TDDE21 Advanced Project: Secure Distributed and Embedded Systems

DRIP Project Report

Thea Antonson
Gustav Carlsson
Olle Mineur
Albin Thulin

{thean981, gusca083, ollmi247, albth693}@student.liu.se

Linköping University
December 12, 2025

Contents

1 Introduction

The report is part of a project in the course **TDDE21 Advanced Project: Secure Distributed and Embedded Systems**. The project is carried out by a group of four students who are continuing the work initiated in previous years of the course.

1.1 Motivation

The number of drones worldwide is steadily increasing, and with this growth the presence of harmful or unauthorized drones is also rising. Such drones frequently appear in news reports due to their potential misuse for activities such as spying on military operations, critical infrastructure, or private individuals. To reduce the impact of harmful drones and make it more difficult to operate them anonymously, there is a need for mechanisms that allow authorities to identify drone ownership. This project addresses that need through the **Drone Remote Identification Protocol (DRIP)**, which provides a method for verifying drones and broadcasting an identification signal within the nearby area. This enables observers to scan and determine the ownership of a drone in real time.

1.2 Project goals

The selected project goals for this year were:

- Setup public DNS registry
- DRIP support for Crazyflie
- Update to registries-33

1.3 Definitions

Below is a list of definitions:

- **BLE** - Bluetooth Low Energy
- **DET** - DRIP Identity Tag
- **DRIP** - Drone Remote Identification Protocol
- **DNS** - Domain Name System
- **FEC** - Forward Error Correction
- **HDA** - HHIT Domain Authority
- **HHIT** - Hierarchical Host Identity Tag
- **IETF** - Internet Engineering Task Force
- **P2P** - Peer-to-Peer
- **RAA** - Registered Assigning Authority
- **RID** - Remote Identification
- **UAS** - Unmanned Aircraft System
- **VNA** - Valid Not After
- **VNB** - Valid Not Before

2 Background

2.1 DRIP

DRIP is a protocol that aims to add a standard way of authentication. DRIP is an expansion on ASTM RID by defining means of authentication [7]. There are multiple types of DRIP messages that are defined in RFC9575 [10]. DRIP's core objective is to ensure UAS Remote Identification (RID) data is immediately actionable. That means sufficient information is available, especially during emergencies or network constrained environments [4].

2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was first introduced in 2010 with the release of Bluetooth 4. The idea of BLE compared to Bluetooth is that it has similar range but lower energy consumption. This is mainly achieved by using short intermittent transmission bursts instead of continuous streaming. Another difference is that BLE uses a simplified protocol stack, which both conserves energy and reduces complexity compared to traditional Bluetooth [6].

Due to the characteristics of BLE compared to Bluetooth it is often a good choice for devices that need low energy consumption and that need small amounts of data infrequently [6].

2.3 Crazyflie

Crazyflie is a open source flying drone. One defining characteristic of the Crazyflie is that it is small and is light weight. It has two types of communication, radio and Bluetooth. One sub-type of radio communication is Peer-to-Peer (P2P) which can be used to send messages between Crazyflies. One perk of using P2P to send messages between Crazyflies is that it is defined in the so called app-layer, which makes it easy to define and send own messages.

2.4 DNS

As a part of the DRIP verification chain, the certificates for any endorsement can be queried from a DNS server. This way, the verifier can walk through the list of broadcast endorsements one by one, query the certificate for that endorsement, and verify its validity. This is done for the HDA, RAA and finally the Apex allowing verification to the root of trust.

2.4.1 Bind9

BIND 9 (Berkeley Internet Name Domain) is the most used DNS software. The software uses text files to store DNS records which is loaded into memory at startup. BIND 9 do not have support for HHIT and BRID, which are needed for DRIP, at the time of writing this report. Instead generic record types have been used as placeholders (TYPE65280). Read more in the BIND 9 Administrator Reference Manual [1].

2.4.2 PowerDNS

PowerDNS is a popular open source DNS, and provides an authoritative server, recursor and a load balancer. It allows for a back-end agnostic approach, where the user can choose between over ten different ones [1]. PowerDNS has recently added support for HHIT (RRType67) and BRID (RRType67).

2.4.3 DNS Resource Records (RR) for DRIP

The `drip-registries-33` draft defines two specific Resource Record (RR) types for the storage and retrieval of DRIP public information:

- **HHIT (RRType 67):** The *Hierarchical Host Identity Tag* record is the cornerstone of the DRIP verification architecture. It binds a drone's DET (its ID) to its **Host Identity (HI)**, which effectively serves as the drone's public key.

[1] https://doc.powerdns.com/authoritative/backends/index.html

When an observer (such as the Android app) queries this record, it retrieves the "Canonical Registration Certificate". This certificate contains the public key necessary to verify the cryptographic signatures on the messages by the drone. Without this record, an observer cannot prove that a drone actually owns the ID it is broadcasting. The data is encoded in CBOR format to ensure it is compact machine-readable.

- **BRID (RRType 68):** The *UAS Broadcast RID* record serves as a static repository for information that the drone typically broadcasts over Bluetooth or WiFi. Its primary purpose in the DRIP context is to store **Broadcast Endorsements**.

  If a drone is too far away to transmit the full chain of trust (which can be split across multiple broadcast packets), the observer can query the BRID record from the DNS to retrieve these missing endorsements. This acts as a reliable fallback mechanism to ensure verification can proceed even with partial broadcast reception. Like the HHIT record, BRID data is also encoded in CBOR.

2.5 CBOR data format

The Concise Binary Object Representation (CBOR) is a data format focusing on simplicity and efficiency. It it loosely based on the highly successful JSON format having a similar name-value structure, and not requiring a concrete schema for the data. Furthermore it improves upon the JSON limitation of needing to encode values, often in Base64. CBOR utilizes binary data directly and can this way avoid unnecessary overhead.

2.6 PostgreSQL

PostgreSQL is an open source relational database management system utilizing SQL as the interaction syntax. It is designed to handle a huge range of workloads, from single machines to data warehouses with many concurrent users. It guarantees atomicity, and consistency even in the event of errors.

2.7 Previous work

The work that was done last year consisted of the following things.

- Development of the drip-core-c library. This library appears to have functionality to encode and decode Open Drone ID messages over DRIP. The repository was loosely based on the Open Drone ID repository [9].
- Developed three DRIP transmitters for different systems.
- Updated the android app to follow the RFC9575 standard [10].
- Tested the range of BLE 5 and BLE 4 on a Raspberry Pi.

The previous report can be found in the following source [5].

3 Method

The approach used to fulfil the 2025 project goals involved dividing the work based on the selected project goals defined in 1.2. We utilized the existing codebase from previous years as a starting point.

3.1 Public DNS Registry (registries-33)

One of the primary goals for this year is to set up a public DNS registry for drone identification, updating the implementation from previous drafts (registries-17, tested in 2024) to the latest draft, **registries-33** defined by [11].

The methodology for this task is as follows:

1. **Analysis of Previous Work:** Any relevant previous work, including the existing DNS registry implementation and testing framework from the 2023 and 2024 projects, was investigated. Of this material it was identified what would be useful for reusing and what would have to be replaced.
2. **Identified Draft Changes:** A differential analysis between the 'registries-17' draft and the new 'registries-33' draft to identify all changes to the specification, including record types, data structures, and query/response formats was performed.
3. **Implemented Updates:** Modifications to the existing DNS service configurations to conform to the 'registries-33' specification were made. In practice, since the underlying DNS service was decided to be changed from Bind9 to PowerDNS, this involved the completely new configuration of a DNS server and the backend data base for it.
4. **Public Deployment:** The updated DNS data registry was deployed on a public-facing server, making it accessible for testing and interoperability.
5. **Testing and Verification:** Develop and execute a test plan to ensure the public registry is working correctly, can handle queries as specified, and securely store identifier data.

3.2 Crazyflie Firmware Integration

A new objective for 2025 was to add DRIP support to the Crazyflie drone platform. This platform is significantly more resource-constrained than the Raspberry Pi devices used in previous prototypes.

Our approach for this task was:

1. **Environment Setup:** Established a development environment for the Crazyflie, including the toolchain for building and flashing firmware.
2. **Code Analysis:** Reviewed the existing DRIP broadcast prototype (nrf52850dk) to understand its core components, dependencies and data flow.
3. **Implementation:** Added functionality to broadcast DRIP messages to the Crazyflie firmware. This was done in two separate ways:
   (a) Started by checking whether it's plausible to send DRIP messages via BLE from a Crazyflie.
   (b) Sent DRIP messages via P2P between two Crazyflies.
4. **Verification:** Confirmed that the Crazyflie is sending valid DRIP messages. This was done by manually checking the messages.

3.3 Android App - DNS integration

This year we focused on developing more on the DNS in the Android application. We now wanted to make a lookup in the public DNS.

1. **Analyse previous work:** We began by analysing the code base of the application and attempting to pinpoint what needed to change in order for DNS verification to work as intended.
2. **Implementation:** The domain and record type was changed for the application, in order for it to correctly do DNS lookups on the new DNS server.

3. **Verify and document:** To verify that both the application and the public DNS worked, we needed to make correct lookups from the Android application. There were some wrong configurations in the flow that we tried to solve, but did not fully complete.

4 Result

This section will report what results were achieved in the different parts of the project.

4.1 DNS

The DNS were updated in different ways and below are subsections with the different parts.

4.1.1 Identified registries-xx Differences

The previous draft, registries-17, describes the data format for both HHIT and BRID to be of one of several alternative types. The listed options are JSON, Binary blob, and CBOR, noting that CBOR is the preferred alternative. The newer draft, registries-33, now only lists the CBOR data type as a valid format, resulting in a more precise specification.

4.1.2 Publicly Available DNS

This project aimed to set up a public DNS supporting the data types defined in the DRIP framework. Previously, a proof of concept Bind9 DNS server was configured. Due to the lack of support for HHIT and BRID, and any custom types in Bind9 it became apparent that a new alternative would be preferable. PowerDNS was chosen due to its richness in functionality and, at the time, recent pending GitHub pull request regarding adding support for the two new RRTypes. It soon thereafter got accepted and merged into the main branch.

To mitigate the risk of facing compatibility issues during development, it was decided that we would host the DNS server using containers. This way, we could easily run the server on our own computers, but still be confident that they would be compatible with the available public server. The system is set up as two different containers, one running the *powerdns/pdns-auth-50* docker image[2], and one running the official PostgreSQL docker image[3]. The PostgreSQL database is set up with tables using the default schema found in the PowerDNS documentations[4]. The DNS container is given the database name, user, and password for communicating with the backend container. These configurations are stored as a Docker Compose file, allowing for easy deployment or dismantling of the service.

4.1.3 DNS in Android observer app

Before this year, the Android app only had support for a fetching and displaying the response from a local DNS running of the application host. Now there is support for using a public DNS which is hosted on a IDA server, and correctly parsing the response data into a certificate object. We changed the DNSData type in the app to DETData to better reflect what it currently does. We began preparing for changes that would allow for verification of the entire DNS based chain of trust in the app, but had to stop due too little time, reliability issues with the *linux-transmitter*, and non existing infrastructure for the RAA and APEX endorsement verifications.

4.2 Crazyflie

One of the main goals for this year was to enable a Crazyflie to broadcast DRIP messages. The messages in question are DRIP link, DRIP wrapper and DRIP manifest. Two different methods of sending these messages were attempted, broadcasting messages with BLE 4 and P2P. The results of these attempts will be shown below.

4.2.1 Broadcasting messages via BLE 4

To broadcast messages with BLE a new service was added to the Crazyflie 2.1 along existing services. This way the Crazyflie could function normally whilst sending out messages. On top of being able to send messages, it was also implemented for the service to divide messages into multiple shorter messages if the message was too long.

[2] https://hub.docker.com/r/powerdns/pdns-auth-50
[3] https://hub.docker.com/_/postgres
[4] https://doc.powerdns.com/authoritative/backends/generic-postgresql.html

The service was tested by sending a static byte-array of an example DRIP link. To check that the message was sent correctly the NRF connect app was used [8]. The app concatenated multiple messages into one, making it a bit difficult to read the received messages, but it was still possible to figure out that the message appeared to send correctly.

After this service was implemented a discovery was made. Due to limitations on how data is transmitted on Crazyflie using BLE 4 it is only possible to send 20 written bytes, despite 32 bytes being transmitted (12 bytes are predefined). It is defined in RFC9575 that when using legacy transport such as 4.X, a minimum of 23 bytes are needed [10]. This makes it impossible to transmit messages using the RFC9575 legacy format and FEC. Following this realization it was chosen to not continue on working on transmitting BLE on Crazyflie and instead move on to using P2P.

4.2.2 Broadcasting messages with P2P

P2P has a limit of 60 bytes per message, this is not enough to send all messages without splitting in the way defined in RFC9575 [10]. However the limit of 60 bytes is well over the minimum threshold for legacy transport [10], meaning that legacy transport and Forward Error Correction (FEC) is possible. Due to this it was decided that this is the type of DRIP messages that were supposed to be broadcasted with P2P.

The following DRIP messages were implemented in the app-layer of the Crazyflie: DRIP link, DRIP wrapper and DRIP manifest. To be able to broadcast positional data, some example GPS coordinates were added to the evidence in the ASTM messages in the DRIP manifest.

To test the implementation of the messages, the Crazyflie Client which is built by bitcraze was used. By using the GUI in the Crazyflie Client on a laptop and two Crazyflies it was possible to see which messages were received from one Crazyflie to another.

5 Discussion

5.1 Result

In this part the results will be discussed.

5.1.1 Crazyflie

The end result for this project is that sending DRIP messages using P2P works well. Whilst it works as planned, it is somewhat limited. Since P2P only works for communication between Crazyflies, it is not possible to send or receive messages with other entities than a Crazyflie. This downside was known, but still limits the usability of the DRIP transmissions.

Regarding sending DRIP messages with BLE 4, it was not possible to send the messages due to too little data being able to be transmitted at one time. It is not currently a good choice to use BLE to send DRIP messages on Crazyflie. This might change in the future if the amount of data able to be transmitted increases, but unless that happens it would be more fitting to use another device than Crazyflie to send BLE messages, for example a Raspberry Pi.

5.1.2 Publicly available DNS server

The available DNS system is fully functional in an operational sense, but currently lacks relevant data entries. During this project only mock data on an approximately correct format has been used. Furthermore, entering new data into the DNS database is not straight forward. The data has to be entered either through direct communication with the PostgreSQL database using a SQL syntax, or through the PowerDNS backend interface. Either way, the entered data currently has to be encoded on the correct format, i.e. the hexadecimal representation of the Base64 encoded CBOR data.

5.1.3 DNS on the Android app

The Android app currently fetches the DET, including the certificate, necessary to verify HDA endorsements of that drone. It does however not fetch any BRID data, nor actually verify any endorsements. In order to fully verify the drone endorsements, the application would have to also fetch RAA and Apex level certificates, and verify the endorsements of the drone, the HDA, and the RAA.

5.2 Challenges

Some general challenges that came up during the project were the following.

- Unclear goals - Due to us not specifying enough in the beginning of the project, we had to change some of the goals during the project. This lead to us having less time on doing work for the final product and instead wasting time on things that were not going to work or that were not relevant.
- Hard understand existing code base - One challenge with this project is that a lot of the repos on the GitLab do not have a well written README. This means that it is hard to known what each repo does and what their purpose is.

5.2.1 DRIP support for Crazyflie

Here are some of the main challenges encountered when working on implementing DRIP support for the Crazyflie.

- **Debugging:** Debugging a Crazyflie was found difficult. There is a debugger for Crazyflie [3], but we did not manage to get it to work. Without a debugger, the only way to debug was by sending messages and reading them. Using P2P it is by sending messages between Crazyflies and listening to the messages using the Crazyflie Client. This means that to test and debug code at least two Crazyflies and a laptop was needed. This was further complicated by the fact that using multiple debugging logs at the same time causes the Crazyflie to crash.
- **Code libraries:** One challenge when developing on the Crazyflie, is that it is not possible to include external libraries. So, to use a library it must be downloaded into the repository and flashed

onto the Crazyflie. This makes coding harder and generally limits external libraries to micro-libraries which easily can be added to the Crazyflie. This is the cause to why the encryption currently used on the DRIP messages is so insecure.

- **Data limit:** As mentioned previously in the report, Crazyflie only allows for 20 bytes to be transmitted using BLE. This limits the ability to use BLE to transmit messages so much that it was decided that another way so send messages would be better.

5.2.2 DNS

Challenges regarding the DNS implementation.

- **Switch to a new DNS application:** PowerDNS is now used instead of Bind9. It worked a bit differently, but not much, the hard part was just get it fully running and understand the configuration. The switch was made because of PowerDNS is now supporting HHIT and BRID.
- **Distributing the work:** In hindsight, a more significant result could possibly have been achieved if the work was split up between the two students working on the DNS related functionality. That would have resulted in more efficient work, where each student could have focused on a particular part of the system. In practice, it could have resulted in both a publicly available DNS server, and a functional endorsement verification chain in the Android application.

5.3 Advice for future students

Below is a list of advice for future students taking this course.

- Ask either Andrei or teaching assistant(s) for a quick and brief overview of DRIP and it's different messages, structures and entities. It's also a good idea to early in the project formulate clear goals, and check with Andrei or teaching assistants if the goals are okay. This way you have a more clear goal to work against, since (at least for us) the goals were very broad and unspecific in the beginning.
- In 5.3.1 there are some useful links, documents and repositories that should be briefly looked over. Just be careful and check if the documents linked have a newer version, this is especially important for IETF drafts.
- Be aware of that the documentation for much of the material produced during previous years is limited, and that some information might be better to ask the course staff about rather than attempting to find out yourselves.
- Having a good overview of the entire DRIP framework is beneficial even if your chosen goal only handles a sub section of it.

5.3.1 Useful Resources

Here are some useful resources that were used during this years project.

- RFC9575
- RFC9574
- RFC9153
- Crazyflie client
- Crazyflie app-layer
- Drip dki
- Registries 33

6 Future work

In this section are some suggestions for future work.

6.1 Crazyflie

Here are some ideas for future work on the Crazyflie.

6.1.1 Security improvements for P2P DRIP messages

The algorithm used to create signatures when making the DRIP messages is not secure. So one improvement that could be done on the Crazyflie is to switch the algorithm for one with better security.

6.1.2 Replacing example position data

Currently the positional data that is sent in the DRIP manifest message is just example data. So one change that could be to add real positional data in the messages. One method to add positional data to the Crazyflie is the Crazyflie Loco Positional system [2].

Furthermore currently all timestamps such as Valid Not Before (VNB) and Valid Not After (VNA) currently uses hardcoded timestamps. This does not follow RFC9575 standard [10], but was done as Crazyflie does not have any way to get actual time. The only way to have some sort of time currently is using ticks since start, however one future approach could be exploring is using the Crazyflies GPS Deck to get a Unix timestamp.

6.1.3 Changing DRIP broadcasting method

The DRIP broadcasting method currently used is P2P. This works well for communication between Crazyflies, but cannot be used to communicate with other objects currently. There is therefore room for improvement to either switch broadcasting method or see if it is possible to use P2P to communicate with objects other than Crazyflies.

There may not be a better way to broadcast DRIP messages using Crazyflie 2.1, so this might require switching to something like a Raspberry Pi or maybe a newer version of Crazyflie.

6.2 DNS

Here is a section about DNS related future work.

6.2.1 Full Integration with Android Observer

To fully test the public DNS, the Android application needs to verify the drone endorsements. The flow of the lookup against the DNS is not fully implemented, and would require a larger change in the control flow of the application. This is since both endorsements of the drone by HDA, HDA by RAA, and RAA by APEX have to be verified in order to cover the entire chain of trust. In practice this requires retrieving separate certificates for each of the levels. Each certificate has to be fetched from a DNS hosted by a separate entity, making it difficult to test until the entities for RAA and APEX endorsements have functional DNS systems in place.

6.2.2 Implementation of DNSSEC

A critical next step is the implementation of DNSSEC (Domain Name System Security Extensions). According to the `drip-registries-33` draft [11] , DNSSEC is mandatory for Apex entities (root of trust) and recommended for others.

Implementing DNSSEC offers significant advantages for the DRIP verification process:

- **Simplified Verification:** Without DNSSEC, the client (Observer App) must manually ”walk” the tree of certificates, performing multiple DNS lookups to verify the chain of trust from the drone up to the root.

- **Data Integrity:** DNSSEC provides cryptographic authentication of the DNS data itself, preventing spoofing attacks where an attacker might inject false DET records.

Since PowerDNS was chosen for this year's implementation and also has built-in support for DNSSEC. It should theoretically be easier to be implemented in than with BIND 9.

6.2.3 Dynamic Registration API

Currently, the registry relies on manual entry of records. PowerDNS has a built-in REST API which can be used to add records more easily. Better would be to make an API that wraps PowerDNS. The best is to discuss this with a supervisor to see what is needed.

6.2.4 linux-drip-transmitter

When we used the *linux-drip-transmitter* we had problems with the hardcoded data in the transmitter and some wrong auth types in the transmitter. This made it difficult for the android app to verify if it is working, which then leads to a difficult task to verify the DNS. Continue developing the transmitter parallel to the android app.

References

[1] *BIND 9 Administrator Reference Manual*. Accessed: 2025-12-03. Internet Systems Consortium (ISC). 2025. URL: https://bind9.readthedocs.io/en/latest/.

[2] Bitcraze AB. *Loco Positioning system — Bitcraze Documentation*. https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/. Accessed: 2025-12-05.

[3] Bitcraze AB. *OpenOCD and GDB Debugging — Crazyflie Firmware Documentation*. https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/development/openocd_gdb_debugging/. Accessed: 2025-12-05.

[4] Stuart W. Card, Adam Wiethuechter, Robert Moskowitz, and Andrei Gurtov. *Drone Remote Identification Protocol (DRIP) Requirements and Terminology*. RFC 9153. Feb. 2022. DOI: 10.17487/RFC9153. URL: https://www.rfc-editor.org/info/rfc9153.

[5] Jiajun Chen, Fabio Crugnola, Albin Svärd Gruvell, and Liza Johansson. "TDDE21 DRIP 2024 Authentication Formats and Protocols for Broadcast Remote Identification". In: (2023).

[6] G. Koulouras, S. Katsoulis, and F. Zantalis. "Evolution of Bluetooth Technology: BLE in the IoT Ecosystem". In: *Sensors* 25.4 (Feb. 2025), p. 996. DOI: 10.3390/s25040996.

[7] E. Larsson-Kapp, V. Kniivilä, Z. Wang, M. Wzorek, A. Lemetti, and A. Gurtov. "Trust-Based Collision Avoidance for Unmanned Aircraft Systems". In: *2024 IEEE International Conference on Aerospace and Signal Processing (INCAS 2024)*. Presented at the 4th International Conference on Aerospace and Signal Processing, Nov 28–30, 2024. Cusco, Peru, Nov. 2024. DOI: 10.1109/INCAS63820.2024.10798560. URL: https://doi.org/10.1109/INCAS63820.2024.10798560.

[8] Nordic Semiconductor. *nRF Connect for mobile*. https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-mobile. Accessed: 2025-12-05.

[9] OpenDroneID Project. *opendroneid GitHub web page*. https://github.com/opendroneid. [Online; accessed 2 December 2025].

[10] Adam Wiethuechter, Stuart W. Card, and Robert Moskowitz. *DRIP Entity Tag (DET) Authentication Formats and Protocols for Broadcast Remote Identification (RID)*. RFC 9575. June 2024. DOI: 10.17487/RFC9575. URL: https://www.rfc-editor.org/info/rfc9575.

[11] Adam Wiethuechter and Jim Reid. *DRIP Entity Tags in the Domain Name System*. Internet-Draft draft-ietf-drip-registries-33. Work in Progress. Internet Engineering Task Force, Aug. 2025. 35 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-drip-registries/33/.