DOLO: Data Over Light Onboarding for Smart Smoke Detectors

Ahmad Charba

University of Ottawa
Canada
achar238@uottawa.ca

Abstract

As IoT devices are getting more and more sensors as the technology progresses, older and unreliable onboarding schemes such as SoftAP are beginning to show their age. These schemes often do not take into account the realities of modern peripherals and security standards, often causing frustrations for the end-user. But, new onboarding protocols taking advantage of these new sensors are being developed to replace their older counterpart. This paper proposes one such protocols, Data Over Light Onboarding, for use in smart smoke detectors equipped with an ambient light sensors. We provided a protocol based on the idea of sending encrypted data through a phone's flash, analyzed the properties of this scheme and compared it to SoftAP. We found that this protocol solves many of the concerns users have with SoftAP, while providing good security guarantees due to its physical nature.

Keywords— IoT, data over light, smoke detector, photovoltaic

1 Introduction

Internet of Things is on the rise, and with it many technologies are gaining new capabilities and added connectivity. Among them are smoke detectors, an essential device that ensures the safety and well being of residents, clients and workers in households and buildings.

Smart smoke detectors are getting more and more useful as the technology matures, and they provide many advantages over their non-smart counterparts. Among other things, they can alert the homeowner or any relevant person of a fire alarm while they aren't home, send warnings when the battery needs to be replaced, often provide more detailed warnings than a simple alarm through voice integration, and make false alarms simpler to handle for the end-user. They also allow for new possibilities in the future such as providing information automatically to first responders [1] and informing neighbours of the alarm. With these

additional capabilities, smart smoke detectors would benefit the public and society as a whole.

However, such devices tend to have the drawback of being pricier and more difficult to set up as their counterparts. Often, they will use SoftAP for onboarding, which requires additional hardware, increasing cost and attack surface, and sometimes isn't as painless as one would hope, with 20% of users failing to use SoftAP to set up a device [2].

To remediate this issue, we propose DOLO (Data Over Light Onboarding), a novel onboarding protocol for smart devices that possess light sensors, with smoke detectors as an example where this technology can be used. We will be using the Google Nest Protect 2nd generation to illustrate that.

DOLO provides a simple, low cost, robust and secure way of sending WiFi credentials by using already existing ambient light sensors in some photoelectric smoke detectors and a smart phone with a working camera and flash. It makes use of symmetric encryption (AES) using a key found on the backside of the detector and an app that communicates instructions to the user and sends the data through the smartphone's camera flash.

2 Outline

In Section 3, we will discuss the Google Nest Protect's hardware and current onboarding scheme. In Section 4, we will specify the required hardware for the DOLO protocol to function. In Section 5, we will explain in detail the DOLO protocol, including how the user interacts with their phone and Nest and how the network information is encrypted, partitioned, error corrected, encoded and then sent to the Nest. In Section 6, we compare DOLO to SoftAP in terms of security and usability. We state the different issues commonly had with SoftAP and discuss how DOLO handles the situations. In section 7, we discuss where DOLO can be used (outside of the Google Nest Protect).

3 Google Nest Protect

The Google Nest Protect is a smoke detector that comes with many features that aren't available with conventional smoke detectors such as WiFi connectivity, ambient light detection, carbon monoxide sensing and temperature change detection.

3.1 Nest Protect's Hardware

It is equipped with the following sensors [5]:

Photoelectric sensor

Electrochemical carbon monoxide sensor

Heat sensor

Humidity sensor

Ambient light sensor

Microphone

Ultrasonic sensor

It has a front-facing LED, a button as well as a speaker. This allows it to notify the user about its state and give them precision about alarms, such as the location of the alarm's source. This also grants the user peace of mind, as the user can know if the speakers and sirens are functional if the LED stays green. The colour will change in the case where they don't pass an automatic test.

As for networking, it is equipped with chips for [6] [10]:

- Wi-Fi: 802.11b/g/n 2.4 GHz (muRata Type ZX WiFi Module)
- Wireless Interconnect: 802.15.4 at 2.4 GHz (EM351 ZigBee SOC)
- Bluetooth Low Energy (Unknown Chip)

The MCU, networking chips and sensor models have not been made available on Google's data sheets, though a tear down of the device [10] allowed us to know that it has for MCU the Freescale Kinetis K60 (MK60DN512VLL10), which has the following relevant capabilities [11]:

- 32-bit ARM Cortex-M4 core
- Hardware Cryptographic Acceleration Unit (Supports DES, 3DES, AES, MD5, SHA-1, and SHA-256)
- Maximum frequency of 100 MHz
- Supports DSP instructions
- 512kb of flash memory
- 256kb of SRAM
- FlexMemory (expendable RAM/memory size)

As there is no information about the ambient light sensor used in the Google Nest Protect, we will be working under the conservative assumption that it is able to measure a peak in light every 0.03 seconds, or at a speed of about 33 Hz [8]. We also suppose that the phone's flash LED is able to achieve this speed, which should be possible on most modern phones.

We will also work under the assumption that it uses SoftAP to connect to the users' WiFi network, or at least a scheme that shares the same drawbacks, since the documentation provided by Google explains that the device creates a "temporary 'ad hoc' WiFi network" [9] during the set-up phase, and has the same potential errors as SoftAP. [3].

3.2 Nest Protect's Current Onboarding Scheme

The onboarding process goes as follows: the user must first have the Nest App on their phone. Then, the user having their phone in proximity to the device, with Bluetooth and WiFi turned on. Google states that some devices may require the user to turn off cellular data, as well as automatic network switching (where the phone would automatically switch from a slow WiFi network to a faster one) [3]. Afterwards, the user must add a new Nest Protect on their app, which will give the user instructions for the continuation of the onboarding process.

The user has to scan the Quick Response code (QR code) found on the back of the device, and then turn on the Nest Protect by removing the battery tab. The device will then (based on our assumption) turn on a SoftAP WiFi hotspot, which the app will make the user's phone connect to. [4] Afterwards, the app will prompt the user to choose a WiFi network and to specify the credentials [3]. The app will then send those credentials to the Nest Protect using the SoftAP WiFi hotspot and the Nest will be able to connect to the network using those credentials [3], ending the process.

This process relies on the user knowing how to deactivate cellular data and automatic network switching, which may not be an obvious process for most end-users. In addition to that, SoftAP adds cost, attack surface and points of failure, by requiring the addition of hardware and multiple sensitive protocols.

4 Required Peripherals for DOLO

In order to execute DOLO, the user requires the same peripherals as the original SoftAP based protocol, that is: a WiFi router, the Google Nest Protect and a phone or tablet with the Nest app installed. The device needs to have the following features:

- A functioning camera (to be able to read a QR code)
- A functioning camera flash

The Google Nest needs, just like the in previous scheme, to have a QR code in the back. The AES symmetric encryption key will be contained within that QR code and will be used to communicate with the Nest.

We assume the WiFi router uses a fixed password.

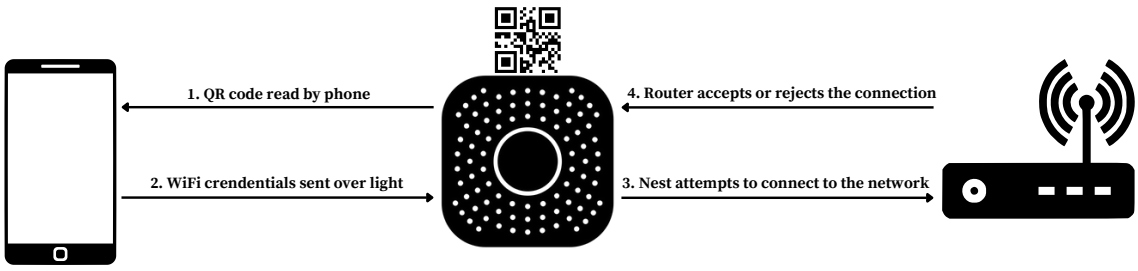

Figure 1: Communication Flow Diagram

5 The DOLO Protocol

An overview of the protocol can be found in figure 1.

5.1 Reading of the QR code

The first step of the DOLO protocol is for the user's phone or tablet to read the QR code in the back of the smoke detector.

This QR code contains both the device's ID (14 alphanumerical characters [17], which can be encoded in 112 bits) and a 192 bit AES key for symmetric encryption, for a total of 304 bits of data. This can be encoded as mixed bits in a version 3 (29x29 modules) QR code with medium correction level [13] [12]. A higher correction level wouldn't be necessary [14], as the back of the device is very unlikely to be dirtied as it will be facing a ceiling after installation.

The reading will be done in the Nest app, in the same manner as it is in the current scheme. The app can access the phone's camera and prompt the user to scan the QR code on the back of the device. Thanks to QR codes built-in error correction and the low risk of the QR code getting dirty, there is little risk of error in this process. If an error were to occur, it can be handled during step 2 by making the Nest's front LED blink and/or change colour, and by announcing the error message using the Nest's speaker and a prerecorded voice line.

5.2 Beginning of the Data Transfer

This is the second step, and also where the process diverges most from the original protocol. The phone now has the symmetric encryption key, and will prompt the user to select the WiFi network the device should connect to. It will then ask the user to type the WiFi password. The Network's SSID and the password are to be encrypted and encoded, and then sent to the Nest Protect using the phone's camera flash. The user will be prompted to press on the Nest's button if the WiFi network is not hidden, prompting the Nest to scan for accessible networks and holding them in its memory.

Now comes the main difference between the original scheme and the one we propose. The data, instead of being sent through a WiFi hotspot, will be sent from the phone camera's flash to the ambient light sensor of the Google Nest Protect.

5.2.1 Motivation for the Choice of Light Sensor

We are using the Nest's ambient light sensor, rather the photoelectric sensor used for smoke detection, mainly for mechanical and engineering reasons. Smoke detectors operate in a manner that requires them to close off the photoelectric smoke sensor from outside light. When entering the chamber, the smoke refracts the light emitted by the light emitter in the smoke sensor, making the light sensor activate. Opening this up for onboarding would then pose safety risks as dust and outside light could falsify the detector's readings.

On the other hand, the ambient light sensor is easily accessible from the front, as its utility is to sense the amount of light coming from the exterior. This makes it a perfect

target for Data-over-Light (DoL).

5.2.2 Motivation for the Choice of Light Emitter

The reason we use the phone's flash is due to to it's high light intensity and it's position (on the back of the phone). This scheme could be made to work with light coming from the phone's screen, but that would make it impossible for the user to receive information from said screen regarding the progress of the onboarding; the phone would be facing the wrong direction.

The additional light intensity also adds a better reliability, as the sensor could have a higher actuation point/more separate ranges of light intensities when decoding the DoL.

5.3 Encryption

The 192 bit AES key found in the QR code can be hard coded in the smoke detector's memory, allowing it to decrypt the incoming traffic with little delay. This is thanks to its micro-controller's Hardware Cryptographic Acceleration Unit and the small amount of data that will need to be transferred. The WiFi SSID consists of 32 characters, and the WiFi password consists of up to 63 characters [15]. This means the data can be sent as seven 128bit AES blocks (¡824 bits), with the 4 last blocks reserved for the password; in the 3 first blocks, 128 bits will come from the password, and in the last block, the last 120 bits of the password will be encrypted, along with the 8 last bits of the device ID.

We can reduce this number to 5 blocks in the case of a WiFi network that's not hidden, by sending the hash of the SSID rather than the full SSID. In this version, the user's phone will calculate the SHA-256 hash of the SSID and truncate it, using only the last 128 bits of the hash (fitting inside of a single block). This will be compared by the Nest to the last 128 bits of the hashes of the WiFi networks' SSIDs, which it found through scanning the network. It is still near-impossible for the last 128 digits of the hash to correspond to the wrong network, due to 128 still being a large number of bits allowing for a good collisions protection [18]. Being that we are not using this hash for security purposes, we do not require the full robustness of the 256-bit hash.

These 5 encrypted blocks will then be encoded by the phone so that they can be sent over light to the Nest.

5.4 Error Tolerance

To those blocks, the phone will add Reed–Solomon error correction [**?**]. As an example, we will use 4-bit symbols, meaning our blocks will consist of 32 symbols each. For the 5 blocks, we will have 160 symbols in total. Because the limitation in the number of symbols that the Reed–Solomon scheme can handle is $2^m + 1 = 17$ [**?**], we will be using 12 blocks of data. For the first 2 blocks of data, we will use error correction $RS(n = 17, k = 15)$, allowing for 1 symbol to be corrected.

$m = 4$ and $t = 1$

$0 < k < n < 2^m + 2$

$0 < 15 < 17 < 2^4 + 2 = 18$

$2t = n - k$
$2 * 1 = 17 - 15$

The other 10 blocks of data with error correction $RS(n = 17, k = 13)$, allowing for 2 symbols to be corrected. This gives us a total of 160 symbols, which is exactly what we need.

$m = 4$ and $t = 2$
$0 < k < n < 2^m + 2$
$0 < 13 < 17 < 2^4 + 2 = 18$
$2t = n - k$
$2 * 2 = 17 - 13$

We can fit the 4 password blocks (B1, B2, B3, B4, 512 bits) inside of the 10 error correction blocks with a 2 symbol tolerance ($10 * 13 * 4 = 520bits$), leaving 8 bits (2 symbols) empty. We can then fit the SSID hash block (B5, 128 bits) in the two error correction blocks with a 1 symbol tolerance ($2 * 15 * 4 = 120bits$) and in the last 8 bits of the 2 symbol tolerance blocks. The password blocks can then be partitioned in the following manner, as seen in figure 2:

- B1[0:51]
- B1[52:103]
- B2[0:51]
- B2[52:103]
- B3[0:51]
- B3[52:103]
- B4[0:51]
- B4[52:103]
- B1[103:127], B2[103:127], B4[103:106]
- B3[103:127], B4[107:127], B5[120:127]
- B5[0:59]
- B5[60:119]

The tolerance for the SSID isn't as crucial to the Nest as that of the password, as the SSID could be sent first, allowing the Nest to compare it to the list of networks it found before having the password. This means the process could be restarted without sending the 10 full password error-correcting blocks.

We end with $12 * 17 = 204$ 4-bit symbols, or 816 bits of data to be sent. The distribution of the data in the blocks is illustrated in figure 2.

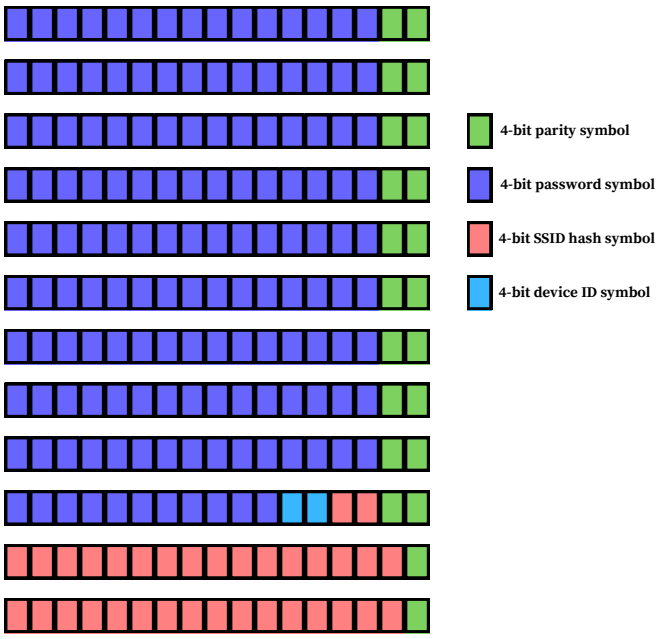

Figure 2: Blocks of Data with Error Correction

5.5 Encoding

This is the start of step 2: sending WiFi credentials sent over light (figure 1).

The phone will now prompt the user to place the phone face-up on the smoke detector, and the user will press an on-screen button to start the data-over-light transfer. The phone will start by turning on the flash at its maximum strength for 0.5 seconds or 2 times 0.20 seconds, with a 0.1 second delay. This will tell the Nest to begin listening for the following data and let it know if it will be receiving the hash of the SSID (non-hidden network) or the actual SSID (hidden network). We will be assuming the former in the next parts, with the latter being a simpler form of this scheme.

The information that will be sent to the nest can be encoded in one of the following ways, with more information on the ambient light sensor required for the decision.

5.5.1 On-Off encoding

In this encoding, the phone will simply send the information by turning on or off the camera flash at 30Hz. For every bit in the data, the phone will turn on the camera flash if it's a 1, or turn it off if it's a 0.

The main issue with this approach is that it would be slower: transferring 816 bits of data at a rate of 30 bits per second would take around 27 seconds to complete, which is a bit longer than what an end-user would expect to wait.

The second issue is that the Reed–Solomon error correction is going to lose in effectiveness, as 1 bit being wrong in 11 different symbols would be beyond the correction threshold of the function.

5.5.2 m-level encoding

In this encoding, the phone will send the data using as many levels of luminosity as there are possible symbols. For each symbol in the data, the phone will turn the camera flash at the level of lightning represented by that symbol, at a potentially slower rate of 15Hz.

This have the advantage of making the Reed–Solomon error correction effective, as symbols will be sent independently and as a whole.

Another advantage of this encoding is the speed, as, if we still assume a 15 Hz rate, using 8-bit symbols, the phone will be able to finish sending the 204 4-bit symbols in less than 14 seconds, or less than 7 seconds if we can execute this at 30Hz.

There are however a few disadvantages to this scheme, which is that it requires a lot of precision from both the sensor and the camera flash. In this case, the phone must have a flash able to shine 16 distinct levels of luminosity.

It would also require a calibration beforehand, as each phone's flash has a different intensity. This could be done after the 0.5 second initiation flash, by having the phone go through all of the 2^m symbols in order, so that the nest can save the value for each of them in lumen. This would take a considerable amount of additional time, as there are 16 possible symbols to go through. The calibration would then add about 1 full seconds on 15 Hz, or about 0.5 seconds on 30 Hz. Smaller symbols than 4-bit would be impractical because it would make the maximal size of the blocks that the error correction can handle very small $(2^m + 2)$.

5.6 Reception of the Data

While the phone is sending the data through the camera flash, the Nest has to decode it, check for errors, decrypt it and make use of it to attempt to connect to the network. The blocks will be sent starting with the 8th (the one containing the device ID), then the 9th and 10th, and finally all of the others. This will grant us a few useful properties.

5.6.1 On the Reception of a block

Upon the reception of a block, the first thing the Nest Protect will do is to use the parity bits to check for errors and attempt error correction if required.

5.6.2 Reception of the first three blocks

Once the Nest decodes its first three blocks, it can decrypt the SSID hash's second half using its encryption key (while still receiving the next bits of data through interrupts). It can then check that the user sent the right SSID by comparing the hash obtained with the second half of the hashes of the networks it found earlier. This can be accelerated using the microcontroller's Hardware Cryptographic Acceleration Unit, which supports SHA-256.

If it cannot be found, the following two errors could have occurred: the encryption key used was not the right one, or the network is not visible to the Nest. Either way, an error message must be given (a sound could be sounded and picked up by the phone's microphone [**?**]) so that the process is restarted using the full SSID.

If it can be found, the Nest will simply keep listening to the next blocks.

5.6.3 Reception of the Last Seven Blocks

Once the Nest Protect receives the last block of data, it can finish decoding the WiFi password. It will put together the partitions of the encrypted password, decrypt it with it's AES key, check that the device id corresponds to its own, and go to the next step in the process.

5.7 Attempt to Connect to the Network

Now that the Nest Protect has the WiFi credentials, it can attempt to connect to the network in the same manner that it would under the previous scheme. The router then sends a response: if the connection is a success, the onboarding is done and the device can be used. Otherwise, the device should sound an error message and ask the user to attempt the process once more.

6 Comparison to SoftAP

6.1 Security

When compared to SoftAP, in terms of security, DOLO has multiple advantages.

Firstly, DOLO has a greatly reduced attack surface when compared to SoftAP. It doesn't require the IoT device to open a WiFi hotspot, which involves a lot of steps and a lot of potential vulnerabilities. For instance, the user having to connect to an unsecured networks poses the risk of an Evil Twin attack, where the attacker would impersonate as the SoftAP network the user wants to connect to [**?**]. This situation puts the attacker in a man-in-the-middle position, which can pose all sorts of issues.

On the other hand, DOLO only adds the possibility to communicate with the device in a very contained manner. The only interactions allowed through this mean are the sending of WiFi credentials and the information about whether or not the network is hidden. This means the user's phone isn't put at risk in the same way, as it doesn't need to connect to an unsecured network. The only security risk for the phone is the QR code, but tempering with the manufacturer's would require the physical presence of the attacker before the user installs their device to the roof.

The added encryption in DOLO makes it so that an eavesdropper looking to steal the WiFi password would need to access both the QR code in the back of the Nest and see the transfer taking place with high-end camera equipment. But, had they had this equipment in the first place, they could simply read the password while the user types it on their phone. As for SoftAP, a false network would be able to trick the user into sending their credentials their way without much trouble.

Another type of attack that is made more complex is that of an attacker who tries to change which network the IoT device connects to. This would then require the attacker to have physical access to the device and remove it from the ceiling to access its QR code and change said network afterwards. Other complementary schemes for permission control could be used to remediate this kind of issue.

6.2 Usability

While SoftAP is an ingenious system, it sometimes causes usability issues.

6.2.1 Automatic Disconnection from Hotspot

Some phones will tend to disconnect from the temporary WiFi hotspot created during SoftAP. This happens because that hotspot does not grant access to the internet, which modern phones need for many of their functions, so the phone's automatic switching feature will switch to another WiFi network or the cellular network. The disconnection would cause issues for the onboarding process and can be frustrating for users, especially those who are less tech literate [2].

The opposite problem can also occur, where the phone doesn't disconnect from the hotspot when done sharing the credentials. The phone would then be stuck without a connection, while not showing any helpful error message. This is exacerbated by the fact that since the phone can't connect to the Internet, users cannot search for fixes through it.

When using DOLO, there is no such conflict. It doesn't monopolize the phone's WiFi or any other essential features. The flash isn't usually controlled by background processes, making it a good candidate for the transfer. The user won't need to deactivate any feature on their phone or reconnect to their network.

6.2.2 iOS's Lack of App Control Over Network Connexion

In the case of iOS devices, using SoftAP for enboarding has an additional layer of complexity for the user. Applications running in that system do not have the rights to make the phone connect to a different network. That means that the app may have to prompt the user to manually connect to the IoT device's WiFi hotspot, and then come back to the app [2]. This can create a lot of confusion for end-users.

As for DOLO, this is a non-issue, as both Android and iOS have a way to control cameras and camera flashes, as can be seen from the multitude of camera and flashlight applications available on their respective app stores.

6.2.3 Short-Lived Hotspots

Since the temporary network is not secure, IoT devices will close it after a certain period of time. Sometimes, this occurs while the user is still typing their WiFi credentials in the application, which may result in errors. These errors may not be recognized as timeouts by the app but as wrong credentials, leading to confusion for users [2].

This is less of an issue for DOLO, as the communication only begins after the credentials have been put in the app. There is still the issue of differentiating wrong credentials and errors in the data transfer, but that can be mitigated with more parity and integrity checking.

7 Other Use Cases

DOLO can be used in many kinds of IoT devices. It simply requires the presence of a light sensor and space for a QR code. This is quite frequent, as IoT devices have been getting

more and more sensors, and ambient light sensors are quite commonly added [**?**]. The device would also need the capacity to hash with SHA-256 and decrypt with an AES key. For error handling, it needs an output of some sort , sucha as an LED or a speaker. This is already present in most IoT devices; very few are eliminated because of this condition.

8 Conclusion

In this paper, we have shown that DOLO is a realistic protocol for sending network credentials to smart smoke detectors. We looked at the Google Nest Protect and its characteristics, seing that it is a good candidate for applying DOLO. We showed that DOLO could be reasonably used by an end-user with little technical expertise, and showed that it is able to achieve reasonable speeds, even under conservative assumptions about sensor and emitter top speeds. Through our analysis, we also saw that DOLO has many advantages over the old scheme because of the added simplicity of the protocol, as well as the fact it doesn't require as many assumptions about the way the device works as the previous scheme.

References

[1] Phaneendra, Lakshmana, et al. "Efficient Smart Emergency Response System for Fire Hazards Using IOT." International Journal of Advanced Computer Science and Applications, vol. 9, no. 1, 2018, https://doi.org/10.14569/ijacsa.2018.090143.

[2] Nelson, Barbara. "The Challenges of Soft AP: What Goes Wrong and Why." Wayback Machine, 6 June 2017, https://web.archive.org/web/20220817020030/https://blog.cirrent.com/challenges-soft-ap.

[3] "Set up Nest Protect with the App." Google Nest Help, Google, https://support.google.com/googlenest/answer/9247146.

[4] "Softap." Wikipedia, Wikimedia Foundation, 7 Aug. 2022, https://en.wikipedia.org/wiki/SoftAP.

[5] Fact Sheet - Nest Protect: Smoke + Carbon Monoxide. 4 Sept. 2014, https://storage.googleapis.com/nest-public-downloads/press/documents/nest-protect-fact-sheet-2.pdf.

[6] "Learn More about 1st Gen Nest Protect Sensors." Google Nest Help, Google, 2023, https://support.google.com/googlenest/answer/9251134.

[7] "Nest Protect Technical Specifications." Google Nest Help, Google, 2023, https://support.google.com/googlenest/answer/9229922.

[8] Vliet, Kari Van. "Feasibility of a Smart-Phone Ambient Light Sensor as a Tachometer." McMaster Journal of Engineering Physics, vol. 2, no. 1, 16 Jan. 2018.

[9] "Troubleshoot a P002, P008, P009, P013, or P018 Nest Protect Message." Google Nest Help, Google, 2023, https://support.google.com/googlenest/answer/9935037?hl=en.

[10] Poole, Nick. "Nest Protect Teardown." Nest Protect Teardown - SparkFun Learn, https://learn.sparkfun.com/tutorials/nest-protect-teardown/all.

[11] "K60 Family Product Brief." Freescale Semiconductor, Freescale Semiconductor, May 2011, https://cdn.sparkfun.com/assets/c/c/8/e/d/52a8c005757b7f4b738b456c.pdf.

[12] Tiwari, Sumit. "An Introduction to QR Code Technology." 2016 International Conference on Information Technology (ICIT), 2016, https://doi.org/10.1109/icit.2016.021.

[13] "Information Capacity and Versions of the QR Code." Information Capacity and Versions of QR Code — QRcode.com — DENSO WAVE, DENSO WAVE INCORPORATED, https://www.qrcode.com/en/about/version.html.

[14] "Error Correction Feature." Error Correction Feature — QRcode.com — DENSO WAVE, DENSO WAVE INCORPORATED, https://www.qrcode.com/en/about/error_correction.html.

[15] Fitzpatrick, Jason. "How Long Can You Make Your Wi-Fi Password?" How, How-To Geek, 20 July 2022, https://www.howtogeek.com/817996/how-long-can-you-make-your-wi-fi-password/.

[16] Fitzpatrick, Jason. "How Long Can You Make Your Wi-Fi Network Name?" How, How-To Geek, 18 July 2022, https://www.howtogeek.com/817563/how-long-can-you-make-your-wi-fi-network-name/.

[17] "Find Google Nest or Home Speaker or Display's Serial Number." Google Nest Help, Google, 2023, https://support.google.com/googlenest/answer/9169258.

[18] Sakimura, Nat, et al. "OpenID Connect Core 1.0 Incorporating Errata Set 1." Final: OpenID Connect Core 1.0 Incorporating Errata Set 1, https://openid.net/specs/openid-connect-core-1_0.html.