Introduction to IBM's ClusterDuck Security Protocal:

This document is a collection of information preparing someone to understand and eventually contribute to the security protocol implemented for the Cluster Duck Protocol. (This document will eventually be formatted into a README doc that can be put on the CDP github). It begins with an explanation on how the duck hardware works, and then goes into a brief crash course on network security and zero trust policy. Finally, the actual encryption used in the CDP is explained, and the current progress is shown. The following is a table of contents of all the topics to be covered:

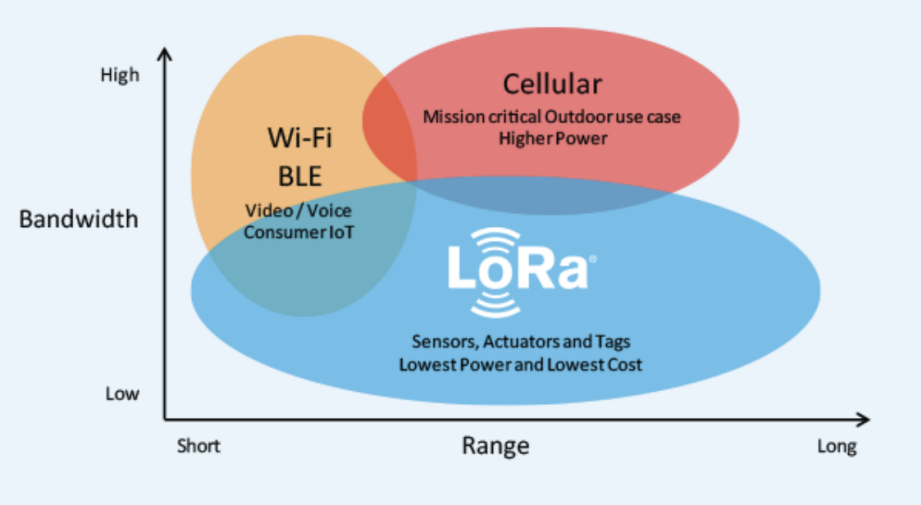
Table of Contents:

What the ClusterDuck hardware relies on to work:

The ClusterDuck hardware is defined as a firmware for Internet-of-Things electronic devices that utilize LoRa (Long Range radio), Bluetooth, Wifi, and a variety of sensor integrations. The custimaatioon allows users to set up ad-hoc sensor and communication networks in areas where connectivity may be degraded, hard to work with, or nonexistent. While it remains a great help in the event of a natural disaster, the protocol has been vulnerable to outside attacks. Since the ducks pick all messages transmitted through a specific frequency over a specific range, anyone with the proper frequency could transmit malicious code. An example of such an attack could be akin to a DDOS, where the nearby ducks are simply overwhelmed with a vast number of random transmissions prohibiting them from taking in transmissions that matter. Other instances would involve someone listening in on the duck transmissions and potentially intercepting any resources/help that is communicated back. Another, arguably worse, case would involve sending a calculated transmission; with a compressed code sequence that once read by the duck, could change its internal implementation. For such reasons, it is important to make sure not only are the messages duck's send to one another are encrypted, but that the network on which they are sent is also safe.

- (source) https://github.com/Call-for-Code/ClusterDuck-Protocol/wiki

How is LoRa, Bluetooth, WiFi, etc. applied to the ClusterDuck Protocal (CDP)?
Most of the ducks are made to use LoRa, as its transmission range is greater than that of standard WiFi and Bluetooth. To understand LoRa, we must first understand its differences to WiFi and calculator networks. In the world of IoT, WiFi and Bluetooth (which have a comparatively short range but high bandwidth capacity) is typically used for video/voice connections, or consumer IoT. Cellular networks are similar in that they have high bandwidth capacities, however they additionally support long range connections (meaning they use high power). LoRa provides a good middle ground as it can be applied to both short and long range connections while still maintaining a reasonable bandwidth capacity. The below image provides a representation of what was just described.



Source: https://www.semtech.com/lora/why-lora

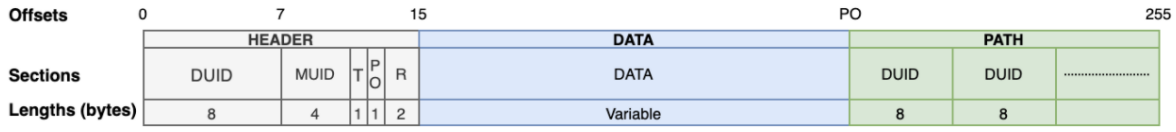
Since the purpose of the CDP is to provide remote locations/locations ravaged by natural disasters with a communication signal, having a network that can work in a wide variety of ranges while requiring minimal power capacity to support the bandwidth is the optimal solution. For this reason, LoRa is used in most of the physical duck hardware (sensors, data transmitters, etc). The CDP has expanded since its founding to include apps and more localized locations that would use the other types of network connections, however, for the sake of conciseness (and also since my focus is the hardware itself), the other networks will not be addressed in great detail. The below links can be used to further explore the types of network connections. Further exploration can also be done into how radio waves transmit data/how RSSI and SNR play into this transmission. As this focuses less on how the CDP works their data and more on radio waves in general, it will not be covered in this document. The bottom three links of this section cover that information.

- (source) https://www.semtech.com/lora/why-lora
- (source) https://cybersecurity.att.com/blogs/security-essentials/demystifying-network-isolation-and-micro-segmentation
- (source) https://www.metageek.com/training/resources/understanding-rssi.html

- (source) https://helpcenter.engeniustech.com/hc/en-us/articles/234761008-What-is-RSSI-and-its-acceptable-signal-strength-#:~:text=RSSI%20stands%20for%20Received%20Signal,a%20lower%20overall%20data%20throughput.
- (source) https://en.wikipedia.org/wiki/Signal-to-noise_ratio

How does the CDP compress and transmit their data through ducks?
The current LoRa technology allows for messages to be 256 characters long, meaning the CDP needed to create a concise manner of relying large amounts of information in a short message. The original message can primarily be shortened (and later expanded) using binary bit methods and shifting methods. This allows larger messages asking what type of resource (water, transport, food, inspection, first-aid, etc) is needed, how many people need it, relevant details about the people who need it (age), if there was been an issue (natural disaster), as well as whether the resources are urgent to fit into a 95 character message.

With the current concise message handling system, users are able to encode up to 256 different topics. The message is divided into the sections labeled below. The MUID does not always need to be filled in as it will automatically detail which duck it came from, which is usually enough to specify the type of message it would be.



Header

DUID	**64 bits** hexadecimal value uniquely identifying the device in the mesh network. Compatible with LoRa DevEUI format.
MUID	**32 bits** value uniquely identifying the message.
T	**8 bits** value representing the message topic.
PO	**8 bits** value representing offset to the PATH (LSB) sections.
R	**16 bits** value reserved for future use.

Variable Sections

PATH	A variable length series of 64 bits Device UID representing the devices in the mesh that have seen the message during its transmission.
DATA	A variable length data payload carried in the transmitted packet.

Source: https://github.com/Call-for-Code/ClusterDuck-Protocol/wiki/MessageHandling

- (source) https://github.com/Call-for-Code/ClusterDuck-Protocol/wiki/MessageHandling
- (reference - binary bit methods) https://www.hackerearth.com/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

Network Security Overview:
The commonly accepted threat model recognised within Network security revolves around the following points: 1) the adversary can intercept/model network traffic 2) the adversary can send packets 3) the

adversary has full control of their own machines 4) the adversary can participate in protocols. Commonly, we see adversaries eavesdropping on packets (which leads to the senders data being unknowingly spread/seen) in addition to sending/spoofing packets. The latter is particularly dangerous as the packet can be disguised to come from any source address and be sent to any IP address. It is important to consider the common network vulnerabilities of the past in order to understand how various aspects have been changed as well as what potential parts adversaries could attack in the future.

How do adversaries acquire the information they need to perform malicious acts?
a) How do they know the software/protocol being run?

Much of the initial information comes general probing on the system or protocol. This includes monitoring a well-known port and seeing if the target system is listening to it. Additionally, this would include acquiring the initial banner message that protocols and systems often send out. If probing leads to no outcome, then the DNS (Domain Name System) can be used to look up the hostname for an IP address. If the target device is specifically an OS device, then nmap can be used to guess the OS by measuring various impl-specific details (The most prominent is using TCP/IP stack fingerprinting). Finally, if all else fails, adversaries will return to the most basic method: guessing. By assuming that the system is vulnerable, they will attempt to find a bug that can be exploited to give them some form of access.

- (source) https://nmap.org/book/man-os-detection.html
- (source) https://en.wikipedia.org/wiki/Transmission_Control_Protocol

b) How do they know the IP address of the system to attack?

While it sounds like a difficult thing to do, obtaining the IP address is far less complicated than one might think. There's even open source that confidently allows large-scale studies of the hosts and services that compose the public internet. In other words, the IP address can simply be found by scanning the internet (which would be about 2^32 addresses). To get a scale of how long this would take, consider the following. There is a 1Gbps (100 MB/s) network link and 64 byte minimum of packets. There are roughly 1.5M packets being sent every second. With 2^32 addresses, you can have 4B packets in ~2500 seconds (or 45 minutes). For a less general search, one could follow the traceroute approach and find the routers along the way. This is commonly seen in GBP attacks (a form of application-layer DDoC attack).

- (reference) https://zmap.io/ -- open source tool mapping hosts and services composing public internet

With all these potential loose ends, it becomes more important to ensure the original specification (and any additional edits/extensions) are able to close up vulnerabilities. Some of the original vulnerabilities, particularly with TCP include:

- Spoof connections to applications that rely on IP addresses
- DOS attacks triggering connection reset
    - E.g. TCP connections between BGP routers
- Hijacking existing connections

This however, is not all. Many issues arise from the fact that the very specification of security protocol is flawed, but due to the vast number of machines/systems using those protocols, it is incredibly hard to convince people to convert on mass. Some other issues include:

1. Routing protocols being overly trusting of participants
2. DHCP within a single Ethernet network
3. GBP (internet wide)
4. ICMP messages (e.g. redirect)
5. Information exposure
6. identd ("Authentication Service")
7. Email (concerning authentication vs authorization and spam problems)
8. Passwords in protocols
9. FTP data transfer protocol

How were these problems mitigated?

1. SYN flooding defense → SYN cookies
2. DOS attack vector → bandwidth amplification
3. TCP fix → congestion control

How can we better improve security?

1. Protocols (compatible fixes to TCP implementation)
2. Firewalls
3. Security on top of TCP/IP:SSL/TLS,Kerberos,SSH, etc
4. Cryptography
5. Replacement protocols
6. Employ harder top securities

The above 6 options are potential improvements to the detailed security problems. However, it is important to consider that even these solutions contain their own vulnerabilities and might not always lead to a solution.

Firewalls, for example, can only provide a partial fix despite being widely used. To start, the adversary may already be inside the firewalled network, making the entire design redundant and providing the user with a false sense of safety. Additionally, it is difficult for firewalls to determine if the incoming packets are malicious or not (which is why some systems are attempting to focus more on what data is being let out as opposed to what data is coming in). Lastly, of the main firewall issues, is authentication, even for present fields like src/dst. Another issue comes in the form of compatibility. TCP/IP's designs are not good matches for the firewall filtering techniques and will often lead to unexpected behavior.

Cryptography, as another example, is easier said than done. To apply it, entire protocols must be designed, key distribution must be implemented, a trust system needs to be worked out (are any users "trusted?? Is there a level of trust? Is there zero trust?), etc.

The same can be said about the last solution: employing harder top securities. This would include things like DoS-resistance and routing, however, like the cryptography solution, this too requires a lot of work and compatibility with those in the network.

- (source video) https://www.youtube.com/watch?v=BZTWXl9QNK8&list=PLUl4u3cNGP62K2DjQLRxDNRi0z2IRWnNh&index=12
- (source notes)
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-858-computer-systems-security-fall-2014/lecture-notes/MIT6_858F14_lec12.pdf
- (source video) https://www.youtube.com/watch?v=GqmQg-cszw4&list=PLUl4u3cNGP62K2DjQLRxDNRi0z2IRWnNh&index=1
- (source notes) https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-858-computer-systems-security-fall-2014/lecture-notes/MIT6_858F14_lec1.pdf

Mobile/Cell Phone Security (Android):
This becomes relevant as the CDP expands to include apps that can be downloaded by anyone willing to participate/contribute to the project. So we start with the basics; an apps threat model. The biggest security issue when it comes to apps is that anyone is able to write an app and anyone is able to install it. These apps could contain malicious code that takes in information its users are unknowingly giving. Additionally, if the app one makes does not have the proper security measurements installed, depending on how it operated, users could gain access to levels of the app they should not, and malicious users could wreak havoc. Since the CDP app is designed by project members, this document will assume that no malicious effects have been added. However, in order to get a full outlook of mobile security, both threats will be considered.

The first thing one needs to understand are the components of an Android (and loosely an Apple) application. There are four such components:

1. Activity: UI component
2. Service: background processing
3. Content provider: a SQL database that can be accessed by other components
4. Broadcast receiver: gets broadcast announcements from other components

Another important concept to understand is “intents,” or what the app/user wants to do when they interact with either other apps or the components of the current app. Many security issues arise when looking closer at how each of the above components handles both app and user intent. The goal with Android’s approach is to make the security/intent aspect transparent to the user. If they are aware what permissions the app is requesting, what functions are being run in the background, functionality of the UI, etc. the user knows what the app has and doesn’t have access to and can prevent any unwanted data to be shared. Explicitly stating these permissions also makes it harder for the reverse to happen: since the apps keep better track of what they can and can’t do, it makes it much harder for a malicious user to come in and perform unspecified commands.

So how do Android applications actually work? First, their model for handling app interaction is largely based on intents. Apps require multiple intents in order to perform various operations. While the course of action is up to the framework, the intent usually gets redirected into a request for access from the user.

Another important aspect is app isolation. A common framework for this is giving each app its own UID, as it ensures no app can manipulate another app's processes files (i.e. the app private directory accessed in /data/data/appname where preferences, sqlite DBs for contend providers, caches files, and more is stored).

As we've seen, intents are pretty crucial to the workings of each app, so who is given the permissions to receive them? Usually apps are permitted to specify these with their own arbitrary parameters. This could work on a lower level, but what if you need to broadcast intents? How would you specify the recipients of an action that normally goes to all available options? This is achieved by adding additional labels to the broadcast intents so that they will only go to recipients who have that label.

Now that we've determined who can receive intents, we must tackle how to authenticate them. This is likely the largest concern for the CDP as the whole app relies on sending various data to one another. Traditionally, the way to authenticate is by reading the permissions label on the content received. However, apps often forget to put permission restrictions on broadcast receivers (which is what the CSP relies on, hence a problem). Authentication too, is problematic. Oftentimes, when relying on names to route intents, the choosing/authentication system is simply a "first-comes-first-serve" method, meaning a malicious app would just have to send information before the actual sender in order for their version to be accepted. The malicious app could go one step further and even use the same name as the normal app, which would practically remove any suspicions of someone watching the network. The solution to this, like many of the other solutions for network security, is to make a closed system with specified and well defined permissions to ensure that only who you want is able to enter the network.

Applications could also rely on signatures for verification, however since there's no real need for a CA, there are ways malicious users could get around it. The three general checks for this method are as follows:

1. Did the new version of the app come from the same developer as the old version sent?
2. Did the two apps come from the same developer?
3. Did the app come from the same developer as the one defined in a permission?

If these three criteria are satisfied, then the sender can be trusted and brought into the network.

Temporary permissions are also a possibility, which will become necessary for the CDP as regular people begin downloading the app and making non-code contributions. There are two parts to doing this: 1) URI delegation and 2) Pending intents. Taking the URI delegation part is effective so long as the permissions are removed when the other app has completed its purpose/task. The URI would first grant read/write access and keep track of any changes being made by the other app. The access would be revoked when the app is finished and a log of everyone granted access will be kept in the original app. Pending intents revolve around callbacks into the application (e.g. alarms/time services) where the pending intents are stored in memory and accepted at a designated time. This provides a large level of control for what is given permission into the network and can prevent unintended users from gaining access.

With these principles in mind we can now understand the optimal security model for mobile phones; whether that be Android based apps or IOS/iPhone apps.

1. All apps run two possible UIDs

The first UID would be for the Apple specific apps, and the other UID would handle the rest. This way, when switching between apps, the entire UID model does not need to change. With this model, apps are isolated by the Apple "sandbox" and additionally from each other.

2. Prompt for permissions at time of use

By requiring a prompt whenever a particular permission is needed, there is considerable transparency between user and app or app and app, making it much harder for data to be siphoned and used unknowingly. This option allows for the continued use of an app, even if the permission was not granted (of course this depends on the apps functionality, e.g. an app meant to tell you your location will not still work if you do not grant it access to your location).

3. Trust the app approval that Apple and Android do in the app stores

This last portion of the security model is the most unreliable aspect. For starters, the security standards of Apple and Android already differ. In addition, it is incredibly hard to go through every single app in the app store and determine whether it is safe for users. On the bright side, the CDP is less worried about this as the security issues are from app users→ app instead of the usual app→ app users concern.

- (source - video) https://www.youtube.com/watch?v=uT7BXusDgDM&list=PLUl4u3cNGP62K2DjQLRxDNRi0z2IRWnNh&index=19
- (source - notes) https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-858-computer-systems-security-fall-2014/lecture-notes/MIT6_858F14_lec20.pdf
- (source) https://developer.android.com/training/articles/security-tips
- (source) https://www.microsoft.com/en-us/research/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F149596%2Fappfence.pdf
- (source) http://css.csail.mit.edu/6.858/2012/readings/ios-security-may12.pdf
- (source) https://reverse.put.as/wp-content/uploads/2011/09/Apple-Sandbox-Guide-v1.0.pdf

Network Security: Zero Trust Policy:

In recent years, network security has had to make a considerable shift on how networks interact with one another. The old policy had been "don't trust until verified" and is now becoming a simple "don't trust." The highest concern is no longer protecting a specific admin level (though that is still very much a priority) but protecting the data itself as it moves around; in other words, data has become the new "border."

While network security won't be applied in the grandest sense to how the CDP operates, elements of it are present, and the mentalities of zero trust can similarly be applied. When changing a network to behave with zero trust, there are 7 primary characteristics that must be considered:

1. The network is always assumed hostile
2. Assume the hostile is already inside your network
3. Network locality (segmentation) is not sufficient for deciding trust in a network
4. Every device, user, and network flow is authenticated and authorized
5. Policies must be dynamic and calculated from as many sources of data as possible

6. The device is no longer the border. A user's identify/data pair is the new boards
7. Containers, serverless, and cloud computing are the new disruptors of traditional security architectures
8. Mobile users, mobile apps, mobile storage

If applied to the duck network, many aspects can remain the same. ClusterDucks can follow the stric rule of securing their data regardless of their level and encrypting said data as it travels up the chain-of-command. The sensor ducks (lowest level) are all programmed to keep track of specific environmental conditions, and would therefore be impervious to remote attacks (could only tamper with data collection if you get a hold of the actual duck) so they only need to worry about encrypting the data sent out.
MamaDucks have the primary task of recieving transmissions sent from all the lower level ducks. Therefore, someone could impersonate a duck and send faulty/malicious data to the Mamaduck. Adopting a zero trust policy where all the existing duck identities are verified would prevent such an attack from happening. Additionally, even once verified, there should be a log of all the network traffic experienced (incorporate Single Packet Authorization, or SPA) between ducks. This way, the distinction between trusted-inside-perimeter and untrusted activity that crosses the perimeter is eliminated.

Outside of the duck network, the CDP has recently also expanded to an app which only increases the necessity for a stronger security protocol. Such security concerns were outlined in the Mobile/Cell Phone Security Section.

- (source) https://www.youtube.com/watch?v=E03gh1huvW4
- (source) https://www.youtube.com/watch?v=EF_0dr8WkX8&t=786s
- (my notes) https://docs.google.com/document/d/1o-YsSzchtq4Z2ismJ2aaf1G6SiWBKngyVB-dZ2qMDS4/edit?usp=sharing

How can that be taken advantage of?
Clients can be granted access to certain aspects of a page/network and if that access is never taken away, it becomes far easier to reach other access levels. In the context of the CDP, if someone is able to get access to any of the first level ducks (sensor, scanners, etc.) they would be able to follow the chain-of-command up to the Mama and Papa ducks where alterations to the entire network of ducks could be done. Even if the advisory cannot access the upper levels, the fact that they made their way into the network and can now alter any sensor/data collecting ducks invalidates the entire security model and no longer allows users of the network to assume the data remains uncorrupted.

With ClusterDuck expanding into the app world, even more security breaches are possible depending on what the user grants the app access to. These are addressed in the Mobile/Cell Phone Security Section.

Current CDP Encryption:
Ducks currently support AES-256 CTR, also known as Advanced Encryption Standard Counter Mode, or Rijndael. AES is a block cypher that has the ability to operate under 5 modes. The modes are as follows:

- ECB (Electronic Code Book)

- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback)
- OFB (Output Feedback)
- CFT (Counter)

Since CluckerDuck uses the CRT mode, that is the only mode that will be explained in detail. AES-CRT requires an encryptor to generate a unique per-packet value and communicate that value to the decryptor. AES is one of the most well-known/used ciphers of today and it has a relatively simple encryption process. The AES block cipher encrypts and decrypts data by XORing it with a key produced by AES encrypting sequential counter block values. By this process, not only is the data encrypted, but each block of data is encrypted by a different key, all stemming from the original one used on the first block. AES is particularly convenient since it can be pipelined and parallelized. Since the data is split into individual blocks that are all XORed by a key stream (unaffected by the data), the entire key stream can be calculated before the actual encryption process. Then when the data is present, each individual block can be efficiently encrypted at the same time. Not only does this allow for smooth assigning of the nonce (an arbitrary number that is used just once) and IV (initialization vector; an input to a cryptographic primitive being used to provide the initial state), but it also allows for a reduction in the packet latency.

- (source) https://tools.ietf.org/html/rfc3686
- (source) block ciphers video: https://www.youtube.com/watch?v=oVCCXZfpu-w

What is AES/how does it work:
As briefly explained above, AES is the most commonly used encryption cipher when it comes to securing data. It utilizes a block cipher chain as a means of quickly encrypting large amounts of data, but how does the block cipher work?
The process is performed by first separating a plaintext, call it PT, into 128-bit blocks (The finally block can be 128 bits or less, so long as the rest are 128 bits). The representation would look like the following:

```
PT = PT[1] PT[2] ... PT[n]
```

Once the plaintext has been split, and all the related keys have been calculated, the plaintext can then be encrypted. First, every PT block is XORed with a block of the previously calculated key in order to create the ciphertext, CT. Of the resulting 128 bits of key steam, the 96 most significant bits are set to the noce value (32 bits) and then the per-packet IV value (64 bits). The least significant 32 bits are then incremented by one to generate the following counter blocks (128 bits). This is also known as the next key stream. The process, when using pseudocode looks like the following:

```
CTRBLK := NONCE || IV || ONE
FOR i := 1 to n-1 DO
  CT[i] := PT[i] XOR AES(CTRBLK)
  CTRBLK := CTRBLK + 1
END
CT[n] := PT[n] XOR TRUNC(AES(CTRBLK))
```

Where the AES() function performs each encryption and the TRUNC() function truncates the output of each AES operation to be the same length as the final plaintest block. It allows for the most significant bits to be returned.

- (source) https://tools.ietf.org/html/rfc3686

Potential leaks/cipher weak points:
1. While this first example is not an attack, it is a significant aspect to consider with the cipher. AES-CTR relies heavily on the fact that counter block values are never used for more than one packet with the same key. If something like this occurs, the entire confidentiality guarantees are voided, as there are now multiple encryption samples that respond differently to the same key and can be used to get back both the key used and original text.

- Usually this is avoided by the use of the Internet Key Exchange (IKE) protocol which creates new, unique keys

2. In order to guarantee that the AES-CTR is working properly, it must adhere to the Encapsulating Security Payload (ESP), which gives protection to upper layer protocols (IP data payload, not the IP header). The ESP is defined as "a transport layer security protocol designed to function with the IPv4 and IPv6 protocols. It takes the form of a header inserted after the internet protocol header, before an upper layer protocol like TCP, UDP, or ICMP, and before any other IPSec headers that have been put in place." () The ESP protects a data packet that has been signed for integrity in an already encrypted area that indicates the protected information. If the AES-CTR does not follow these instructions to the "t", information could leak into an unprotected layer, thereby defeating the entire purpose of the encryption cipher.

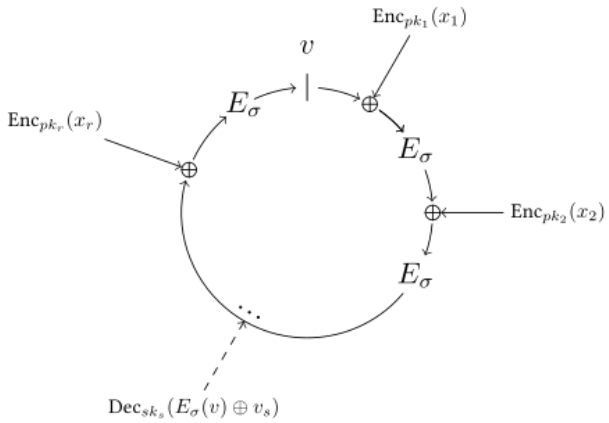
- (source) https://blog.finjan.com/encapsulating-security-protocol/

3. The first of the potential leaks involves an man-in-the-middle attack during the key creation portion of execution and is mostly a vulnerability for 128-bit AES keys (as opposed to the 192-bit and 256-bit keys). Keys of such a length are limited to a theoretical strength of 2^(n/2) bits, where n is the number of kits in the key. With such a key length, and since AES does not pad its key or plaintext/ciphertext, an attacker (given they have the proper memory and processor resources available) could have the means to make precomputation attacks where they are able to recreate sensitive data by searching huge tables of ciphertext associated with the known plaintext and keys.

- (source) https://tools.ietf.org/html/rfc3686 (section 7)

4. Cipher Malleability
CRT mode used in conjunction with AES is malleable, which means that it is possible to change the ciphertext into another ciphertext which then decrypts to a related plaintext. As you can probably guess, this would lead to problems should an attacker find a way to manipulate the original ciphertext. Worse yet, the attacker does not need to know the contents of the message in order to manipulate it.

Since AES-CRT is malleable by design, there is a different approach to concerns with morphing data. Cryptosystems that are meant to be malleable can follow/be applied to specific schemes, known as the homomorphic encryption schemes, which look at the malleable aspect as a positive feature. In this case, an encryption of m could be turned into a valid encryption of f(m) for restricted classes of functions f without needing m. This specific case is usually used for privacy-preserving outsourced storage and computation.



https://en.wikipedia.org/wiki/Homomorphic_encryption

While concerns are there, if the cipher follows its specifications when being computed (preferably the CDP could use a public programming of the cipher that's trusted and also time-tested), it can be semantically secure meaning only trivial information could be extracted from the ciphertext (specifically when run against any probabilistic, polynomial-time algorithm).

- (source) https://crypto.stackexchange.com/questions/33846/is-regular-ctr-mode-vulnerable-to-any-attacks
- (source) https://en.wikipedia.org/wiki/Malleability_(cryptography)
- (source) https://en.wikipedia.org/wiki/Homomorphic_encryption
- (source) https://en.wikipedia.org/wiki/Semantic_security

What do the field tests tell us:
It is important to remember that the Cluster Duck Protocol just began its expansion into adding security features, and thus, does not have much in place. As of now, the AES-CTR encryption has been successfully implemented and stood against any potential decryptions/data leaks. As part of the things that went well: the encryption works as intended and has the benefit of being easy to implement. The 0x50 sync work worked as intended (performed by the Red Team) and the Blue Team (0x25) did not receive any data from the Red Team. In other words, data that was not meant to be received by CDP users was successfully not received. An additional bonus from the field test was that the ducks were able to send and receive data securely while being far away (in broadcast range).

On the flip side, the field tests also revealed some failures. The most significant error would be that the ducks are dropping MANY packets. While it's good that the sent data can't be mutated/additional malicious data cant be added, dropping data is just as bad and defeats the purpose of an encryption if attackers are going to receive that data anyway. Additionally, there was a test where the control PapaDuck (0x12) received data from the Blue Team even though the sync word was explicitly set (e.g. data was receded when it shouldn't have been). Lastly, the detector duck (duck designated to all things security) was noted to not be easy to change, which could become a considerable issue -- especially if a vulnerability is found and needs to be accounted for quickly.

Overall, the field tests are very clear in showing the direction the CDP will be going with their testing/security implementation.

- (source) https://docs.google.com/document/d/1TEP2Coty3UJfZKZ8YtEnmwEcIZ68qTJAino9DNOHckQ/edit