difficult to stop and to investigate.

We focus in this paper on the popular Philips Hue smart lights which had been sold (especially in the European market) in large numbers since 2012. The communication between the lamps and their controllers is carried out by the Zigbee protocol, which is the radio link of choice between many IoT devices due to its simplicity, wide availability, low cost, low power consumption, robustness, and long range (its main disadvantage compared to WiFi radio communication is its limited bandwidth, which is not a real problem in most IoT applications). The Hue lamps contain a ZigBee chip

made by Atmel, which uses multiple layers of cryptographic and non-cryptographic protection to prevent hackers from misusing the lamps once they are securely connected with their controllers. In particular, they will ignore any request to reset or to change their affiliation unless it is sent from a ZigBee transmitter which is only a few centimeters away from the lamp. Even though the attacker can try to spoof such a proximity test by using very high power transmitters, the fact that the received power decreases quadratically with the distance makes such brute force attacks very hard (even at ranges of a hundred meters). This requires high power dedicated equipment and cannot be done with the standard ZigBee off the shelf equipment.

Our initial discovery was that the Atmel stack has a major bug in its proximity test, which enables any standard ZigBee transmitter (which can be bought for a few dollars in the form of an tiny evaluation board) to initiate a factory reset procedure which will dissociate lamps from their current controllers, up to a range of 400 meters. Once this is achieved, the transmitter can issue additional instructions which will take full control of all those lamps. We demonstrated this with a real war-driving experiment in which we drove around our university campus and took full control of all the Hue smart lights installed in buildings along the car's path. Due to the small size, low weight, and minimal power consumption of the required equipment, and the fact that the attack can be automated, we managed to tie a fully autonomous attack kit below a standard drone, and performed war-flying in which we flew hundreds of meters away from office buildings, forcing all the Hue lights installed in them to disconnect from their current controllers and to blink SOS in morse code.

By flying such a drone in a zig-zag pattern high over a city, an attacker can disable all the Philips Hue smart lights in city centers within a few minutes. Even though such an attack can have very unpleasant consequences, its effects are only temporary since they can be reversed by the tedious process of bringing each lamp to within a few centimeters from its legitimate controller and reassociating them.

To test whether the attacker can cause more permanent damage, we used a combination of known and novel techniques in order to reverse engineer the process chosen by Philips in order to make a lamp firmware update possible. We did it only by using updates for older models lights as reference, without ever seeing any actual firmware update issued by Philips for the new Atmel chip. We discovered that in order to be accepted by the lamp, each firmware update had to be both encrypted and authenticated by a state of the art AES - Counter with CBC-MAC (CCM) cryptographic algorithm; however, all the lamps (at least from the same product type) use the same global key. We managed to deduce all the secret cryptographic elements used by Philips (such as IV and key) within a few days, using novel side channel attacks that used only cheap and easily obtained equipment costing a few hundred dollars, and without physically extracting them from their secure memory. Once we obtained these secret values, we could create any new firmware and upload it into any Philips Hue lamp.

We have thus demonstrated that a really devastating low-budget attack can be mounted on this IoT system: A single infected lamp with a modified firmware which is plugged-in anywhere in the city can start an explosive chain reaction in which each lamp will infect and replace the firmware in all its neighbors within a range of up to a few hundred meters. This is similar to the worm scenario which was accidentally triggered by Robert Morris Jr., and brought the whole internet to a standstill within minutes in 1988. To avoid any similar accidental outcome due to our experiments, the only change we actually made in the new firmware we installed in the infected lamps was to change the firmware version number string to `IrradiateHue`, which could not possibly cause any harm. However, a real attacker could permanently brick all the infected lamps by simply disabling their firmware update process. Such lamps cannot be rescued, and have to be thrown away.

Our new attack differs from previous attacks on IoT systems in several crucial ways. First of all, previous attacks used TCP/IP packets to scan the internet for vulnerable IoT devices and to force them to participate in internet-based activities such as a massive DDOS attack. Since internet communication is heavily monitored and can be protected by a large variety of security tools, such attacks can be discovered and stopped at an early stage, at least in principle. Our attack does not use any internet communication at all, and the infections jump directly from lamp to lamp using only unmonitored and unprotected ZigBee communication. Consequently, it will be very difficult to detect that an attack is taking place and to locate its source after the whole lighting system is disabled. Another major difference is that our attack spreads via physical proximity alone, disregarding the established networking structures of lamps and controllers. As a result, such an attack cannot be stopped by isolating various subnetworks from each other, as system administrators often do when they are under attack. In this sense the attack is similar to air-borne biological infections such as influenza, which spread almost exclusively via physical proximity. Finally, previously reported attacks are carried out via linear scans and infections which are all carried out in a star-shaped structure with a centrally located attacker, whereas our chain reaction attack spreads much faster by making each infected lamp the new source of infection for all its adjacent lamps; the attacker only has to initiate the infecting with a single bad lamp, and can then retire and watch the whole city going dark automatically.

The paper is organized as follows: the remainder of the introduction will first discuss how widely the worm could spread in the RF environment, summarize the related work on IoT lightbulb attacks, and present our attack. Section 2 provides the necessary background for the attack: first we will describe the ZigBee Light Link (ZLL) standard which is used by the Philips Hue bulbs, and describe the features we will target in this attack such as the commissioning and upgrade process. In Section 2 we will also summarize the cryptographic primitives and the side-channel attacks we will be using against them. Section 3 will describe

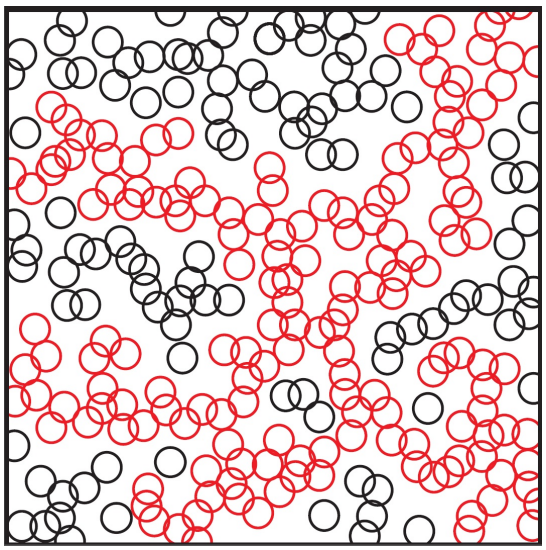

Figure 1. 2D continuum percolation with disks

the physical hardware setup, along with the results of a preliminary power analysis attack against the hardware. Sections 4, 5, and 6 are the technical details of the attack. Section 4 provides details of the upgrade process and how it relates to spreading a worm, assuming that the correct encryption keys could be found to disseminate a valid encrypted and signed firmware file. Section 5 uses power analysis to recover the encryption keys (and other associated secret information such as the I.V.), which our novel attack against AES-CCM that requires only approximately one to two times the effort of breaking AES-ECB. With encryption keys known, in Section 6 we solve the problem of taking control of a device at very long ranges, which allows us to feed a device the 'updated' firmware image which we have encrypted & signed with the recovered keys. Finally section 7 will provide some examples of what this worm could accomplish, before concluding the paper in Section 8.

1.1. Estimating the critical mass

Consider a city whose area is A, and assume that its shape is roughly circular (i.e., it is flat, convex, not too elongated, and without holes). We place N smart lights at random locations within the city, and define an infection graph by connecting any two lights whose distance is smaller than D by an edge. The connected components in this graph define the possible infection patterns which can be started by plugging in a single infected light. For a small N all the components are likely to consist of just a few vertices, but as N increases, the graph goes through a sudden phase change in which a single giant connected component (that contains most of the vertices) is created. This is the critical mass at which the infection is likely to spread everywhere in the city instead of remaining isolated in a small neighborhood.

The mathematical field dealing with such problems is called *Percolation Theory*, and the critical N is called the *Percolation Threshold*. A good survey of many variants of this problem can be found in [2], and the particular version we are interested in appears in the section on thresholds for 2D continuum models, which deals with long range connectivity via overlapping two dimensional disks of radius R, as described in Fig 1. Since two points are within a distance D from each other if and only if the two disks of radius $R = D/2$ around them intersect, we can directly use that model to find the critical mass in our model: It is the value N for which the total area of all the randomly placed disks (i.e., $\pi R^2 N$) is about 1.128 times larger than the total area A of the city. In other words, $N = 1.128A/\pi(D/2)^2$.

To get a feeling for how large this N can be, consider a typical city like Paris, which is fairly flat, circular in shape, and with few skyscrapers that can block the available lines of sight. Its total area is about 105 square kilometers [4]. According to the official ZigBee Light Link website [16], the range of ZigBee communication is between 70 meters indoors and 400 meters outdoors [1]. There is probably no single number that works in all situations, but to estimate N it is reasonable to assume that one lamp can infect other lamps if they are within a distance of $D = 100$ meters, and thus the disks we draw around each lamp has a radius of $R = 50$ meters. By plugging in these values into the formula, we get that the critical mass of installed lights in the whole city of Paris is only about $N = 15,000$. Since the Philips Hue smart lights are very popular in Europe and especially in affluent areas such as Paris, there is a very good chance that this threshold had in fact been exceeded, and thus the city is already vulnerable to massive infections via the ZigBee chain reaction described in this paper.

1.2. Related Work

In recent years numerous works on the security of IoT devices and protocols were published. Regarding connected lights, several vulnerabilities were discovered. Alex Chapman [5] managed to extract hard coded encryption keys used to encrypt data sent between LIFX brand light bulbs. From this he recovered the Wi-Fi password of the local network that was sent between the bulbs. Dhanjani [6] had shown DoS (denial of service) attacks against Philips Hue. Ronen and Shamir [7] have shown how to use the Philips Hue and LimitlessLed systems to create a covert channel to exfiltrate data from air-gapped networks, and to create strobes that can cause epileptic seizures. Heiland [8] found weaknesses in the Osram Lightify app such as unencrypted Wi-Fi passwords, lack of authentication in the gateway and vulnerable usage of ZigBee Home Automation profile. However those vulnerabilities are not related to the ZLL (ZigBee Light Link) protocol discussed in this paper.

There are several works specific to the ZLL protocol and related products. Armknecht et al. [9] proposed a formal

1. The Philips engineers we talk with stated that in a dense urban environment, the effective range can be less than 30 meters

security model. Zillner [10] and Morgner et al. [11] demonstrated weaknesses in ZLL and ways to take over lights. However to be able to take over lights from a distance they had to use custom hardware with much stronger transmission power. O'Flynn [12] reverse engineered some of the Philips Hue security design choices, where he raised the possibility of a lightbulb worm, but did not bypass either the firmware security protection or provide a spreading mechanism.

The first power analysis attacks on Atmel AES hardware accelerators was done by Kizhvatov [13] against the Atmel XMEGA using AES-ECB mode. O'Flynn and Chen [14] used the same leakage model to attack the Atmel MegaRF128RFA1 hardware, and attacked the ZigBee CCM* mode of operation under the assumption of a known nonce. Jaffe [15] had shown an attack on counter mode encryption with unknown nonce, but would require 2^{16} sequential block operations on our hardware with the same nonce while our firmware can have at most 2^{14} traces. Moreover, modifying the method by which the counter updates (using a linear feedback shift register, for example) would present a serious challenge to his attack.

1.3. Our attack

Our attack is much stronger than all of the previously described attacks since it creates the first native and autonomously self spreading ZigBee worm, targeting the Philips Hue light system. It is a combination of two novel attacks:

1) A Correlation Power Analysis (CPA) attack against the CCM mode used to encrypt and verify firmware updates, allowing us to encrypt, sign and upload malicious OTA updates to infect lights.
2) A takeover attack allowing us to take full control over lights from long distances without using custom hardware.

Our novel takeover attack uses a bug in Atmel's implementation of the ZLL Touchlink protocol state machine (used in Philips Hue lights) to take over lights from large distances (up to ZigBee wireless range that can be as far as 70 meters indoors or 400 meters outdoors [16]), using only standard Philips Hue lights. Our attack does not assume any prior knowledge about the attacked lights, and does not even require the knowledge of the ZLL's secret master key. This attack can run simultaneously on all lights within range. As we demonstrate, this attack can be used in wardriving and warflying scenarios.

Our novel CPA attack targets the verification phase of the CCM mode. Our attack has several advantages over previous power analysis attacks against AES-CCM:

1) It does not assume any knowledge about the nonce or IV.
2) It works with any type of counter implementation.
3) It does not require any valid encryption sample.
4) It requires at most the verification of messages as short as 2 blocks.

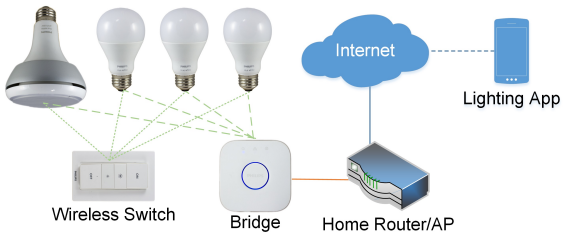

Figure 2. The ZLL architecture.

Figure 3. Philips Hue bridge (gateway), lights, and wireless switch.

5) It requires at most twice the number of traces required to break ECB mode.

We used this novel attack to recover the Philips Hue OTA update verification and encryption keys. With those keys we were able to create a malicious software update and load it to a Philips Hue light.

2. ZLL (ZigBee Light Link) and Smart Light systems

As seen in Figure 2, smart light systems, give users wireless control over lights either from a remote control or from a smart-phone application using a gateway [16]. The gateway is used to bridge the IP world to the ZLL world. ZLL, is an industry standard intended "for interoperable and very easy-to-use consumer lighting and control products. It allows consumers to gain wireless control over all their LED fixtures, light bulbs, timers, remotes and switches" [16], [17]. It is developed and supported by most of the major home lighting manufacturers like Philips, GE and OSRAM.

2.1. The Philips Hue personal wireless lighting system

Philips Lighting had 2015 sales of 7.4 Billion EUR, of which 7% was for consumer products (as opposed to industrial solutions) [18]. If even a small (5%) of these consumer sales were for smart lighting product, this would translate to millions of shipped units in 2015 alone.

Although it is hard to get reliable figures, Philips Hue is considered the most popular smart light system. It was first released in 2012 and since then a large variety of Hue products were introduced. As can be seen in figure 3 their product line includes different models of lights, bridges

and switches. Philips provides an open API to the bridge, allowing 3rd parties to develop applications that can control the lights.

2.2. ZLL Touchlink Commission protocol

In the ZLL official website [16], we can find the "advertised security" of the ZLL standard, claiming to use authentication to "secures networks from neighboring networks", while allowing interoperability of products from different vendors. The basis for this is the ZLL Touchlink commission protocol defined in the ZLL standard [17]. This protocol is used to establish PANs (Personal Area Networks) and then instruct new devices (such as light, remotes etc.) to join the PAN and to receive the encryption key. Each PAN has its own unique encryption key which is used to encrypt and authenticate the messages sent in the PAN. For example, the Philips Hue starter kit includes two lightbulbs and a bridge that can be connected to a router for internet access. The controller and lightbulbs are preconfigured to be in the same PAN sharing a secret key. To connect a new lightbulb to the bridge, the Touchlink protocol is used to join the lightbulb to the existing PAN.

The ZLL protocol messages are not encrypted or signed. Encryption is only used to encrypt the unique encryption key sent to new devices joining the network. For this encryption, a secret "Master ZLL key" is used. This key is shared by and stored on all ZLL certified products. It is not specified in the standard and is only provided to ZigBee alliance members that are developing ZLL certified products. Unsurprisingly this master key was leaked in 2015 and can be found on-line [10]

Each Touchlink protocol instance is called a transaction. A transaction involves an initiator (in our example a bridge) and one or more targets. The initiator starts a transaction by sending a broadcast message called Scan request. Each transaction is identified by a unique 32-bit random nonzero ID that is sent in the Scan request. All other messages in the transaction will include this Transaction ID. Upon receiving the Scan request, the target sends a Scan response message that includes a random nonzero 32-bit Response ID. The combination of the Transaction and Response ID's identifies a unique transaction between an initiator and a specific target. There are 2 types of messages that can change the state of a lightbulb.

1) Reset to factory new request – Receiving this message with a valid Transaction ID will cause the target device to reset to factory new, deleting all PAN information and keys.
2) Join (or start) network request – These messages instruct the target to join the initiator PAN. They include the PAN's unique encryption key, encrypted using the ZLL master key derived with the Transaction and Response ID.

2.2.1. Touchlink's proximity check protection mechanism. The Touchlink protocol allows any initiator with the ZLL master key to force any lightbulb to reset to factory new or to join a new PAN. For example one can try to use a bridge to take over his neighbor's lightbulbs. To prevent this ZLL enforces a proximity check mechanism, that checks that the initiator is in very close physical proximity to the target.

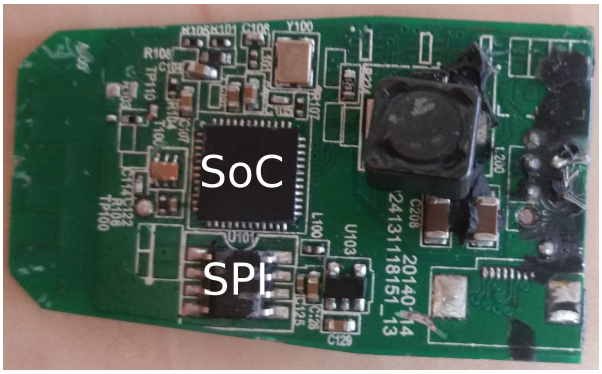

Figure 4. Philips Hue board

The Scan request is sent with a nominal output power of 0 dBm. Upon receiving this Scan request the target device checks if the measured RSSI (Received Signal Strength Indication) is above a certain manufacturer specific threshold. Otherwise it will ignore the request. In our experiments with the First Gen Philips Hue bridge, an older model lightbulb (Hue LCT001) will respond from about 1 meter distance, but a newer model (Lux LWB004) will only respond from about 45 centimeters. However in the Second Gen Bridge it seems that Philips increased the transmission power significantly and for the same Lux model the range increased to about 75 centimeters (about 2.7 times stronger).

2.3. Zigbee OTA (Over The Air) update

The ZigBee alliance provides a standard for OTA updates [19]. As written in the standard: "The main goal of Over The Air Upgrade cluster is to provide an interoperable mean for devices from different manufacturers to upgrade each others image". The standard defines a client server protocol for the transfer of an update image to the client (a lightbulb in our case). Although the standard suggests using asymmetric verification of authenticity and integrity, this is not mandatory for most ZigBee applications including ZLL.

2.4. Philips Hue light hardware and software

Although the first version of the Philips Hue light used Texas Instruments' CC2530 SoC (System on Chip), it was discontinued in 2012 and all lights produced afterwards use Atmel's ATmega2564RFR2 SoC. This SoC includes many different components:

1) An Atmel AVR microprocessor.
2) 256KB flash for bootloader and firmware code.
3) 32KB SRAM for program data.
4) An AES hardware accelerator.

5) An IEEE 802.15.4 low-power radio transceiver.
6) Anti debug fuses can be set to protect the firmware and internal data from being accessed from the outside.

These elements allow Philips to provide a system where the firmware, keys and all sensitive operations are protected inside the SoC with no access from outside the SoC. The Philips Hue light also comes with an external 4Mbit SPI (Serial Peripheral Interface) flash chip. Part of this flash is used to store the encrypted OTA images. A board that was extracted out of a Philips Hue Lux model light can be seen in Figure 4.

We assumed that Philips used Atmel's open source BitCloud ZLL stack as a base for their code. This assumption can be verified by viewing the SoC serial interface log messages which reference the BitCloud ZLL stack [12].

2.5. Counter with CBC-MAC encryption mode

CCM [20] is an authenticated encryption mode used to sign and encrypt data. It is used in IPSEC [21], TLS 1.2 [22] and the IEEE 802.15.4 [23] upon which ZigBee is based. As seen in Figure 6 the inputs to the CCM mode are a nonce N, associated data for authentication A, and plain data for encryption P. A and P are signed using CBC MAC. The nonce is combined with an incrementing counter, and then encrypted in ECB mode to create a CTR (counter based) stream cipher encryption used to encrypt P and the resulting CBC MAC tag. It is important to note that the same key is used both for the CBC MAC and CTR encryption.

2.6. Differential Power Analysis (DPA)

Side channel power analysis measures the power consumed by a digital device as it performs operations. With this attack it is possible to infer something about the data being processed by the device, which was first demonstrated as a method of breaking cryptographic algorithms by Kocher et al. [24]. While DPA may refer to both a specific technique and a general field, we will use DPA in this paper to refer to the original difference of means method of partitioning a number of power traces into two different sets from [24].

The difference of means relies on splitting power traces into two sets: one where an assumed intermediate bit is '1', and another set where the assumed intermediate value is '0'. By subtracting the means of these sets, we can determine the value of an intermediate bit. This is shown in practice in Figure 5, which shows the recovery of one byte of an AES-CTR output, with the use of the XOR operation. At a specific point (around sample 6952 here), we do a threshold if the difference is positive or negative. It can be seen from the figure the difference is very pronounced and in practice can be reliably recovered. This is then simply repeated for all 16 bytes.

Note the difference of means shows such a large spike only at the moment in time the manipulation is occurring. Other times show no noticeable difference – notice for example samples 6990 and onward in Figure 5. In practice it is possible to recover this from a black-box device with unknown timings, such as we accomplished in this paper.

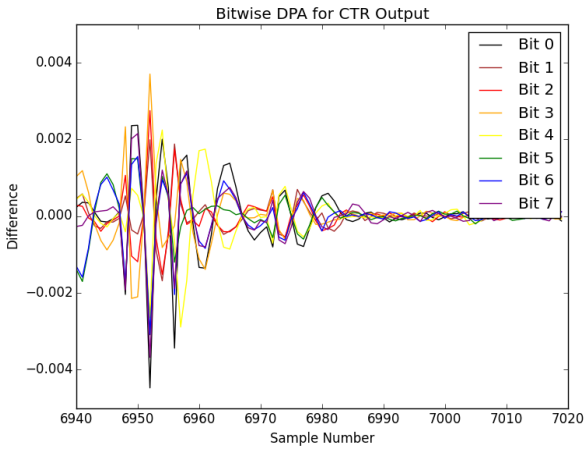

Figure 5. Bitwise DPA attack on AES-CTR 'pad', where all 8 bits are recovered.

2.7. Correlation Power Analysis (CPA)

The subsequently published CPA attack by Brier et al. [25] uses a more complex leakage assumption, such as the one that the number of bits set to '1' on a data bus has a linear relationship with the power consumption at the moment on time the data is manipulated. Rather than requiring an attack to determine a single bit at a time, the CPA attack makes it possible to rapidly determine the value of an entire byte. The CPA attack is especially effective when targeting the output of non-linear functions, such as the output of the S-Box operation in AES.

This work will use both the original bitwise difference-of-means DPA attack for determining the result of certain XOR operations (such as the AES-CTR 'key stream'), and the byte-wise CPA attack for breaking the AES hardware accelerator.

3. Experimental Setup

3.1. Overview

Our experimental setup includes three main parts:
1) A research experimental setup used to test the ZLL protocol and its implementation.
2) A hardware attack setup used to reverse engineer and attack the Philips Hue OTA update process.
3) A ZLL attack setup used for testing and demonstrating our takeover attack.

3.2. The ZLL experimental setup

We created an experimental setup that allows us to send and receive ZLL messages and run complex state machines

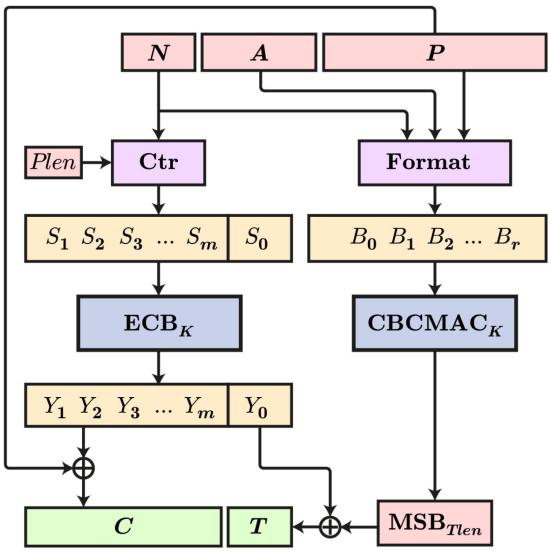

Figure 6. CCM encryption mode

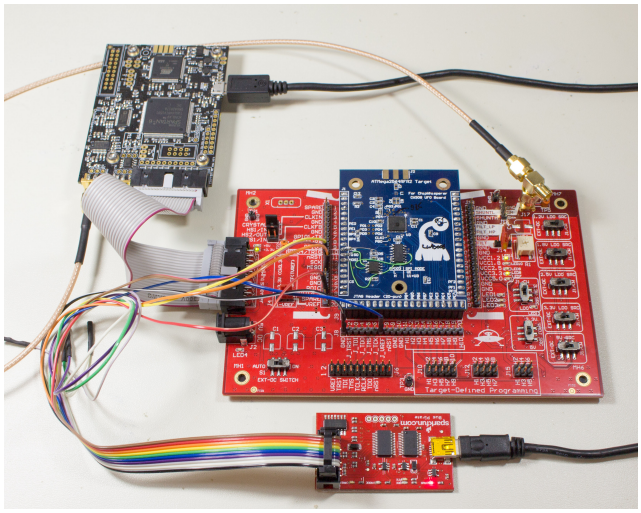

Figure 7. Power analysis on the ATMega2564RFR2 from a Philips Hue bulb was done using a ChipWhisperer-Lite (top left), connected to a custom PCB with the ATMega2564RFR2 mounted (middle blue PCB) and using a Bus Pirate (bottom small PCB) to reprogram a SPI flash chip with various byte sequences.

in order to research and implement our attacks. Although similar work had been done previously by Wright [26] and Goodspeed et al. [27] we decided to create our own setup that was more suitable to our available hardware and our specific needs, and also to implement the relevant parts of the ZLL standard that were not previously available.

3.2.1. Hardware setup. We needed a RF transceiver capable of sending and receiving ZigBee messages. We have chosen to use Texas Instruments' CC2531EMK evaluation board. It is a USB dongle that can be connected to a PC, and uses the same CC253x family chip and ZLL stack as the older Philips Hue lights (the only difference is the added USB support). The default software provided allows for sniffing of raw ZigBee messages. We have compiled our own code to be able to also send raw ZigBee messages.

3.2.2. Software framework. We created a Python implementation of most of the ZLL stack. This provides us a simple and quick method to generate and parse ZigBee and ZLL messages, and also to implement a state machine for implementing our attack.

3.3. Power analysis setup

3.3.1. Philips Hue board. Several of the 1st-generation Philips Hue (BR30 color) and 2nd-generation Philips Hue Lux (white-only) bulbs were disassembled. The 2nd (and later) generation Lux bulbs are using an Atmel ATMega2564RFR2 device. Our first power analysis was done on a modified Hue board. Later, a custom PCB was designed to fit the ATMega2564RFR2 along with support circuitry such as the SPI flash chip. Several of these boards were built: some were loaded with chips removed from production Hue lights, some were loaded with blank chips.

The boards loaded with Hue chips were used in breaking the actual bootloader and encryption key, whereas the blank chips were used in investigating the power signature of the ATMega2564RFR2 when performing known tasks, and determining the rough amount of power traces required to break the hardware AES peripheral.

For power analysis we needed many power traces where the bootloader is decrypting the same block, but that block takes on random input values. To accomplish this, a low-cost SPI programmer quickly re-writes the SPI flash chip, and the ATMega2564RFR2 reset pin is toggled. This causes the bootloader to attempt loading of the binary – loading which will fail after the bootloader realizes the signature is invalid, but by this state has already performed the required operations for power analysis to succeed.

The power measurement itself is done via a resistor inserted in the core power supply pin, as in [14]. The power measurement is taken using the ChipWhisperer hardware platform [28], and a photo of the power measurement setup is shown in Figure 7.

The ATMega2564RFR2 device was assumed to have similar leakage characteristics as the ATMega128RFA1 explored in [14]. To evaluate our test setup we performed a CPA attack against the hardware accelerator, which is detailed in Appendix A.

With this setup we are able to attack AES-ECB using a CPA attack, and reliably recover the encryption key. As in previous work this can be done on later rounds, which can also be used for testing if the correct first-round key was recovered when there is no access to the output of the AES-ECB block.

3.4. ZLL attack setup

For our attack demonstrations we used a slightly different setup. For our transceiver we used the same TI CC2531 chip but on a Zlight2 evaluation board. This board has a slightly better RF characteristics and it is easier to mount on a drone during warflying missions. As we wanted our attack kit to be fully autonomous we have implemented our attack logic in C code using Texas Instruments' ZigBee ZStack.

We bought 4 different models of the Philips Hue lights to test our attack on.

4. Creating a lightbulb worm

4.1. Attack scenario

Our goal is to create a worm that can automatically spread among physically adjacent lights in a chain reaction which spread over a large area using only the standard ZigBee wireless interface. Our worm will spread from one lightbulb to all the neighboring lightbulbs and from one ZLL network to another. For this we require two main abilities:

1) Persistence of code execution on a lightbulb.
2) Lateral movement - a method for one lightbulb to infect another lightbulb.

4.2. Persistence of code execution in the Philips Hue system

We explored two methods to achieve persistence in code execution

1) Exploiting a software vulnerability.
2) Using the ZigBee OTA feature.

The Philips Hue lightbulbs are very hard targets for finding and exploiting software vulnerabilities. They use processors with Harvard architecture that does not allow for code execution from memory. We are forced to use only ROP (Return-oriented programming) attacks that require knowledge of the code and customization for each model. Instead we looked at the possibility of exploiting the ZigBee Over-the-Air Upgrading Cluster standard [19]. As this standard allows a lot of customization for different vendors, we wanted to record and analyze a Philips Hue software update. Unfortunately there was no software update available to any of the models we had in our possession. We wrote a python script using our ZLL testing framework that allowed us to impersonate a lightbulb and to change our version, MAC address and so on. Searching online we found out several different older Hue models that had software updates in the past. By looking at recordings for the models we had, we found the translation between the Human readable software version and the hex code that the OTA standard requires to send over the air. For example in our Lux model the software version is 66012040 – 66 0 12040 – 0x42 0x00 0x2f08 – 0x42 0x00 0x2f 0x08. Using the API we instructed the bridge to complete the OTA process with our impersonation code. The firmware image that we recorded was not dependent on our impersonated lightbulb MAC address. This brought us to the conclusion that the software is protected by a single key that is shared at least between all the lights from a specific model. Implementing asymmetric cryptography is currently uncommon in this type of products (as can be seen in several OTA implementations such as Atmel's BitCloud [29] and Texas Instruments' CryptoBootloader [30]). OFlynn [12] have also reached similar conclusions. Assuming that only symmetric cryptography is used, recovering the encryption and authentication keys from one Philips Hue lightbulb will allow us to do a software update to any other lightbulb (at least from the same model) and thus load our own malicious code.

An important observation is that unlike computers or smart phones, this kind of attack is irreversible. There is no way to re-flash the Philips Hue lights firmware to get rid of our worm, and the only possible solution is to replace the lightbulb with a new one. Note that in order to prevent the new lightbulb from being infected in the same manner, the user must wait for a software patch to be available from the manufacturer before installing it.

The user however would need to power on the bulb to allow the manufacture update to occur. Obtaining this update requires at minimum first installing the bulb and setting up the bridge – currently it additionally requires the user to download and setup the official app and agree to the terms of service before the update process begins.

The worm however could begin downloading to the bulb as soon as power is applied, not being dependent on the base station. In addition the worm can rapidly "retake" new bulbs which the user has attempted to associate with the legitimate base station, making it almost impossible for vulnerable bulbs in range of another infected bulb to receive an OTA patch before the worm has spread.

4.2.1. Understanding the Philips OTA image structure. As there were no software update yet available for one of the new models[2], we looked for a way to create our own OTA image. We used our impersonation code to retrieve all available OTA image files for the older models. We then noticed that the new models rejected the old firmwares. We have reversed engineered the Philips OTA image structure. With some trial and error we manipulated an old firmware image, to be compatible with the hardware type and image size range expected by the new Atmel model. As the light uses the external SPI flash to store the update Image, we used a SPI sniffer to record and understand the communication between the processor and flash during the OTA process. To start an OTA image verification process all we need is to set a flag on the flash and put the image in a specific offset in the flash. During power on, the processor checks the flag, and if it is set, it starts to read and then verify the OTA image.

4.2.2. Extracting encryption and signature keys. To be able to create a valid malicious software update, we had

2. Firmware updates for these bulbs were released before this paper was published, but at the time this work was completed they were not available.

to understand the cryptographic primitives used to encrypt and sign the update firmware, and extract the keys used. As mentioned in [12], Philips had set all of the ATMegaRF anti debug fuses, to disable external reading of the program and keys saved internally. In section 5 we describe how we were able to break Philips bootloader using CPA.

4.3. Lateral movement - spreading the worm

To be able to carry out a software update on a lightbulb, we must first be on the same network and share the same key. This can be done either by sniffing the Touchlink protocol when new lights are added, or by forcing the lightbulb to join our own network. As the ZLL secret master key was leaked, we can use the Touchlink commissioning protocol to take over lights. However we are limited by the protocol's proximity check mechanism which forces us to be very close to the attacked light (as discussed in section 2.2.1). Previous methods to cause a key exchange in the network or to force the light to join a new network required either very close physical proximity or a customized hardware with much stronger transmission power [10], [11], and in that case we will not be able to use one standard light to infect another light. To implement our worm we have to find a way to do this from a long distance using normal power levels. In section 6 we show how we accomplished this with the Philips Hue lights.

5. Breaking Philips' cryptographic bootloader with Correlation Power Analysis

5.1. Understanding Philips OTA image cryptographic primitives

Our initial assumption was that Philips used the CCM encryption mode for the OTA image. This enables them to reuse the CCM code from the Zigbee encryption, which was also used in an old TI cryptographic bootloader implementation, that could have been used as a reference to their implementation in the older TI based models.

When we started this work, the newer bulbs based on the Atmel ATMega2564RFR2 did not have an OTA update released. Instead we used an image for the CC2530 bulbs as a reference. To perform the bootload process, the new (encrypted) image is programmed in the SPI flash. On boot the bulb will first check a flag to indicate if an OTA update is pending,; if so, it reads the entire image to verify the signature. Then it reads the image a second time to actually perform the flash programming. We determined this based on (1) modifying the image – which would invalidate the signature – causes the bulb to perform only the first read, and (2) the second read-through contains gaps which align with the expected flash memory page-erase process required when actually programming.

As we knew the leakage mode for the ATMega2564RFR2 AES hardware engine, we targeted the newer hardware. The CC2530 OTA upgrade file was modified by changing the hardware type and image file size to fit the requirements of the bootloader on the new hardware, so the bootloader on the ATMega2564RFR2 would attempt the verification process. The actual verification will *fail* as this was not a valid OTA image for this platform, meaning we were able to perform this attack without having access to a valid firmware image.

The hardware AES engine on the ATMega2564RFR2 has a unique signature which makes detecting the location of AES straightforward. Looking at the power traces of the verification process we could notice two AES operations for each 16 byte block, which supported the CCM assumption as shown in the top portion of Figure 8.

In addition, we performed a DPA attack where the leakage assumption is simply the input data itself being loaded. This shows locations where the input data is manipulated (this will also track linear changes to the data, such as an XOR operation). We notice the input data is manipulated after the first AES operation and before the second AES operation as in the lower part of Figure 8.

This would be consistent with the first AES operation being CTR mode, the output of CTR mode being a pad which is XORd with the input data to decrypt the block. The decrypted data is then fed into the CBC block. Note the XORs of the input data still generate the high difference spikes, as the input data is effectively being XORd with constants (either the AES-CTR output with the same CTR input, or the CBC output).

5.2. CPA attack against the CCM CBC MAC verification

Under the CCM assumption, we had to find a way to break the mode of operation under the following limitations:

1) We have no knowledge of the key
2) We have no knowledge of the encryption nonce
3) We have no knowledge of the signature IV or associated data.
4) We have no sample of a valid encrypted message.
5) Our target won't accept messages larger than around 2^{14} encryption blocks.

We will first summarize the existing related work on breaking AES-CCM.

5.2.1. Previous work on AES-CTR and AES-CCM. Performing power analysis on AES-CTR mode is made more complicated as the majority of the bytes are constant (the nonce), and only the counter bytes vary. A standard first-order CPA attack is only able to recover the key-bytes where the associated input bytes vary, meaning that at most two bytes of the counter are recovered. A solution to this was presented by Jaffe, where Jaffe performs the attack over multiple AES rounds [15].

Jaffe's technique of performing the attack over multiple rounds allows recovery of a combination of the AES Round-Key XORd with either the constant plain-text or the output of the previous round. This allows us to ignore the unknown constant values, as they will eventually be removed [15].

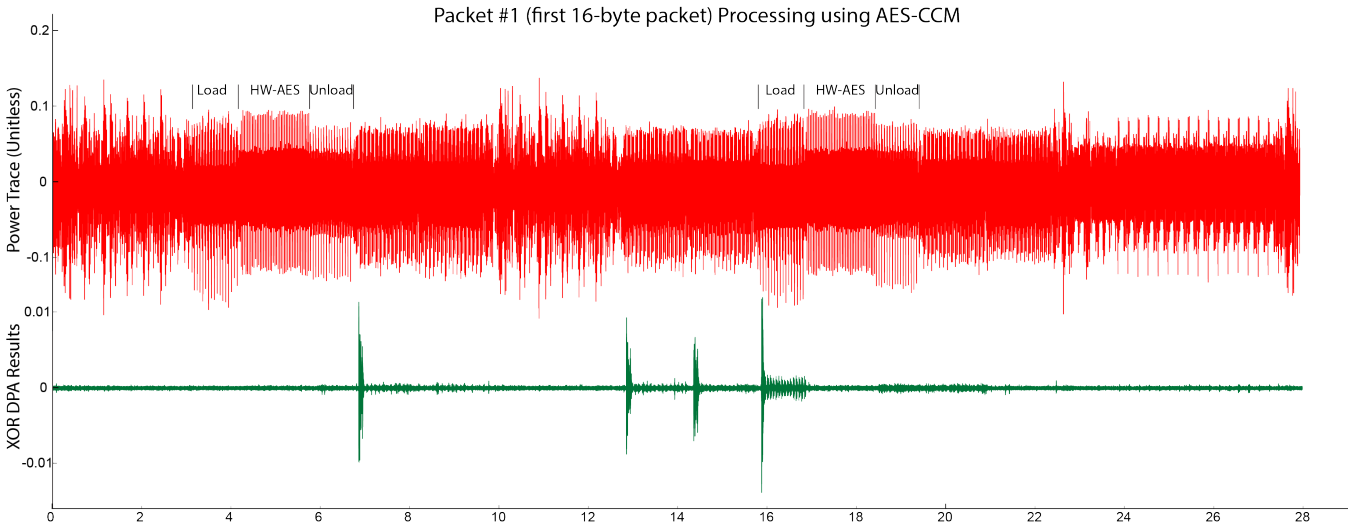

Figure 8. Power analysis of processing a single 16-byte block by the cryptographic bootloader. HW-AES locations are marked based on comparison to a reference platform performing hardware AES encryptions.

The AES-CTR attack requires 2^{16} encryptions, to ensure power traces are recorded for all values of all 16 bits of the counter. While Jaffe reports the attack may succeed with a smaller subset of these 2^{16} traces, the subset will include traces from throughout the set, such that even if a set of 2^{14} traces pulled from the larger set was sufficient, capturing only 2^{14} *consecutive* traces will not provide enough data.

The leakage observed by the hardware AES peripheral is such that leakage occurs before the S-Box operation, and it is not possible to reliably perform the attack using the output of the S-Box. Had the output of the S-Box leaked, it would have been possible to recover higher-order bits of the key for which there is no associated toggling of higher-order bits of input data, due to the non-linear property of the S-Box.

The AES-CCM as used in IEEE 802.15.4 was specifically broken by O'Flynn and Chen, which used a modified version of Jaffe's attack [14]. Rather than use successive encryptions with AES-CTR, O'Flynn used the known mapping of input data to the AES-CTR nonce to allow him to perform power measurements where 4 bytes of the nonce are varied, but the AES-CTR counter is constant.

As in Jaffe, this required O'Flynn to perform multiple rounds of CPA attacks as the key was progressively recovered. For every round-key recovered, some work is needed to validate that it appears to be correct as well, since any error will compound. This was done by looking for correlation spikes for round $i+1$, when comparing various candidates for round key i. O'Flynn did not use the AES-CBC portion of the AES-CCM process and instead only broke the AES-CTR portion.

A solution to the general problem of unknown counter inputs is also given by Hanley et al. where a template attack can be performed even with completely unknown input to the AES block [31]. This attack has the downside of worse performance (in terms of number of traces required) compared to a known plaintext (or ciphertext) attack.

Because our target only accepted about 2^{14} 16-byte blocks, we had limited ability to use the existing AES-CTR attacks. We also were unaware of the nonce format – if we had a known mapping of some input data field to AES-CTR none, the attack in [15] as used by [14] would have been possible.

5.2.2. Unknown Plaintext with Chosen Differentials CPA attack against AES. For our attack we introduce a novel method of efficiently converting most chosen plaintext CPA attack against ECB mode in the case of unknown plaintext with chosen differentials. Our attack works under the following assumptions:

1) We have a black box chosen plaintext CPA attack that can break the first round of an ECB mode encryption implementation.
2) We do not know the input to our ECB mode encryption, but we can measure repeated encryptions with the same unknown input XORed to any chosen differential.
3) For each differential we can measure the power trace of at least the first and second AES round.

As in previous works [15], we use the notion of a 'modified key'. We will use the following notation:

P_i is the ith byte of the unknown plaintext input to the AES encryption.
D_i is the ith byte of the chosen differential.
$K_{j,i}$ is the ith byte of the jth round key of AES.

In the general case of first round of AES encryption, the key and plaintext bytes are used in calculation of the output of the SBOX. The output of the SBOX on byte i can be written as $Ouput_i = S(P_i \oplus K_{1,i})$. Any chosen plaintext CPA attack on the first round will be able to retrieve all of the bytes of K_1 by measuring traces of different inputs P. In our case P_i is constant and we can choose D_i, we get $Ouput_i = S(D_i \oplus P_i \oplus K_{1,i})$. We will denote our 'modified

key' as $K'_{1,i} = P_i \oplus K_{1,i}$, rewriting $Ouput_i = S(D_i \oplus K'_i)$. We can now use our black box CPA attack to retrieve all of the bytes of K'_1. Using D and K'_1, we can now calculate the input to the second AES round. As the first and second round of AES are identical, we can use the same black box CPA attack against the second round with known inputs, and retrieve the real second round key. In most CPA attacks we can choose our inputs at random, and use the same power traces we used for the first round attack. If real chosen plaintext is needed, we can use the invertible structure of the AES round, and calculate the required differentials in the first round.

After getting the real second round key K_2, we can use the invertible AES key expansion algorithm to find K_1 and then all bytes of P_i. In the normal case where only random plaintext is needed for the CPA attack, we can break our ECB mode with unknown plaintext and chosen differentials in the same number of traces required to break ECB with chosen plaintext.

5.2.3. Breaking AES-CCM. For efficiently breaking the CCM mode, we attack the CBC MAC state calculation, on two consecutive blocks. We'll first summarize some notation for AES-CCM.

If we consider the AES-ECB function using key k as $E_k(x)$, we can write the CTR and CBC portions of the CCM mode as follows. The input will be in 16-byte blocks, where block m is the index. CTR mode requires some IV and counter which is input to an AES-ECB block, we assume our input is $\{IV||m\}$, where IV is a 14-byte constant that is concatenated to the block number m. Counter mode first generates a 'stream' based on the counter and IV:

$$CTR_m = E_k(\{IV||m\})$$

This stream is XORd with plaintext/ciphertext for encryption/decryption respectively. Thus decrypting block PT_m would be:

$$PT_m = CT_m \oplus CTR_m$$

In addition to decryption, CCM provides the authentication tag which is the output of a CBC mode encryption across all PT_m (and possibly other) blocks. The internal state of this CBC mode after block m will be CBC_m, which can be written as:

$$\begin{aligned} CBC_m &= E_k(PT_m \oplus CBC_{m-1}) \\ &= E_k(CT_m \oplus CTR_m \oplus CBC_{m-1}) \end{aligned}$$

If we target a given block m, CTR_m and CBC_{m-1} will be constant. CT_m is the ciphertext we input to the block (e.g., by the firmware file we sent the device), allowing us to control the value of CT_m. We consider our unknown plaintext to be $CTR_m \oplus CBC_{m-1}$ and using the ciphertext as the chosen differential, and then we can use our CPA attack to recover the CBC MAC key k, and the value of $CTR_m \oplus CBC_{m-1}$. As the CCM mode reuses the key between encryption and verification we also get the key used for encryption. We now repeat our attack for the first round of block $m+1$. From our attack on block m we can calculate CBC_m, and from our attack on block $m+1$ we can retrieve $CTR_{m+1} \oplus CBC_i$ and from that CTR_{m+1}. We can now find the nonce used by decrypting CTR_{m+1} with the key we found.

5.2.4. AES-CTR DPA Recovery Optimization. Our attack on CCM requires twice the traces of ECB mode since we must attack two blocks: both CBC_m and CBC_{m+1} in order to retrieve the CTR output. We can optimize our attack by using a bitwise difference-of-means DPA attack to recover the output of the AES-CTR encryption directly for block m. The DPA attack is attacking $CTR_m \oplus CT_m$ operation. An example of this on the actual bootloader power measurement is shown in Figure 5, where a single byte is being recovered.

Note there may be multiple locations where a strong 'difference' output is seen. These locations come about as any linear operations on the CT_m data will present such spikes – for example not only the XOR we are targeting, but also the data loading, and when the AES-CTR output is used in the AES-CBC input. In addition there will be both positive and negative spikes as the internal bus switches from precharge, to final state, back to precharge.

We found about 10 locations with such strong differences across the entire trace, giving us 10 possible guesses for the output of the AES-CTR on the first block on the same traces under the CPA attack. Using the key retrieved from the CPA attack, we tried decrypting the guesses, and simply chose the one that decrypted to the correct counter value in the last bytes. The correct guess occurred in the window where the AES-CTR XOR operation was expected to occur (around sample point 6950 in Figure 8), meaning the additional guessing may not be required in most cases.

5.2.5. Extending the CCM attack to other block ciphers. By combining the CPA attack and DPA optimization, we can break any SPN(Substitution-permutation network) based cipher block algorithm regardless of the key expansion algorithm under the following assumptions:

1) We have a CPA attack that can break any round using chosen plaintext.
2) We can measure the power traces for all rounds
3) The DPA attack provides a small number of possible guesses for the output of the CTR.

We use the CPA attack against block m to retrieve the CBC MAC state CBC_m. By using the DPA attack we can retrieve the possible guess to the CTR output CTR_{m+1}. As the CBC MAC encryption in block $m+1$ is $E_k(CT_{m+1} \oplus CTR_{m+1} \oplus CBC_{m-1})$ we can now do a chosen plaintex attack against block $m+1$ to retrieve all of the round keys including the first round.

5.3. Loading arbitrary code

At this stage we know the AES-CCM key used by both the AES-CBC and AES-CTR portions, and the AES-CTR nonce format. We also have the CBC MAC state that

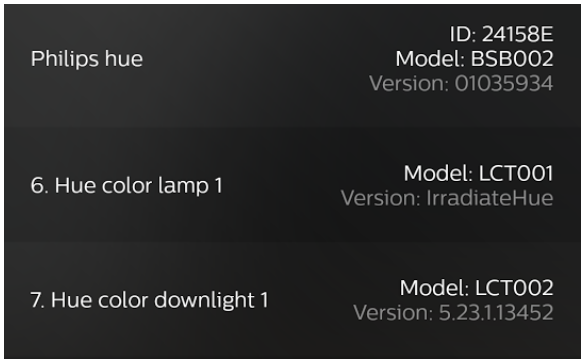

Figure 9. Screenshot of Hue App connected to a modified bulb; the version number of bulb 6 has been modified by us in the firmware image.

is XORed to the first block (after encryption of IV and associated data) that we can use as a 'modified IV'.

We used a DPA attack to determine which bytes of the header were manipulated after the last block of data was received by the bootloader, assuming that those will be the bytes of the MAC authentication tag being compared to the calculated value.

Using this information we can now encrypt and sign arbitrary code. As a first test, we decrypted one of the CC2530 update images, changed some strings and then re-encrypted the file with a newly calculated MAC. This firmware was accepted by the light, confirming that we were able to generate signed code. An example screenshot of the official Philips Hue App showing a modified and unmodified bulb is present in Figure 9. Note that the version number of the modified bulb has been changed to `IrradiateHue`.

We then loaded a small program that read out the first few pages of flash memory. On the CC2530 the bootloader typically resides at the beginning of flash, so this allows us to recover the complete original bootloader (along with full information on keys, I.V., etc.). Note that on the ATMega2564RFR2 there is an option which prevents application code from reading out the bootloader, but this feature was not enabled on Hue bulbs using the ATMega2564RFR2 we have investigated, meaning that we can perform this same style of attack on newer lights.

6. Take over attack

6.1. Bypassing the proximity check mechanism

The proximity check ensures only a very close device (approx 1m or less) is able to reset and then take over a device, as was discussed in section 2.2.1. This proximity check logic will be present in any ZLL compliant stack (and product).

6.1.1. Atmel's BitCloud Touchlink implementation. Atmel provides their customers with a complete software implementation of the Touchlink protocol in the BitCloud stack under the ZLL Platform section. It is implemented in two parts:

Listing 1. Size check examples

```
case SCAN_REQUEST_COMMAND_ID:
  if (ind->asduLength ==
      sizeof(N_InterPan_ScanRequest_t))
  {
    ProcessReceivedScanRequest(ind);
  }
  break;
case SCAN_RESPONSE_COMMAND_ID:
// size is checked in ProcessReceivedScanResponse
  s_ProcessReceivedScanResponse(ind);
  break;
```

Listing 2. Response Parameters structure

```
typedef struct N_LinkTarget_ResponseParameters_t
{
  uint32_t  transactionId;
  uint32_t  responseId;
  uint8_t   zllInfo;
  uint8_t   zigBeeInfo;
} N_LinkTarget_ResponseParameters_t;
```

1) N_InterPan - Handles the InterPan layer, determines what type of message was received, and does some basic sanity checks
2) N_LinkTarget - Implements the Touchlink protocol state machine.

The code was written with security in mind. Message sizes are either checked or a clear reference is made for the location of the check. A code example can be seen in Listing 1.

The proximity check mechanism is implemented in N_LinkTarget.c at the function **ReceivedScanRequest**. It checks that the RSSI value is above a threshold that is defined by the stack user for every specific product. Only if the value is over the threshold it will save the message parameters and start the Touchlink protocol state machine.

Upon receiving a valid Scan Request message, the state machine waits for a random jitter time period (to make sure not all lights respond at exactly the same time). After that time period the function **StartTransaction_StoreCurrentTransmitPower_SendScanResponse-_StartChangeDelay** is called. This function sends a Scan Response message. The Transaction and Response ID are saved in the ResponseParameters structure that can be seen in Listing 2. To support multiple instances of the Touchlink protocol the stack holds an array of length 3 of this structure.

Upon receiving any other protocol message, the transaction ID parameter of the message is checked against the values stored in the array. If the transaction ID is not found in the array, the message is dropped.

6.1.2. Bypassing the proximity check. The Response Parameters structure seen at Listing 2 has no dedicated valid flag variable. After boot, or at the end of a protocol session, the structure is filled with zeros, as zero is considered an invalid value for the Transaction Id.

Listing 3 shows the function **IsTransactionIdActive** that is used to check if a Transaction Id is valid or not. The reader

Listing 3. Transaction Validation Check

```
/** Check if the transaction id is active.
\note The value zero is already rejected
      by N_InterPan.
*/
bool IsTransactionIdActive(uint32_t transactionId)
{
  if (GetFromResponseTable(transactionId) == NULL)
  {
    return FALSE;
  }
  return TRUE;
}
```

can see a note written by the programmer stating that there is no need to check if the Transaction Id value is nonzero as "the value zero is already rejected by N_InterPan". However, as we reviewed the code, we found out that this sanity check is only done upon receiving a Scan Request message. There is no such check for any other message in the protocol. This means that we can send any other message assuming zero value for the Transaction and Response Ids and it will be received and process as a valid message by the light.

6.1.3. Taking over light bulbs. After we discovered this bug, we looked for ways to exploit it to take over lights from a large distance. Our first approach was to simply send a Network join router request with Transaction ID set to zero. This message will instruct the light to join our network. However this only caused the light to reset. After looking at the code, we found that the function **N_Security_DecryptNetworkKey_Impl** that is called to decrypt the new network encryption keys, performs a sanity check to make sure that the Transaction and Response Ids are nonzero, and resets the light in such case. As joining a light to a new network requires sending a network key, we could not find a way to use the ZLL Touchlink protocol directly to take over the lights.

However, as is also mentioned in [10] the ZLL standard mandates compatibility with non-ZLL ZigBee networks. In the ZLL standard [17] at section 8.1.6 it is written that: "In order to ensure interoperability with other ZigBee devices, all ZLL devices should implement the compatible startup attribute set (SAS) specified in this sub-clause. ZLL devices can join other non-ZLL ZigBee networks and allow non-ZLL devices to join ZLL networks under application control.". This commissioning protocol described in [32] does not have any restrictions such as the proximity check mechanism. Upon joining a network the network key is sent encrypted with a "Trust centre link key" default key that is specified in the ZLL standard. While under normal conditions a Philips Hue light will not try to join a ZigBee network, it will actively search and try to join such networks if it is in a "Factory New" state.

Our attack proceeds in the following way: We send a unicast Reset to Factory New Request command to our target Philips Hue light. This command only included a Transaction ID value that we set to zero. Upon receiving the message the light will undergo a factory reset, and start scanning for a new network by sending a ZigBee Beacon Request messages. Then we respond with a ZigBee Beacon message with the Association Permit flag set to true. This causes the light to start a ZigBee association process and join our network.

6.1.4. Simultaneously taking over multiple lightbulbs. Our next step was to improve our attack, so that it can be run simultaneity and efficiently against several lights.

The ZLL standard clearly states that the Reset to Factory New Request command shall be formated such that "the destination address field shall contain the IEEE address of the destination and the source PAN ID field shall be set to the same value used in the preceding scan request inter-PAN command frame". However, this is not verified anywhere in the code. This allows us to send the Reset to Factory New Request command as a broadcast message, simultaneity causing all lights within ZigBee range to factory reset. After that all the lights will respond to our Beacon message and start the association, and join our network.

But what will happen if one of the lights didn't receive our Factory New Request command, or was just outside the range? If we will try to resend the message it will cause the lights already associated to our network to reset again. To solve this problem we use the ZLL support of different channels available at 2.4 GHz range. The ZLL standard states that: "A ZLL device shall be able to operate on all channels available at 2.4GHz, numbered from 11 to 26 ... Within this range, two sets of channels shall be defined. The primary ZLL channel set shall consist of channels 11, 15, 20 and 25 and shall be used in preference for commissioning and normal operations. The secondary ZLL channel set shall consist of channels 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24 and 26, which can be used as a backup to allow the ZLL device to connect to a non-ZLL network". After a factory reset, the Philips Hue light cycles through all available channels. In each channel it sends a Beacon Request message, and waits for a short period of time for a response. It will chose one of the relevant Beacon messages it received, and switch to that channel for association and further communication. We repeatedly send the factory reset messages in the primary ZLL channels, and then switch to one of the secondary channels for sending the Beacon message and association process. This allows us to try to repeatedly reset all the lights in our area, while keeping the lights we already took over on our network in a secondary channel so that they will not be affected.

6.2. Implementing and testing our attack

We started by verifying our attack against all the different Philips Hue light models we had in our lab, using our experimental setup. We then implemented an autonomous attack kit using Texas Instruments' ZLL stack and the ZLight2 evaluation board powered by a 1800 mAh USB powerbank (that is enough for more than 20 hours of work). The stack we were using did not have a convenient API for on-the-fly channel switching (this is supported by the

Figure 10. ZigBee wardriving with Zlight2 Evaluation Board

Figure 11. ZigBee warflying scenario

hardware but doing this will require us to rewrite large parts of the code and replace some closed source libraries of the stack we used). So in order to simplify our development process we chose for our experiment to attack only via the first ZigBee channel 11 and to use two evaluation boards, one for the factory reset attack on channel 11 and another for the association and takeover on channel 24. The evaluation board used for the factory reset attack was programed to repeatedly send a Reset to Factory New Request command every 3 seconds at the highest transmission power (about 4.5 dBm) . This gives the lamps ample time to switch to the other channel and complete the association process with the second board before the next Reset to Factory New Request command is sent

6.2.1. Wardriving. - As can be seen in Figure 10, we installed 3 Philips Hue lights in offices at the first floor of our faculty building. We successfully tested our full attack from a car which was parked across the lawn at a distance of about 50 meters, while the factory reset part of the attack worked from ranges of more then 150 meters. We then successfully tested our attack while "wardriving" the car at the far edge of the lawn.

6.2.2. Warflying. - For a warflying demo we found a more interesting target: an office building in the city of Beer Sheva in the south of Israel which has a high concentration of well-known cyber security companies (and is also next to the building that host the Israeli CERT). As can be seen in Figure 11, we have installed 5 Philips Hue lights on the third floor of the building. We then mounted our attack kit on a DJI Inspire pro drone as can be seen in Figure 12. The powerbank was attached to the bottom of the drone, while

Figure 12. Attack start position

the evaluation boards were hanging from a 1 meter USB cable beneath it. This was done to avoid RF interference from both the drone's motors and its powerful 2.4 GHz video and control transmitter.

As can be seen in Figure 12 we started our attack on the ground at a distance of about 350 meters. Right after takeoff, at that distance, the factory reset part of the attack started working. As the drone got closer to the building the takeover part was completed. To demonstrate the successful take over, we added code that causes all the lights to repeatedly signal SOS in Morse code while the drone hovered in front of the building.

6.3. Ethical disclosure

We made full ethical disclosure of the takeover vulnerabilities found to Philips Lighting and to Atmel on July 2016, providing all the relevant technical details and suggestions for a fix. We received a confirmation of our findings from Philips. As this attack affects most of the deployed Hue lights the fix had to be tested on a large number of versions, and the first update was released by Philips in October 2016, which would reduce infection range to only 1m or less using standard ZLL Touchlink messages. We have also notified Philips on the recovery of the firmware encryption keys. Those finding were also confirmed.

7. Worm application

7.1. Bricking attack

We can use the worm for a bricking attack. Unlike regular DoS attacks this attack is irreversible. Any effect caused by the worm (blackout, constant flickering, etc.) will be permanent. As previously mentioned, once the worm is

downloaded, the worm can decide what OTA updates to allow. The worm is entirely replacing the existing firmware, so it is up to the worm designer to support the OTA update protocol if they wish to allow other OTA updates.

There is no other method of reprogramming these devices besides a PCB-level connections. As these lights are not designed to be disassembled – having a potting compound around the main PCB, along with glue used to secure the main dome – fixing the issue would require a substantial recall or warranty replacement.

Any old stock would also need to be recalled, as any devices with vulnerable firmware can be infected as soon as power is applied. The consumer is unlikely to have time to perform the legitimate OTA update before the worm would infect the bulb, as we previously discussed.

7.2. Wireless network jamming

The IEEE 802.15.4 standard which ZigBee runs over uses the 2.4 GHz ISM (Industrial, Scientific, Medical) license-free band. This band is widely used by many standards, including IEEE 802.11b/g (n mode supports both 2.4 GHz and 5 GHz bands). Due to the low data rate of IEEE 802.15.4 the channel bandwidth is lower (2 MHz for 802.15.4 vs 20 MHz for 802.11g), meaning that several 802.15.4 channels overlap with each 802.11 channel.

These 802.15.4 SoC devices have a special 'test mode' which transmits a continuous wave signal that is used during the FCC/CE emission certification process. This test signal can be tuned to overlap on any of the 2.4 GHz 802.11 channels (or sweep between them). Since the radio is not performing the clear-channel assessment (CCA) in this mode it can be used as a very effective jammer. As this signal may come from many devices at once, such a jammer could easily be used to disrupt WiFi (or other 2.4 GHz) traffic in an area.

A more dedicated attacker could also use this platform in attacking specific products using IEEE 802.15.4 on the 2.4 GHz band: examples include WirelessHART, MiWi, ISA100.11a, 6LoWPAN, Nest Weave, JenNet, and Thread. It is possible for example to perform more specific DoS attacks against specific devices or protocols [33].

7.3. Data infiltration and exfiltration

Ronen and Shamir [7] have used the Philips Hue to exfiltrate data at a rate of about 10KB per day, getting one bit of data from every message sent from the bridge. Using infected lights, we can create a similar covert channel, at much higher rates. This can be done by reading bits from user chosen data in message and flickering at higher rates than that allowed by the API. Infected lights can also be used to infiltrate data into the network, by changing user readable data such as model version.

7.4. Epileptic seizures

Ronen and Shamir [7] have also shown how the Philips Hue can be used to trigger epileptic seizures. This attack can now be executed from a remote location, covering large areas. In infected lights it is also possible to drive the LEDs at frequencies that increase long-term discomfort in humans rather than attempting to overtly trigger seizures [34].

8. Conclusions

In this paper we described an attack which has the potential to cause large scale effects. Moreover, fixing the malicious software update will require the physical replacement of every affected lightbulb with a new one, and a waiting period for a software patch to be available before restoring light. This scenario might be alarming enough by itself, but this is only a small example of the large scale problems that can be caused by the poor security offered in many IoT devices.

Our attacks exploit a specific implementation bug of the Touchlink commission protocol and a specific design for the OTA update process, but they are just an example of the way security in IoT is designed today. The Atmel code we reviewed was very well written and documented, but it is extremely difficult to implement complex state machines of such protocols without any bugs. The main problem is in the insecure design of the ZLL standard itself. We believe this will not be the last bug or attack found against ZLL commissioning. While the vendor's main design goal of ease of use is understandable, a better trade-off between usability and security must be made, and the security community and academia should be allowed to take part in the process. The sharp contrast between the open and inclusive manner in which TLS 1.3 standard was designed and the secretive work on the ZigBee 3.0 specification that is still not open to the public, is a big part of the problem.

We believe that in the same manner of the leaked ZLL master key, the OTA updates keys will also be leaked. The reuse of symmetric encryption and signing keys between lightbulbs is a big security risk and it enables attackers to create a chain reaction of infections. Security by obscurity has failed time after time. Working with the security community and academia will probably leads to better alternatives to ZLL commissioning than using a master key, and better ways to protect OTA updates than a shared symmetric key. In our computers and smart phones, software updates are usually protected by asymmetric signatures that validate the origin of the software, or the connection to the update server. There are many solutions that can be found for low cost asymmetric cryptography (for example using hash based signatures). Moreover in the same manner that the chip manufacturers added AES hardware accelerators, a strong requirement in standards like ZigBee will encourage them to add hardware support for asymmetric cryptography. IoT devices are becoming more and more common and affect larger parts of our life. We can learn from history about the importance of good design practices for security protocols and how to implement them. We should work together to use the knowledge we gained to protect IoT devices or we might face in the near future large scale attacks that will affect every part of our lives.

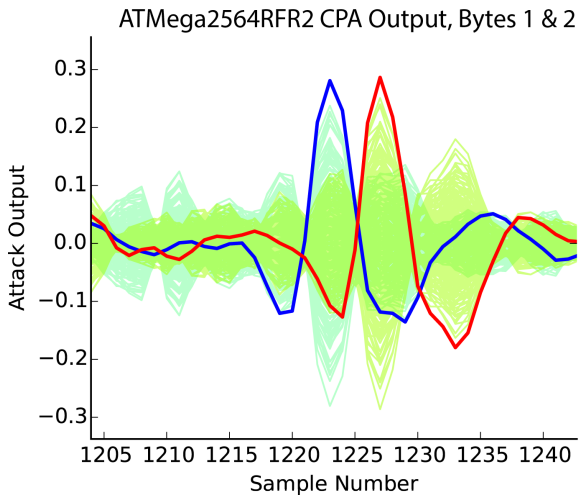

Figure 13. Correct values of the correlation analysis attack for byte 1 (in blue) and byte 2 (in red) compared to all incorrect guesses (in light cyan and green) show both the positive and negative peak we can exploit.

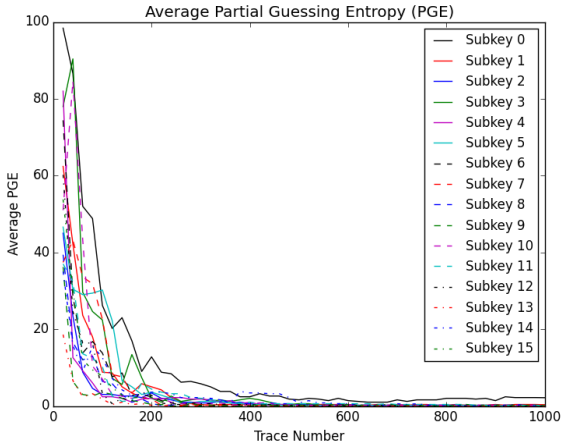

Figure 14. PGE of ATMega2564RFR2 Hardware AES Peripheral – a PGE of 0 indicates that encryption key byte is fully recovered.

Appendix
ATMega2564RFR2 Leakage

The ATMega2564RFR2 was assumed to have a leaky hardware AES engine, as previous work has demonstrated such leakage on the similar ATMega128RFA1 and the Atmel XMEGA devices [13], [14]. We characterized the leakage of the AES-ECB peripheral in the ATMega2564RFR2 device in order to determine the approximate number of traces required with our test setup. In this setup the device was running at 4 MHz[3].

This characterization phase was also required as part of the black-box attack, since we were unsure the exact encryption mode in use, or where such encryptions occurred in the bootloader. The ATMega2564RFR2 device has a unique signature during hardware AES encryptions – we could correlate the known signature with the unknown traces to detect this, but in practice it was even visually obvious (as in Figure 8).

For attacking the AES-ECB mode, a first-order CPA attack was used with the leakage model from [13], [14]. Figure 13 shows the correlation peaks we measured for 2 of the 16 bytes; they have both a positive and negative component, so we combined these peaks to improve the attack efficiency. In addition windowing is required, as a larger (incorrect) peak occurs a few hundred cycles before the correct peak, similar to what is reported in [14].

In practice the window can easily be applied, as from the power signature it is trivial to determine where the AES hardware operation is occuring. Thus with almost no experimentation we can take the window offsets determined from the reference platform (where we control the code and encryption key), and apply them to a CPA attack on the 'black-box' device.

To determine the number of traces we need to capture, we've performed 50K AES-ECB encryptions on our reference platform (which is the same as Figure 7 but with our own firmware loaded where we control the key). To measure the attack success, we are using the partial guessing entropy (PGE) which indicates how many (incorrect) guesses would be required for each key byte based on the information available after processing N traces [35]. A PGE of 0 means that byte of the encryption key was fully recovered. These results are shown in Figure 14, where we have plotted the PGE for $N = 20, 40, 60, \cdots, 1000$.

Note that byte 0 is the most difficult to recover — its PGE falls to zero after about 2000 traces, while the other 15 bytes are recovered with only about 800 traces. As previously proposed if an insufficient number of power traces exist it is instead possible to perform a CPA attack on bytes 1 – 15, and use a brute-force check to recover byte 0 [13], [14]. As we had no limit on the number of traces that could be acquired, we simply acquired around 5000 traces for each block of interest. This made it very likely we could recover the true key without having to enumerate any 'most likely' options.

3. The device normally runs at 16 MHz. We used a slower speed as the SPI flash would not reliably work with the long leads of the bus pirate connected, but clocking the device at 1/4 frequency proved more reliable with the programmer attached.

Acknowledgments

The authors would like to thank our colleagues from Ben Gurion University: Yossi Oren and Omer Shvartz for their help with our initial power measurements, and Prof. Yuval Elovici and Sergey Kosyagin for their help with the drone attack demonstration.

We would also like to thank Dvir Shirman for his helpful insights about DPA and Ziv Menahem for a lot of hard and delicate solder work.

References

[1] B. Krebs, "Hacked Cameras, DVRs Powered Todays Massive Internet Outage," October 2016. [Online]. Available: https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/

[2] Wikipedia, "Percolation threshold — Wikipedia, the free encyclopedia," 2016, [Online; accessed 30-Oct-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Percolation_threshold

[3] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella, "Performance study of ieee 802.15.4 using measurements and simulations," in *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, vol. 1, April 2006, pp. 487–492.

[4] Wikipedia, "Paris — Wikipedia, the free encyclopedia," 2016, [Online; accessed 30-Oct-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Paris

[5] A. Chapman. (2014) Hacking into internet connected light bulbs. [Online]. Available: http://www.contextis.com/resources/blog/hacking-internet-connected-light-bulbs/

[6] N. Dhanjani. (2013) Hacking lightbulbs: Security evaluation of the philips hue personal wireless lighting system. [Online]. Available: http://www.dhanjani.com/docs/HackingLighbulbsHueDhanjani2013.pdf

[7] E. Ronen and A. Shamir, "Extended functionality attacks on iot devices: The case of smart lights," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 3–12.

[8] D.Heiland. (2016) R7-2016-10: Multiple osram sylvania osram lightify vulnerabilities. [Online]. Available: https://community.rapid7.com/community/infosec/blog/2016/07/26/r7-2016-10-multiple-osram-sylvania-osram-lightify-vulnerabilities-cve-2016-5051-through-5059

[9] F. Armknecht, Z. Benenson, P. Morgner, and C. Müller, "On the Security of the ZigBee Light Link Touchlink Commissioning Procedure." [Online]. Available: https://www1.informatik.uni-erlangen.de/filepool/publications/zina/ZLLsec-SmartBuildingSec16.pdf

[10] T. Zillner, "Zigbee exploited - the good, the bad and the ugly," in *Black Hat USA*, 2015. [Online]. Available: https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf

[11] P. Morgner, S. Mattejat, and Z. Benenson, "All Your Bulbs Are Belong to Us: Investigating the Current State of Security in Connected Lighting Systems," *arXiv preprint arXiv:1608.03732*, 2016.

[12] C. OFlynn, "A lightbulb worm?" 2016. [Online]. Available: https://www.blackhat.com/docs/us-16/materials/us-16-OFlynn-A-Lightbulb-Worm-wp.pdf

[13] I. Kizhvatov, "Side channel analysis of AVR XMEGA crypto engine," in *Proceedings of the 4th Workshop on Embedded Systems Security*. ACM, 2009, p. 8.

[14] C. O'Flynn and Z. Chen, "Power Analysis Attacks against IEEE 802.15. 4 Nodes," *COSADE*, 2016.

[15] J. Jaffe, "A first-order dpa attack against aes in counter mode with unknown initial counter," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 1–13.

[16] Zigbee light link. [Online]. Available: http://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbee-light-link/

[17] (2012) Zigbee light link standard version 1.0 - zigbee document 11-0037-10.

[18] Philips, "Philips, 2015 Annual Report," http://www.philips.com/corporate/resources/annualresults/2015/PhilipsFullAnnualReport2015_English.pdf, 2016.

[19] (2014) Zigbee over-the-air upgrading cluster version 1.1 - zigbee document 095264r23.

[20] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," RFC 3610, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc3610.txt

[21] R. Housley, "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)," RFC 4309 (Proposed Standard), Internet Engineering Task Force, Dec. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4309.txt

[22] D. McGrew and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)," RFC 6655 (Proposed Standard), Internet Engineering Task Force, Jul. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6655.txt

[23] "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," pp. 1–320, Sept 2006.

[24] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology – CRYPTO 99*. Springer-Verlag, 1999, pp. 388–397.

[25] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems – CHES 04*. Springer-Verlag, 2004, pp. 135–152.

[26] J. Wright, "Killerbee: practical zigbee exploitation framework," in *11th ToorCon conference, San Diego*, 2009.

[27] T. Goodspeed, S. Bratus, R. Melgares, R. Speers, and S. W. Smith, "Api-do: Tools for exploring the wireless attack surface in smart meters," in *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE, 2012, pp. 2133–2140.

[28] C. OFlynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[29] Atmel. Atmel AVR2058: BitCloud OTAU User Guide.

[30] T. Instruments. Crypto-Bootloader (CryptoBSL) for MSP430FR59xx and MSP430FR69xx MCUs.

[31] N. Hanley, M. Tunstall, and W. P. Marnane, *Unknown Plaintext Template Attacks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 148–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10838-9_12

[32] (2012) Zigbee specification- zigbee document 053474r20.

[33] C. O'Flynn, "Message denial and alteration on ieee 802.15. 4 low-power radio networks," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5.

[34] A. Wilkins, J. Veitch, and B. Lehman, "Led lighting flicker and potential health concerns: Ieee standard par1789 update," in *2010 IEEE Energy Conversion Congress and Exposition*, Sept 2010, pp. 171–178.

[35] J. Massey, "Guessing and entropy," in *Proceedings of IEEE International Symposium on Information Theory (ISIT '94)*, 1994, p. 204.