# Analyzing the Security of Companion Apps of Popular Smart Home Devices

Bradley Helliwell
*Department of Computer Science*
*Colorado School of Mines*
Golden, CO, USA
bhelliwell@mymail.mines.edu

Mengxia Ren
*Department of Computer Science*
*Colorado School of Mines*
Golden, CO, USA
mengxiaren@mymail.mines.edu

Ty Christensen
*Department of Computer Science*
*Colorado School of Mines*
Golden, CO, USA
tchristensen@mymail.mines.edu

Thien Ngo Le
*Department of Computer Science*
*Colorado School of Mines*
Golden, CO, USA
thienngole@mymail.mines.edu

***Abstract*— An IoT device's cryptographic strength is only as strong as its weakest endpoint [1]. This gives rise to concern about the cryptographic strength of the mobile applications associated with IoT devices. This project aims to use reverse engineering tools, like the NSA's Ghidra, to reverse engineer the companion applications of smart home devices and reveal information about how they communicate with their corresponding device. We then identify and categorize cryptographic strength and weakness that are introduced by the companion mobile applications of these devices.**

***Keywords—IoT; internet of things; smart home devices; cryptography; vulnerabilities;***

## I. Introduction

There are lots of communication protocols between smartphones and computers, such as GPS, GSM, TCP, UDP, SSL, TLS, HTTP. These protocols usually use some cryptographic algorithms to encrypt their sensitive information. However, there is too much attention on analyzing smartphones and computers. Since IoT devices have also been an important part of the Internet. IoT devices are becoming increasingly popular by the day; Research companies have predicted that IoT will grow to 26 billion units in 2020 [2]. It is important to analyze communication protocols used by IoT devices.

Internet of Things, or IoT refers to a system of internet-connected devices that have the ability to exchange data. These devices provide us with many advantages, such as having smart homes and smart cities. However, there are many challenges and issues for these devices, such as power consumption of devices, limited battery, memory space, performance cost, and security in the Information Communication Technology network [6]. There are two basic types of communication, one is based on HTTP and the other one is based on events. HTTP-based communication is usually implemented through TCP, while event-based communication is based on UDP, such as DDS, CoAP and MQTT. Furthermore, there are some security protocols such as DTLS, TCG and SMACK in IoT. How do these protocols transmit data? Do they encrypt transmitted data and is that encryption secure? These are key problems for security of IoT devices and also the motivation behind our project.

Our goal is to analyze these protocols to see what encryption they use and if the encryption can be cracked and offer some counter measurements.

## II. Approach

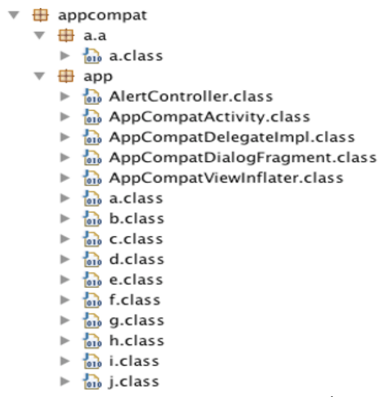
### A. Key Idea

Since the IoT lab cannot be used, we cannot set up an IoT virtual environment and we cannot capture communication packets between IoT devices directly either. Instead, we will analyze these IoT mobile apps to see what protocols they use to communicate with IoT devices and how these apps deal with communication packets sent from IoT devices. By doing this, we can know how these IoT protocols work and whether their encryptions are secure.

### B. Methodologies

First, we collected mobile application packages of the devices we want to analyst (Kasa Smart, August Home, Nest App, Ring). We used APKPure to collect these packages; it is one of the leading websites in the smartphone software industry that allows its users to download android app packages.

Next we reverse engineer collected apps for code analysis. Most of the time, the app's source codes are obfuscated by the developer before it is deployed. The purpose of obfuscation is to make something harder to understand, usually for the purposes of making it more difficult to attack or to copy.



```
appcompat
  a.a
    a.class
  app
    AlertController.class
    AppCompatActivity.class
    AppCompatDelegateImpl.class
    AppCompatDialogFragment.class
    AppCompatViewInflater.class
    a.class
    b.class
    c.class
    d.class
    e.class
    f.class
    g.class
    h.class
    i.class
    j.class
```

Figure 1

Figure 1 shows how obfuscation could make it harder to understand or replicate a given source code if it is reverse engineered. All the classes' names in figure 3.1 are labeled as a, b, c so it is meaningless to any external entities. However, obfuscation is not a strong control such as employed encryption, it is more like an obstacle. Obfuscation, like encoding, can often be reversed by using the same technique that obfuscated it. So, it is important to choose the right tool to decompile the android application package for code analysis because the obfuscation may cause some difficulties for the analysis process. We tried several different tools for the decompiling task. We observed that each tool has its own pros and cons. Tools like Ghidra work very well on decompiling apps that are written in C or C++ but not really good with Java and Python apps. On the other hand, APKtool works well for Java apps but not C or C++. But APKtool decompiles the .apk file into smali source code which is not a human readable format, other tools like Dex2jar and JD-GUI need to be used to convert smali files to java files to make it easier for analysis. From our experience, Ghidra works best for decompiling apps written in C or C++, the combination of APKtool + Dex2jar + JD-GUI work best for java and python apps.

Lastly, we performed code analysis. During the code analysis, we will find parts that deal with communication with IoT devices. If we find what encryption algorithm they used to encrypt/decrypt data before they transmit a message, we will research that encryption algorithm for known vulnerabilities. If we find what protocol they used to transmit data, we will figure out what encryption algorithm is used for the protocol. We will then analyze known vulnerabilities for that encryption algorithm. Our group analyzed the following IoT devices' mobile applications. Kasa Smart Plug, NEST Dropcam, August Smart Lock and Ring Doorbell. The Kasa Smart Plug is a smart in-wall outlet whose application allows the user to control power output to the outlets. The NEST Dropcam is a security camera that uses motion detection to alert via the NEST app. The August Smart Lock allows the user to lock and unlock a door as well as grant others permission to lock and unlock that door via the app. The Ring Doorbell functions as both a security camera and communication device. Through its app, the user can talk to someone standing at their doorway and alerts the user if motion is detected in front of their doorway.

For each of these apps we will be judging their security. If an app breaks one of the following six rules they simply cannot be secure [4].

**Rule 1:** Do not use ECB mode for encryption.
**Rule 2:** Do not use a non-random IV for CBC encryption.
**Rule 3:** Do not use constant encryption keys.
**Rule 4:** Do not use constant salts for PBE.
**Rule 5:** Do not use fewer than 1,000 iterations for PBE.
**Rule 6:** Do not use static seeds to seed SecureRandom().

Breaking one of these rules creates a big security vulnerability. This is typically because they go against best practice and create an issue that a specific encryption mode has. For example, with ECB unless messages are very small and unique there will be an issue. This is because with ECB if the plaintext is identical it will encrypt to identical cipher text [4]. Similar specific issues occur with PBE and CBC as well. Using constant encryption keys is a bad idea in general as the key needs to remain private and unknown. Using static seeds for something that is supposed to be random creates an issue because that random item is no longer truly random. For these reasons if any of these six rules are broken, the encryption for the app is deemed insecure [4].

In addition to the rules listed above, we will also take other factors into consideration including how well sensitive data is stored, programming issues, network communication and other known vulnerabilities. Evaluation of these general cyber security practices will be used in conjunction with the rules listed above to evaluate these applications. So, an application might have strong cryptographic practices, but cannot be considered secure due to other vulnerabilities introduced in the application. Specifically, we will use the guidelines outlined by the Your Things Project which scores IoT devices on their overall cyber security strength [5].

## III. Application analysis

### A. *Kasa Smart for Kasa Smart Plug*

Kasa Smart is a mobile application developed by TP-Link company. It allows users to add, configure, and control users' connected TP-Link devices from anywhere. For example, you can use Kasa Smart app to schedule your Kasa Smart plug to turn on or off according to your schedule. we used APKtool, Dex and JD-GUI to compile this application.

Based on what we observed, Kasa Smart has their own encryption function that uses DES encryption algorithm. It uses HTTPS and CA pinning protocol for cloud communication. Unfortunately, the initial vector is hardcoded in the encryption function which makes the encrypted message breakable. We also found that the app has a sort of wifi scanning function that looks for the wifi network that has the same name as the network the smart plug is connecting to. So we suspect that the app and the smart plug have local communication. After a quick search, we confirmed that there is local communication between the app hosting device and the smart plug. According to the related work done by a security team, SoftScheck [7], Kasa Smart also supports local communication using TCP protocol on port 9999. But there is no authentication implemented on the local communication. This means that everyone on the same network with Kasa devices could be able to control the devices. The team also found that the encrypt/decrypt key (171) is also hardcoded in the device's firmware.

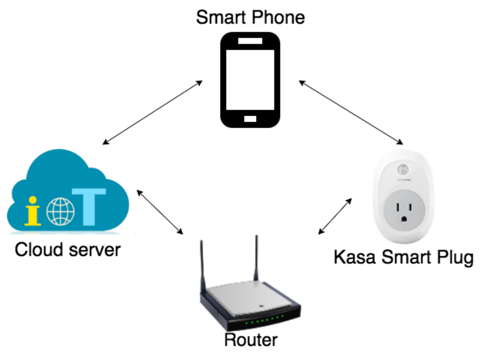


Figure 2

Figure 2 illustrates all the communication of the Kasa Smart app and its smart plug.

## B. *NEST for NEST Drop Cam*

The NEST app is a hub for all the IoT devices Google Nest offers. The app will allow you to control items such as your thermostat, alarm system, and indoor/outdoor dropcams. Through the app you can change the temperature of your home, arm and disarm your security system, and monitor video from your drop cams with a feature that allows you to talk through them as well. The app holds up to 30 days of video history and can send frequent face alerts.

Overall, the app's security is very strong. It uses multiple encryption types including 128-bit and 256-bit AES typically using CBC or GCM modes. These security modes are secure because brute force attacks are theoretically impossible due to the amount of time needed to crack the key. GCM is usually more secure than CBC but also more expensive so it is important to pair it with CBC to be the most secure. The app uses 2048-bit RSA private keys for its key exchange, which is the community standard. The only possible vulnerability that was discovered is the fact that some of their passwords and router information may be stored in plaintext. This includes the network password as it is needed for the IoT devices to work.

## C. *August Home for August Smart Lock*

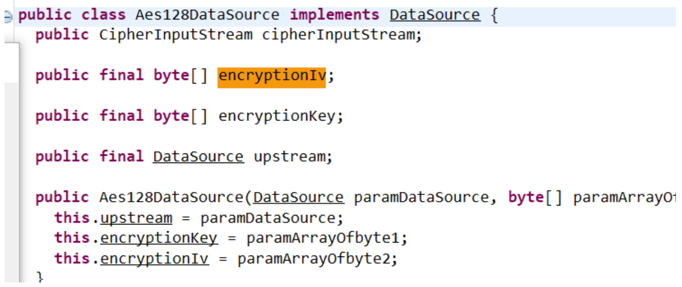
The August Smart Lock is used to gain more control over who is able to lock or unlock a door. Some of the features offered by the mobile companion application include allowing a user to unlock and lock their door remotely, granting access to others during a specific timeframe and closely monitoring who has access to their home. Ghidra was used to decompile this application because its source code was written in C.
The August Smart Home app generally has fairly good encryption techniques and did not violate any of the rules or best practices this project sought out to evaluate. However, when communicating with the August IoT platform, the mobile app uses RC4 encryption. This raises concerns because there are known vulnerabilities regarding statistical biases in the RC4 cipher. The attack laid out in the paper *All Your Biases Belong To Us:Breaking RC4 in WPA-TKIP and TLS* shows that about 9227ciphertexts are needed in order to perform this attack [3]. This shows that, the way August has implemented their encryption, an attacker could not get that many ciphertext. So, their use of RC4 seems secure but a more secure cipher like AES would provide more security.

## D. *Ring for Ring Doorbell*

It is used to help people monitor situations around their home. When people ring your doorbell and you are not at home, you can audio with them remotely through this ring app. This means the only IoT device this application interacts with is a smart ring bell. We used APKtool, Dex and JD-GUI to compile this application.

*Communication between Ring App and IoT devices.* After decompiling the app, we found that this app contained *com.amazon.identity.auth.device* package and used http/https protocol to login in amazon to do device identity authentication. This means the app does not directly communicate with the smart ring bell. Instead, it communicates with smart devices through an IoT platform which is designed by Amazon, called AWS IoT platform. This IoT platform allows internet-connected devices to connect to the AWS Cloud, then the applications connected in this cloud platform can interact with these IoT devices which also connect in this platform.

*Data encryption.* we found that this app will encrypt data source and data before it sends them out. Then encryption is a combination of AES algorithm, CBC cipher mode and PKCS7PADDING.The evidence we found in decompiled code is shown as Figure 3.

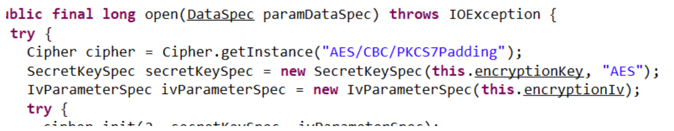


```
public class Aes128DataSource implements DataSource {
  public CipherInputStream cipherInputStream;

  public final byte[] encryptionIv;

  public final byte[] encryptionKey;

  public final DataSource upstream;

  public Aes128DataSource(DataSource paramDataSource, byte[] paramArrayOf
    this.upstream = paramDataSource;
    this.encryptionKey = paramArrayOfbyte1;
    this.encryptionIv = paramArrayOfbyte2;
  }
```

Figure 3

AES is a symmetric encryption algorithm and in practice, the app used a 128-bit key to encrypt data. The AES128 algorithm is secure enough to protect data, since the brute-force attack needs thousands of years to crack it.

In CBC cipher mode, before being encrypted, a plaintext block will be XORed with the previous ciphertext block. By doing so, each ciphertext block will be impacted by all plaintext blocks processed up to it. An initialization vector is used in the first block. One of the security requirements for CBC is initialization vector IV should be generated randomly otherwise with a predictable IV, it will be possibly cracked by chosen plain text. In decompiled code, we found that the app used SecureRandom class to generate IV for CBC encryption. The evidence we found is shown as Figure 4. Without a static seed given, SecureRandom class will generate IV randomly. This usage satisfies **Rule 2:** Do not use a non-random IV for CBC encryption and **Rule 6:** Do not use static seeds to seed SecureRandom(). We can say the data encryption in this app is secure.



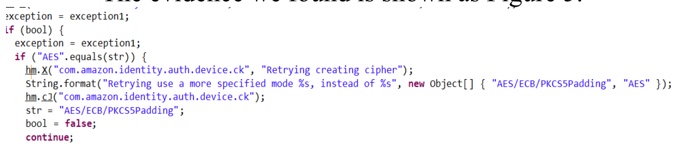
```
ıblic final long open(DataSpec paramDataSpec) throws IOException {
 try {
   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS7Padding");
   SecretKeySpec secretKeySpec = new SecretKeySpec(this.encryptionKey, "AES");
   IvParameterSpec ivParameterSpec = new IvParameterSpec(this.encryptionIv);
   try {
```

Figure 4

Since the app did not communicate with the IoT device, we can only get very little information from decompiled code of the application. For obtaining a more accurate analysis, we also read the introduction and related documents of the AWS IoT platform.

Based on codes and documents, we learned that one common protocol used between IoT devices and AWS IoT platform is MQTT. MQTT is an insecure protocol and it does not require devices to authenticate to servers. Thus, communication between a device and AWS IoT is protected through X.509 certificates. X.509 is a standard defining the format of public key certificates.

We found that in the app, the device identity authentication is completed based on a combination of AES algorithm, ECB cipher mode and PKCS5PADDING. We assume that for IoT devices, they also use the scheme for identity authentication. The evidence we found is shown as Figure 5.



```
xception = exception1;
if (bool) {
  exception = exception1;
  if ("AES".equals(str)) {
    hm.X("com.amazon.identity.auth.device.ck", "Retrying creating cipher");
    String.format("Retrying use a more specified mode %s, instead of %s", new Object[] { "AES/ECB/PKCS5Padding", "AES" });
    hm.cJ("com.amazon.identity.auth.device.ck");
    str = "AES/ECB/PKCS5Padding";
    bool = false;
    continue;
```

Figure 5

In ECB mode, the message will be divided into several blocks, and each block will be encrypted separately. This method lacks diffusion since there is no feedback. Furthermore, in this mode, same plaintext blocks will be encrypted as same ciphertext blocks. ECB is not able to hide data patterns well.

The usage of ECB violates **Rule 1:** Do not use ECB mode for encryption. It is not as secure as CBC. However, based on the app analysis, ECB is not used to encrypt data, instead it is used in identity authentication to create a cipher key for digital signature. AES128 is secure enough to protect data and it is hard to be cracked by brute force attack. Therefore, we think this usage of ECB is also secure.

## IV. Limitations

Our goal is to learn the communications in IoT devices. However, analysis only based on decompiled code of IoT applications is incomplete. And we cannot verify whether our analysis reflects the real situation of communications between IoT devices and applications.

Sometimes it is difficult to analyze how it communicates with IOT devices since the decompiled code is not logical. For example, in the decompiled code of Ring app, there is a function A(). If you cannot understand what this function is used for from the code in its body, you cannot have any idea from its function name either.

The way of communications between IoT devices and the IoT application will also impact the analysis results. If the IoT application communicates with IoT devices directly, for example, Ring Doorbell app, we can obtain the protocols or encryption algorithms used in IoT devices, since the application and the device will use the same rules during their communications. However, when the application and IoT devices communicate through an IOT platform, it is difficult to know what the real case of communication between IOT platform and IOT devices is. We can only track the real situation of communications between applications and the IoT platform and can only assume the communications between IoT devices and the IoT platform.

Since we only can decompile and analyze manually, we only analyzed 5 IoT applications. The security issues of these IoT devices and IoT applications we found is only a lower bound. There must be more security problems in the IoT environment.

## V. Future Work

In the future, we will try to find some tools to help us decompile the apk and target encryption snippets in decompiled code automatically. Thus, we can analyze more IoT applications. Based on a bigger dataset, it is possible for us to find more security issues in IoT.

## VI. Conclusion

In this project, we aim to find some security issues of encryption algorithms used in IoT protocols and IoT devices by analyzing IoT applications. We decompiled applications by some decompiled tools, such as apktool, JD-GUI and dex. After obtaining decompiled code, we searched the keywords related to encryption in decompiled files, such as “AES”, ”encryt”, “crypto” and then analyzed these encryption related snippets.

From our analysis, we found that some applications did not encrypt their data and send it directly. Some applications used outdated encryption algorithms, e.g. RC4, to encrypt their data. While in some other applications, they used insecure encryption mode, e.g. ECB, however, they combined it with a secure enough encryption algorithm, e.g. AES128.

For improving the security of communications in IoT, we recommend that developers should encrypt their data before sending it out with a secure encryption algorithm or a secure cipher mode. When they implement the cipher mode, they need to keep the six rules we listed in mind.

## References

[1] ManagementMania, “Weakest Link Principle,” ManagementMania.com, 17-Feb-2018. [Online]. Available: https://managementmania.com/en/weakest-link-principle. [Accessed: 12-Feb-2020].

[2] How Big Is the Internet Of Things? How Big Will It Get? https://paxtechnica.org/?page_id=738

[3] K. Sha, W. Wei, T. A. Yang, Z. Wang, and W. Shi, “On security challenges and open issues in Internet of Things,” Future Generation Computer Systems, 07-Feb-2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17324883. [Accessed: 13-Feb-2020]

[4] Egele, Manuel, David Brumley, Yanick Fratantonio, and Christopher Kruegel. “An Empirical Study of Cryptographic Misuse in Android Applications.” Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS 13, 2013. https://doi.org/10.1145/2508859.2516693.

[5] “YourThings Scorecard,” YourThings Scorecard. [Online]. Available: https://yourthings.info/. [Accessed: 31-Mar-2020].

[6] Singh, Saurabh & Sharma, Pradip & Moon, Seo & Park, Jong. (2017). Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. Journal of Ambient Intelligence and Humanized Computing. 1-18. 10.1007/s12652-017-0494-4.

[7] “Reverse Engineering the TP-Link HS110,” https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/, SoftScheck GMBH, 2018.