Linköping University | Department of Computer and Information Science
Bachelor's thesis, 15 ECTS | Datateknik
2022 | LIU-IDA/LITH-EX-G–22/061—SE

Formal security verification of the Drone Remote Identification Protocol using Tamarin

Formell säkerhetsverifiering av Drone Remote Identification Protocol med hjälp av Tamarin

Jakob Ahokas
Jonathan Persson

Supervisor : Andrei Gurtov
Examiner : Marcus Bendtsen

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The current standard for remote identification of unmanned aircraft does not contain any form of security considerations, opening up possibilities for impersonation attacks. The newly proposed Drone Remote Identification Protocol aims to change this. To fully ensure that the protocol is secure before real world implementation, we conduct a formal verification using the Tamarin Prover tool, with the goal of detecting possible vulnerabilities. The underlying technologies of the protocol are studied and important aspects are identified. The main contribution of this thesis is the formal verification of session key secrecy and message authenticity within the proposed protocol. Certain aspects of protocol security are still missing from the scripts, but the protocol is deemed secure to the extent of the model. Many features of both the protocol and Tamarin Prover are presented in detail, serving as a potential base for the continued work toward a complete formal verification of the protocol in the future.

Acknowledgments

We would like to thank our supervisor Andrei Gurtov for giving us the opportunity to carry out this project, as well as for guidance along the way. We would also like to thank Suleman Khan and Felipe Boeira for additional guidance, feedback and discussions regarding the project.

Contents

List of Figures

List of Tables

List of Acronyms

UA Unmanned aircraft (Only the drone).

UAS Unmanned aircraft system (The drone + the operator).

DRIP Drone Remote Identification Protocol.

RFC Request for comments.

PCS Post-Compromise Security.

IETF Internet Engineering Task Force.

HIP Host Identity Protocol.

HI Host Identity.

HIT Host Identity Tag.

HHIT Hierarchical Host Identity Tag.

BEX Host Identity Protocol Base Exchange.

DH Diffie-Hellman.

RID Remote Identification.

BLE Bluetooth Low Energy.

WiFi NaN WiFi Neighborhood Area Network.

HID Hierarchy ID.

RAA Registered Assigning Authority.

HDA Hierarchical HIT Domain Authority.

DET DRIP Entity Tag.

EdDSA Edwards-Curve Digital Signature Algorithm.

MITM Man-in-the-middle.

1 Introduction

As more and more aspects of society implement technology and the internet, cybersecurity is gaining traction and becoming more important than ever before. Many industries are starting to realize the devastating effects of cyberattacks and are focusing more on cybersecurity, developing new protocols and protecting their data as best they can. One branch that is lacking in cybersecurity, however, is the aviation industry. Most messages sent between aircraft and ground controllers are not encrypted at all, and therefore can be intercepted and read by anyone. The case is even worse for unmanned aircraft (UA) such as drones. When these drones send messages regarding their position, they can be freely read and since they implement no form of authentication, anyone can easily spoof these messages and broadcast false information [4].

Currently, drones use the ASTM Remote ID specification for creating messages containing information about the drone [4]. However, these messages do not currently provide secure authentication of the drones. In order to further improve this specification a new protocol, the Drone Remote Identification Protocol (DRIP) has been proposed in order to provide a secure way to remotely identify unmanned aircraft.

1.1 Aim

The purpose of this thesis is to implement and formally verify the security of DRIP using the state-of-the-art verification tool Tamarin Prover. Once DRIP is fully incorporated into real world usage, the security of standard for Drone Remote Identification will be drastically improved. Both civilians and officials stand to benefit from being able to capture data sent by drones and thereby identify them, their positions and their owner. Beyond security, the protocol would also add a form of verification of the drones, meaning that impersonation attacks are harder to carry out. The proposed DRIP scheme will be thoroughly tested using Tamarin Prover, with the purpose of formally verifying the protocols security features and underlying algorithms. This tool has previously been used to formally verify the encryption protocols such as TLS 1.3 [13], so using Tamarin to prove the security of DRIP will grant increased credibility to our results.

There are technical limitations that prevent us from fully testing the proposed DRIP scheme that may influence the results of our work, the main one being that Tamarin is a relatively new tool with limited resources and documentation relating to certain DRIP-specific features and attacks. Additionally, DRIP is as of writing this thesis currently just a proposal and is in the process of being reviewed by authorities for approval. As such, certain details may get revised and other features may get added to DRIP that were not taken into consideration when writing this thesis.

1.2 Research questions

In preparation for the work, the following research questions were formulated;

- How does formal verification of a protocol work?
- Is the proposed Drone Remote Identification Protocol secure?

1.3 Approach

First, the proposed DRIP will be studied through RFC's and previous scientific work. Once a sufficient understanding of the protocols architecture is achieved, further study will be put into important details of the protocol such as encryption and signature algorithms. A thorough understanding of how to model and prove security protocols using Tamarin Prover is also required. Next, a detailed schema of the important parts of DRIP will be created to use as reference when modelling DRIP in the Tamarin language. When complete, the security of the model will be examined to identify possible attacks. If any limitations to this protocol are found these will be listed and, if possible, addressed.

2 Background and Related Work

Security protocols exist to protect data being sent over a network from adversaries that want to gather information or insert fake information on the data stream. Without such protocols, it would be possible for any entity in a network to read any message being sent between any other two entities. In case the data sent is sensitive this could have disastrous consequences. If no security is implemented, it is also easy for adversaries to send messages imitating one of the participants in a connection. This way, malware can be inserted into the information stream and cause further harm. Security protocols exist to mitigate the possibility of such attacks by way of encryption of data, setting up secure sessions between users and creating means of authentication of the original sender. In this way, compromised messages or totally fake ones can be identified and either flagged to the user or ignored altogether.

2.1 Classification of attacks

Security protocols are usually designed with two or more roles in mind that communicate with each other, for example a client and a server. Different roles follow different rules while establishing a secure connection and during the communication itself. In addition to these set roles, there are also attackers which do not necessarily have to follow any of the rules. In the context of security protocols, attackers can be divided into two main groups, passive attackers and active attackers. Passive attackers operate from the shadows and stealthily intercept and listen to communication, while active attackers can alter, impersonate, delete and redirect messages. Further categorization of attacks is possible, such as dividing them into attacks that try to break the underlying cryptographic properties or attacks that exploit the general architecture of the protocol [5]. When formally verifying the security of a protocol, the hope is to detect all types of attack, but one may need different tools to detect different types of weaknesses.

2.2 Post-compromise security

Post-compromise security (PCS) addresses how a protocol handles previous successful attacks on a participating entity. Given two users running a protocol, what happens if one has previously had their secrets leaked to an adversary? This concept has been defined and examined by Cohn-Gordon et al [11]. PCS can be achieved in several different ways when one or more of the communicators have been compromised. The standard case often considered, is one where the two communicators' secrets must remain secure but every other long term key can be compromised. There are however also protocols that achieve PCS through their perfect forward secrecy, meaning that long term keys can be compromised after the session has ended without risk for the session's contents to be in danger. Even cases where one agent's long term key has been revealed to an adversary may implement a form of PCS, if there has previously been a secure session between the two agents where they agreed upon a shared value. If this value is used in the new session, the adversary will still not be able to impersonate the agent. In the context of DRIP, PCS would mean that even if a long term key of a drone operator has been compromised it could still set up secure sessions in the future without risking leakage of vital drone information.

2.3 The Dolev-Yao model

This thesis will examine DRIP's security using Tamarin Prover, which implements symbolic protocol models also known as Dolev-Yao models. The focus of these models is on two-party communication protocols and secrecy properties, for example key exchanges. A sufficient understanding of Dolev-Yao models is a prerequisite for understanding the Tamarin language. A main point of these models is that the honest parties are stateless. This means that they do not retain knowledge from previous messages. In every step of the protocol, which are known as rules in the Tamarin context, the parties only have access to their initial knowledge and the message they just received in that exact step. The adversary on the other hand, is able to store values and maintain states. Another key feature of Dolev-Yao models is that they allow for an arbitrary amount of concurrent protocol executions. Each party can partake in many different communications with several parties, which allows the adversary to have more options to exploit the system. This is one of the reasons why the symbolic Dolev-Yao model gained popularity. When it was developed, similar models mostly focused on a single execution of the protocol. Examining a single execution can be useful but it does not cover all bases, as many of the harder to detect vulnerabilities exploit multiple sessions of the same protocol, for example oracle attacks [29].

The adversary has great power in Dolev-Yao models, and is basically the network itself. The adversary has access to public information or any message sent over the network, and is able to receive, forward, block and replay messages. Honest parties are not able to know if the message they received was from another honest party or if it was in fact received from the adversary. This places a strong focus on encryption and signing to prove the integrity of the message. However, in these models the cryptographic properties are assumed to be completely secure, meaning that encrypted data cannot be decrypted by the adversary without having the correct decryption key. The adversary has full control of the public channels but they cannot guess secrets or otherwise break the encryption [14]. As such, this thesis will only examine the security of the general architecture of DRIP, not the underlying cryptographic algorithms.

2.4 Current state of drone remote identification

Current drone identification implements no form of security protocols, meaning that spoofing signals sent from UAs is trivial. Previous studies such as a paper by Bunse and Plotz from 2018 [8] show the relative ease that UAs can be attacked with. Attacks range from simple denial of service attacks to intercepting and actually taking over control of the drone. Identification broadcasts are equally vulnerable. This needs to be addressed, as there is no value in drones having an identification broadcast that cannot be trusted. One way of remedying the issue is by implementing the proposed Drone Remote Identification Protocol (DRIP). The main idea of DRIP is to introduce an integrity check for the identification broadcasts of the drones. The messages are broadcast and can then subsequently be picked up by observers in the area. These observers are then supposed to be able to look up information about the UA in some sort of database. The full implementation details of this database are not yet finalized, but there are some suggestions. One proposed way of fulfilling the database requirements is by using the permission-based blockchain framework Hyperledger Iroha [18]. When receiving a broadcast from a drone using DRIP, the observer is ensured that the information received from the broadcast is trustworthy and that the drone is who it claims to be.

DRIP is being developed by the Internet Engineering Task Force (IETF). The IETF is an international organization working toward making the Internet better [2]. This is accomplished through publishing technical documents for specifying new proposed standards for development and management of the Internet. The DRIP drafts are such experimental documents which propose how to implement Remote Identification of UAs. IETF is an open organization, meaning that anyone interested can read into ongoing work and also participate [2]. This means that work carried out by IETF is publicly available on the Internet, resulting in easy access to necessary documents when working with the concepts discussed by the organization.

In addition to increasing the security of remote identification of drones, DRIP aims to create a globally recognized standard for this purpose. Seeing as different regions today have different requirements regarding what types of drones must identify themselves as well as how this identification is carried out, this would make using imported drones easier everywhere. For example, in the European Union, only some sizes of drones are currently required to adhere to the regulations regarding remote identification [34]. However, as work with the proposed U-Space in the European Union carries on, more network identification of UAs may become mandatory [35]. U-space is a proposed type of service, where drones are efficiently and securely automated to carry out tasks in an airspace. Examples of such tasks could be delivery of goods, food or healthcare. As such, current DRIP drafts are addressing the requirements made by the European Union in order to fulfill the security and functionality needs for U-space.

2.5 Related work

As the underlying cryptography and architecture of DRIP have not been thoroughly proven to be fully secure, a formal verification of the protocol is needed to ensure protection against all different types of attacks. DRIP builds upon and extends another technology called the Host Identity Protocol (HIP) to deliver its identifier [10]. Previous research has been conducted to formally verify HIP back in 2005 by Tschofenig et al. using the AVISPA verification tool [33]. A newer verification using a more modern tool while also implementing DRIP specific features would be of great benefit for the credibility of DRIP. The newer, state-of-the-art tool Tamarin Prover is used in this thesis to formally verify DRIP and parts of the underlying HIP protocol. Tamarin has previously been used by Mäurer et al. [24] to verify a station-to-station cell attachment procedure of another aviation protocol, LDACS, making Tamarin an obvious choice for the formal verification of DRIP. This thesis implements parts of their modelling procedure, such as building a clear two-party step-by-step schema of the protocol to serve as a base for Tamarin rules.

There is a wide variety of attacks and security properties to examine in Tamarin, some of which were tested on a vehicular group protocol called Ensemble by Boeira and Asplund [7]. In the paper, the security properties were modeled as lemmata and subsequently proven secure with the goal of ensuring liveness, secrecy and authenticity of the protocol. By using Tamarin, they formally verified Ensemble and managed to identify parts of the protocol that were susceptible to exploitation. Boeira and Asplund ran their lemmata four times with different configurations, mostly to examine the effectiveness and computational resource consumption of the different configurations. The verification of DRIP presented in this thesis does not employ any of these strategies and does not use helper-lemmata or oracles, as the DRIP verification is not as computationally demanding.
Our work takes inspiration from Boeira and Asplund's verification strategy when trying to implement the different forms of secrecy and authenticity in Tamarin lemmata. Their paper follows Lowe's hierarchy for authenticity properties, namely liveness, weak agreement and non-injective agreement are modeled via lemmata. DRIP functions a little differently than the vehicular group protocol though, as Ensemble only allows a single session per vehicle. This is the reason why only non-injective agreement is modeled but not injective agreement. In the case of DRIP, it would be more interesting to model the stronger constraint in the form of injective agreement. Successfully modelling injective agreement allows for a more thorough security verification as DRIP allows entities to establish multiple concurrent sessions.

3 Theory

This section will introduce and give a theoretic background to both DRIP, Tamarin and their underlying components, which is necessary to understand the rest of the thesis. High-level overviews and in-depth details will be given for both technologies, as having knowledge of both is useful for understanding the protocol model and security proof.

3.1 Host Identity Protocol

When a drone using DRIP broadcasts its ID to nearby observers, it does so in the form of a Hierarchical Host Identity Tag (HHIT). This is an extension of a regular Host Identity Tag (HIT) which is a part of the the Host Identity Protocol (HIP). HIP is a key feature to ensure the security of DRIP, and is used for ensuring integrity and authentication. This is achieved by generating a Host Identity (HI) from the public/private key-pair of every entity and using this HI to verify and authenticate messages sent by this entity. The HI is essentially just the public key of the entity, and the goal of the HI is to directly serve as an identifier of the host sending a packet. The problem with this however, is that there are many different public key algorithms that can be used to generate a key-pair, with many of them resulting in public keys of different length. Having identifiers of different length is not desirable and may lead to problems. Consequently, HIP solves the length problem by introducing Host Identity Tags (HITs). HITs are derived directly from the HI by using a hash-function to ensure a uniform length of 128 bits for all HITs. These tags can be broadcast to observers in a network as a form of identification, similar to a license plate on a car. There are several key security properties of HITs that make them desirable to use for DRIP broadcasting: HITs are self-certifying, the probability for a collision of HITs is extremely low, and HITs are the same length as a standard IPv6 address and can thus be used in protocols with fields designed for that specific length [27].

3.1.1 HIP Base Exchange

Another key feature of HIP that makes it useful for DRIP is the HIP Base Exchange (BEX). BEX is used by HIP to create an association and establish a secure connection between two entities. It is predominantly based on the Diffie-Hellman key exchange with some additional features to prevent denial of service attacks. BEX has two involved parties, the initiator and the responder. The exchange has four stages with each stage representing a packet sent between the two parties. The key points of BEX is shown in Figure 3.1 Here, "Key" refers to the HI public key and "Sign" is a signature using that key. Certain parameters of the sent packets is not shown in this figure.

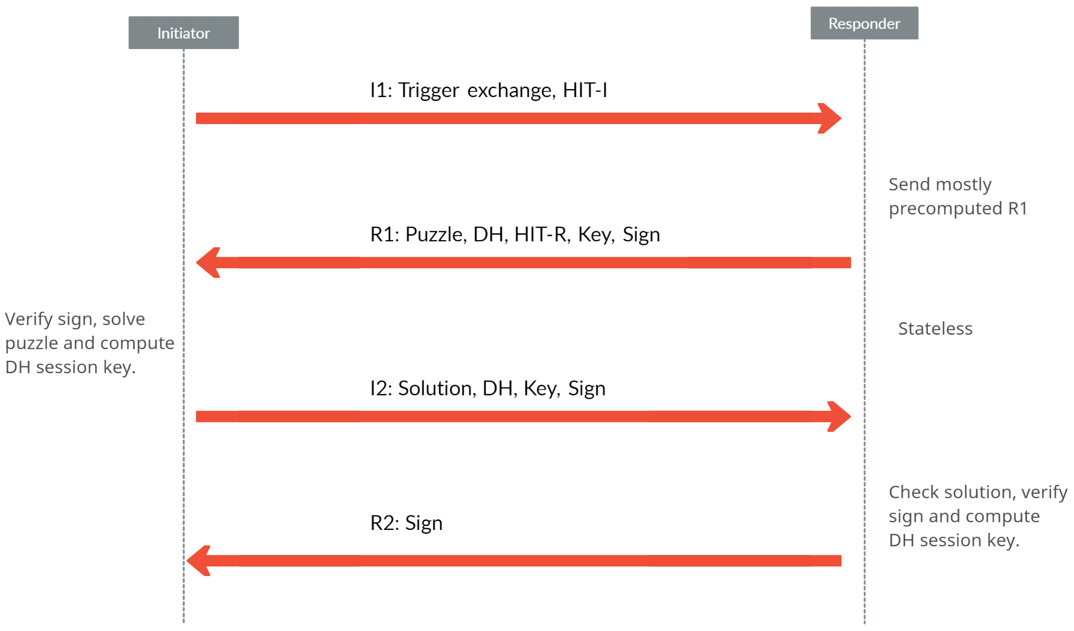

Initiator
Responder
I1: Trigger exchange, HIT-I
Send mostly precomputed R1
R1: Puzzle, DH, HIT-R, Key, Sign
Verify sign, solve puzzle and compute DH session key.
Stateless
I2: Solution, DH, Key, Sign
Check solution, verify sign and compute DH session key.
R2: Sign

Figure 3.1: HIP Base Exchange message procedure

The first message, I1, contains very limited information as the purpose of this message is just to start the exchange. It contains the HIT of the initiator as well as the initiators preferred Diffie-Hellman (DH) parameters. The responder replies with R1, containing a cryptographic puzzle, the responders preferred DH parameters, their Host Identity Tag and their public key. This packet is also signed to ensure integrity. The puzzles purpose is to deter denial of service attacks. In I2, the initiator has to send the solution to this puzzle and only if it is correct will the responder continue with the exchange. Since R1 contains mostly precomputed information that the responder has on hand, the responder does not have to do any real work until the initiator sends the solution. In this way, the computational burden is at first only on the initiator and thus it is much harder to perform denial of service attacks [27] [3].

3.2 Drone Remote Identification Protocol

Drone Remote Identification Protocol (DRIP) is a protocol that aims to improve upon the way drones can be remotely identified by an observer. There are several proposed requirements in order to provide secure and reliable identification. Unmanned aircraft must have a unique identifier, similar to license plates on cars. This identifier is broadcast from the drone via WiFi or Bluetooth in the form of a Host Identity Tag from the Host Identity Protocol. The tag utilizes hashing in order to create a standard value for identifying hosts. This due to the fact that the HIT is always 128-bits, meaning that different underlying technologies will always yield the same tag, as opposed to using a host's identity public key [10]. DRIP must also provide verification to ensure that this ID is in a registry to allow lookup, and also specify in which registry. DRIP needs to support using remote identification (RID) to access key information about the drone, such as model, size and location. It is of utmost importance that this identifier is verifiable and trustworthy and that it cannot be spoofed.

Additionally, it should also be possible to access information about the owner of the drone and whether the operator is trusted or not. There is also a key separation in lookup methods in that there must be both a public and a private lookup method. The general public should only be able to access certain public information about the drone, while cognizant authorities such as law enforcement and military need access to private information about the owner of the drone [9].

A high-level overview of the structure in DRIP can be seen in Figure 3.2

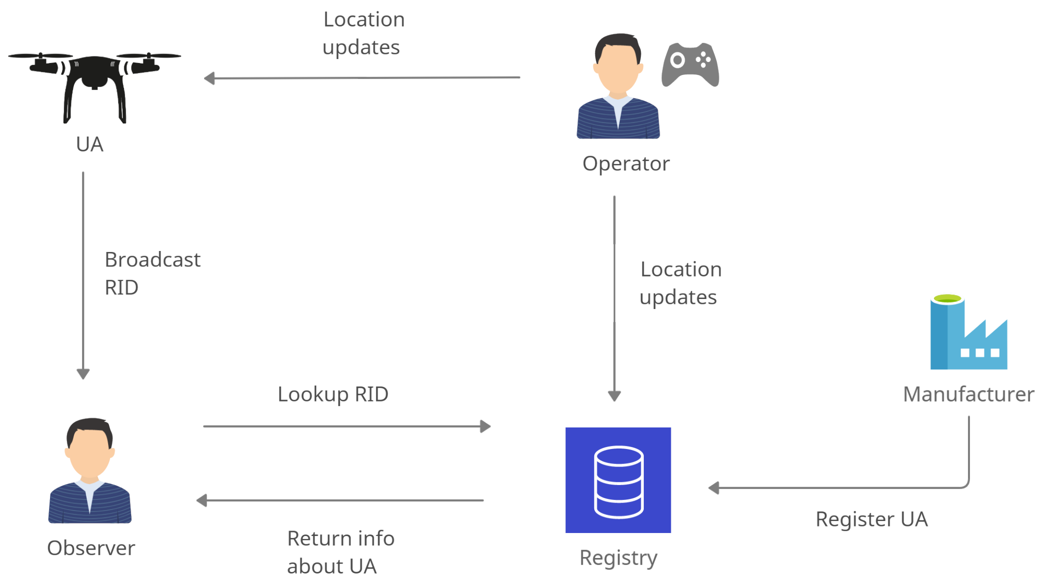


Figure 3.2: An overview of the parties involved in DRIP communication and how they relate to each other.

3.2.1 DRIP broadcasting

Bluetooth is used in DRIP for its Bluetooth Low Energy (BLE) feature that has been around since Bluetooth 4 was released. BLE lets a device broadcast a message to other devices in the area. The size and range of these broadcasts were initially quite small, but with the release of Bluetooth 5 longer ranges and larger packet sizes were made possible. With the help of commands found in the Bluetooth Core Specification [17], a device can be configured to broadcast a certain message with a specified interval. Thus, BLE can be used to broadcast the drone identification message to all nearby devices. If combined with an application that receives these messages and inserts them into a database, quick lookup of a drone's identity and location can be made in said database.

In WiFi networks, when trying to connect to a network, a device will utilize the WiFi Beacon functionality, where a router usually sends out a beacon in order to allow devices to connect to it as well as broadcast messages about received packets. Similarly, the WiFi NaN (Neighborhood Area Network) can be used in order to connect two devices without the need for an access point between the two. Using the "publish" and "subscribe" functions, one device can broadcast a service, say identification messages, that the other device will receive, as it is listening for that service [16]. This can provide the functionality needed for broadcasting identification messages from a drone via WiFi. A drone will "publish" its information and all nearby entities will "subscribe" to this service, thus identifying and tracking the drone remotely.

These remote identification technologies are preferred over existing technologies used in manned aircraft due to the risk of congestion should a large amount of drones send data over the same technology as planes flying in the airspace [28].

3.2.2 Hierarchical Host Identity Tags

Hierarchial Host Identity Tags (HHITs) are a form of identifier in the form of self-asserting IPv6 addresses, created to extend the capabilities of HITs as identifiers in Remote Identification [1]. A HIT lacks the ability to be registered to a specific Host Identity (HI), and thus, different HIs might give the same HIT. This means that lookup of the drone identity is not possible using just a HIT. Including text-based information about the hierarchy in which the tag is registered, means that no other tag with the exact same ID will be accepted into any registry, should a hash-collision occur. The HHITs are built up of a number of fields of data. These fields are;

- An IANA (IPv6) prefix.
- A Hierarchy ID (HID), which can be used for organization of HITs into a domain.
- A HHIT Suite ID, which includes information about Host Identity and hash algorithms.
- An ORCHID hash, which can include extra information, such as the hierarchical information necessary for a HHIT.

The HHIT is visualized in figure 3.3 according to [1].

In the context of Remote Identification (RID) of drones, HHITs are often referred to by the Internet Engineering Task Force as DRIP Entity Tags (DETs). The extra information included in the DET is contained in a generated Overlay Routable Cryptographic Hash Identifier in accordance with [21].

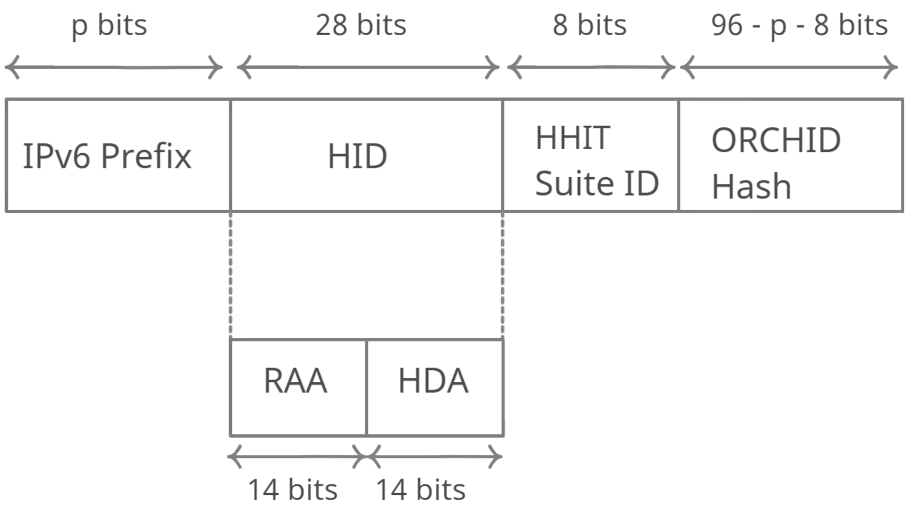


Figure 3.3: A Hierarchical Host Identity Tag as described in [1].

3.2.3 Hierarchy ID

A Hierarchy ID (HID) enables organization of HITs into different domains, and is built up through a combination of Registered Assigning Authority (RAA) and Hierarchical HIT Domain Authority (HDA) [1]. A HDA is an organization that handles the services needed for UAS registry management. A RAA, on the other hand, is an organization that manages a registry of HDAs. These two thereby work together to provide the hierarchy needed for DRIP to implement unique HHITs across different registries, in other words enabling global uniqueness of drone IDs.

3.2.4 EdDSA authentication

Edwards-curve Digital Signature Algorithm (EdDSA) is a signature scheme designed to be fast and secure. EdDSA utilizes Twisted Edwards curves when generating keys [19]. When the scheme is used, first the following are chosen;

- A finite field F over an odd prime power p
- An elliptic curve E over F, where the group $E(F)$ of F-rational points has the order $\#E(F) = 2^c l$, where l is a large prime and 2^c is the cofactor.
- A base-point $B \in E(F)$ with order l
- A hash function H with an output of $2b$-bits, where $2^{b-1} > p$

A private key k in EdDSA is a randomly chosen b-bit string. The public key for k is noted as $A = s \cdot B$, where s is the b least significant bits of $H(k)$ as an integer. This means that A is a curve point of $E(F)$ encoded in b bits [19].

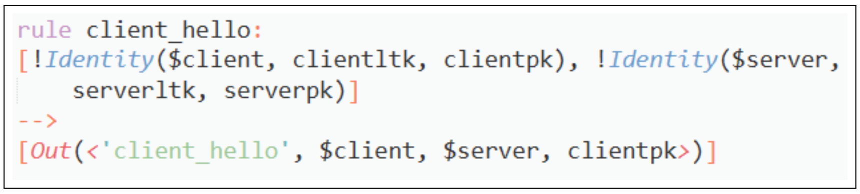
EdDSA's role in DRIP is to create an extended version of HIT's Suite ID for HHITs, where the Suite ID is updated from 4 bits to 8 bits. These Suite IDs specify the Host Identity as well as the hash algorithm used for generating keys [1].

3.3 Tamarin Prover

Tamarin Prover is a tool used for analysing and modeling security protocol using multiset rewriting rules. Tamarin can automatically detect weaknesses and possible attacks on the input, which is a security protocol model made up of rules and lemmata describing the protocol's desired properties. If Tamarin detects an attack that would violate the rules of the protocol, a counterexample will be generated showing the exact order of steps taken in the form of an attack graph. If no attacks are detected by any combination of parallel actions and instances, Tamarin can also construct a mathematical proof showing that the protocol fulfills the specified properties [32]. Previously, Tamarin has been highly successful and has been used to verify the Transport Layer Security (TLS) protocol, where vulnerabilities were found and subsequently handled [13].

3.3.1 Modeling a protocol

There are three key components to model a protocol in Tamarin. These components are terms, facts and multiset-rewrite rules. A term is a message that is sent between two roles, such as between a client and a server. Messages sent can contain constants, variables, or a mix of the two. The final line in figure 3.4 is a term containing the *client_hello* -message and the public key. Facts can be used to describe what internal states different roles are in, to serve as sort of a checkpoint. There are also a few built-in facts in Tamarin, with the *Out()* fact being used in the example below. Finally, a multiset-rewrite rule describes a transition from one state to another of a protocol. For the purposes of this paper, whenever rules are mentioned in a Tamarin context, it is always multiset-rewrite rules. Modeling a protocol is typically done by specifying different rules for the protocol. The rules represent the actions the participants take, what information they have, what they transmit and what they receive. These rules specify certain parts of the protocol, such as how public and private keys are generated or describing the handshake process between a client and a server. The rules are defined using different states, showing exactly what messages will be sent over the network, as well as what information the adversary has access to [31]. An example of a rule for a client initiating a handshake with a server can be seen in figure 3.4.



```
rule client_hello:
[!Identity($client, clientltk, clientpk), !Identity($server,
    serverltk, serverpk)]
-->
[Out(<'client_hello', $client, $server, clientpk>)]
```

Figure 3.4: Client Hello rule in Tamarin language

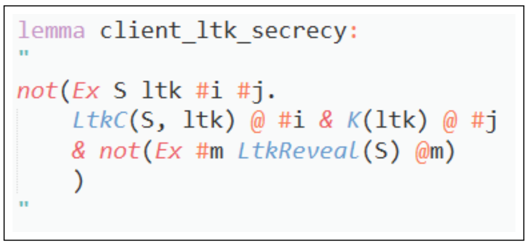
Rules are declared in the form of a left hand side and a right hand side with an arrow between them. These two sides represent two different states of the protocol, with the left hand side describing the situation before the rule is executed, and the right hand side showing the state after execution. Information sent out to, and received from the network is modeled using the `Out()` and `In()` facts respectively. Any information sent out over the network at any time using the `Out()` fact will be treated by Tamarin as information known by the adversary.

In the example above we start with two identities, which can be seen as data structures having a name, a long-term key and a public key. The exclamation marks before the identities signify that these entities are persistent, meaning that they do not disappear after executing the rule [26]. The client and server will of course still exist after initiating a handshake, hence we prefix it with an exclamation mark. Any facts that are not in the right hand side or marked as persistent will otherwise not exist after executing the rule. After execution, a hello-message has been sent out to the network from the client to the server. For a full picture, one must also write rules for generating identities and for the server receiving the hello-message. Important to note is that as previously mentioned, these are multiset rules. In a multiset, elements are non unique and several instances of the same element can exist more than once, meaning that Tamarin can run these rules as many times as it wants, in any order that it wants with the purpose of finding a security breach [30].

3.3.2 Security properties

The properties of the security protocol that one wants to evaluate can be expressed using lemmata. This essentially means that one must formally describe each security feature of the protocol in smaller chunks, similar to the rules mentioned in the previous section. The difference between rules and lemmata is that rules describe how the protocol works, while lemmata are the security goals one wants to examine if they hold using the rules. For example, one might have a protocol that guarantees the secrecy of the clients long-term key when connecting to a server. One can then write a lemma that specifies that there cannot exist a trace where a client has set up a connection with a server, and the adversary has learned the long-term key of the client unless it has access to the server.

For instance, the example lemma described above may be modeled as in figure 3.5.



```
lemma client_ltk_secrecy:
"
not(Ex S ltk #i #j.
    LtkC(S, ltk) @ #i & K(ltk) @ #j
    & not(Ex #m LtkReveal(S) @m)
    )
"
```

Figure 3.5: Long term key secrecy lemma in Tamarin language

While this may look abstract at first glance, it can be broken down into more understandable text. The whole lemma is encapsulated by the `not()` function to which the input cannot be true. In this case, the input states that there exists a session *S* between a server and a client with long-term key *ltk*. The long-term key is used by the client for the session. The adversary is globally modeled in Tamarin by a set of multiset rewriting rules. In practice, the knowledge of the adversary is represented by the *K()* action fact. Here it is stated that the adversary knows the long-term key. Finally, it is stated that a long-term key reveal has not occurred on the session *S*. The *not()* function which encapsulates the whole lemma negates the whole input and says it cannot be true. In essence, it is stated that it cannot be that a session exists between a client and a server, such that the adversary knows the clients long-term key without it already having been revealed. Temporal variables *i, j, and m* are also declared. These represent time points for when actions occur. Temporal variables are a necessary part of setting up lemmata, specifying if actions take place at the exact same time or if they occur separately. It is also possible to order temporal variables, stating that a certain action must always happen before another [32].

3.3.3 Proofs in Tamarin

Finally, after modeling the security protocol using rules and specifying what security properties one wants to test using lemmata, Tamarin can start trying to prove or disprove the given lemmata. The two main types of lemmata that exist are `all-traces` lemmata and `exists-trace`. The difference between these two is that `all-traces` is proven to be true if the examined property hold for all traces while `exists-trace` is true if there exists just a single trace that fulfills the lemma. Tamarin proves lemmata via constraint solving. There are two main methods of proving lemmata that one can use, there is the option of manually guiding Tamarin in what to do next or one can use the auto-prove feature. Tamarin's auto-prove automatically chooses the next fact to examine using preconfigured based on heuristic methods and equational reasoning.[32] If this does not succeed or if one has a certain trace they want to examine for a `exists-trace` lemma, manual proofs can be an effective option. Assuming everything is set up correctly, Tamarin will either give a proof that the property holds, or give a specific counterexample that violates the property [15].

Tamarin also has a graphical interface that can be accessed via a locally hosted web server. There, one can view the proofs and attack-graphs, as well as additional options to alter the heuristics used.

3.4 Protocol security

Security of a protocol can be defined in multiple ways depending on the context. When discussing the security of an Internet protocol, it is most often defined as no adversary being able to receive secret messages or inject false messages. This can be achieved in multiple ways, for example by ensuring privacy of the cryptographic keys used for encryption of messages sent between communicating users. By ensuring the data sent is safe, several different types of attacks on the protocol can be mitigated and it can as such be classified as "secure" in the aspect of those attacks. G. Lowe presents several different forms of authentication, meaning that all messages are verified to come from the correct sender without being tampered with [22]. One example of such authentication forms is aliveness, which means that if a client A has run the protocol with a client B, then B must have run the protocol at some point. This authentication guarantees that random clients can not be impersonated by an adversary, essentially meaning that a random identity can not be generated to trick clients. Further authentication is provided by agreement. Agreement can cover several different cases and protect against different types of attacks. For example, weak agreement is an improvement upon aliveness where every time an initiator A runs a protocol with a responder B, the responder has at some point also run the protocol with A. This type of agreement does not necessarily mean that the two participants have had the same roles in both runs of the protocol. Authenticating that the runs of the protocol are in fact in the same session is called injective agreement. There is also non-injective agreement, where the roles are correctly corresponding but the runs may not have occurred in the same session.

4 Method

To gain an understanding of the work, the proposed version of DRIP was studied through RFC's and previous related works, such as the report written in the fall of 2021 by Suleman Khan et al [20]. In order to properly write a Tamarin model for the proposed scheme, a deeper understanding of both DRIP and Tamarin was required. The Tamarin Prover manual [32] proved to be helpful for understanding the basics of the tool, however specific instructions for implementing certain features not discussed in the manual were harder to find. Previous Tamarin work in the form of a formal verification of a cell attachment in 2021 by Märuer et al. [24] included some information that proved crucial in understanding how to model certain features in the Tamarin language.

4.1 Setting up Tamarin Prover

A lot of preparations were required for setting up Tamarin Prover on Windows 10 . Various different programs such as Windows Subsystem for Linux, Git Bash and Windows Powershell had to be installed. With the help of these programs, a virtual environment could be set up, where Homebrew, a packet management application, was installed. Using Homebrew, Tamarin Prover could be installed and set up in the virtual environment. From this environment test scripts for Tamarin can be run to formally verify the security protocols written in them. In order to write these scripts, Sublime Text 3 and Sublime Merge were also installed as an IDE and as a version control handler, respectively.

4.2 Modelling DRIP in Tamarin

Before translating DRIP into the Tamarin language, a clear step-by-step schema of the DRIP authentication process had to be created. The schema details exactly what is done and how it is done when an observer initiates a connection with a broadcasting drone. This method of constructing a clear schema to base our rules and lemmata off of has been used in previous formal verification work using Tamarin, such as the previously mentioned work of verifying an attachment procedure of LDACS in 2021 by Mäurer et al [24]. To construct such a schema, the latest DRIP drafts along with various other IETF meeting documentation were carefully studied.

To ease the process of formulating rules, the previously mentioned schedule was split up into two parts. One part being the HIP BEX schema seen previously in figure 3.1 and the other one detailing the registration process for a new UA. The schema serve as step-by-step guides of how DRIP is supposed to work. Having extremely detailed instructions to derive the rules from proved vital for the modelling process. As honest agents in the Dolev-Yao model do not retain information between rules, the slightest error or gap in knowledge could essentially invalidate the whole model. The two schemas do not cover every part of DRIP, but they were deemed as the most necessary parts and they served as a guide through modeling the more complicated parts of DRIP. Not every single DRIP feature could be modeled in Tamarin, and other parts were straightforward enough to understand that a detailed schema would have been unnecessary. The schema detailing the UA registration process can be seen in figure 4.1.

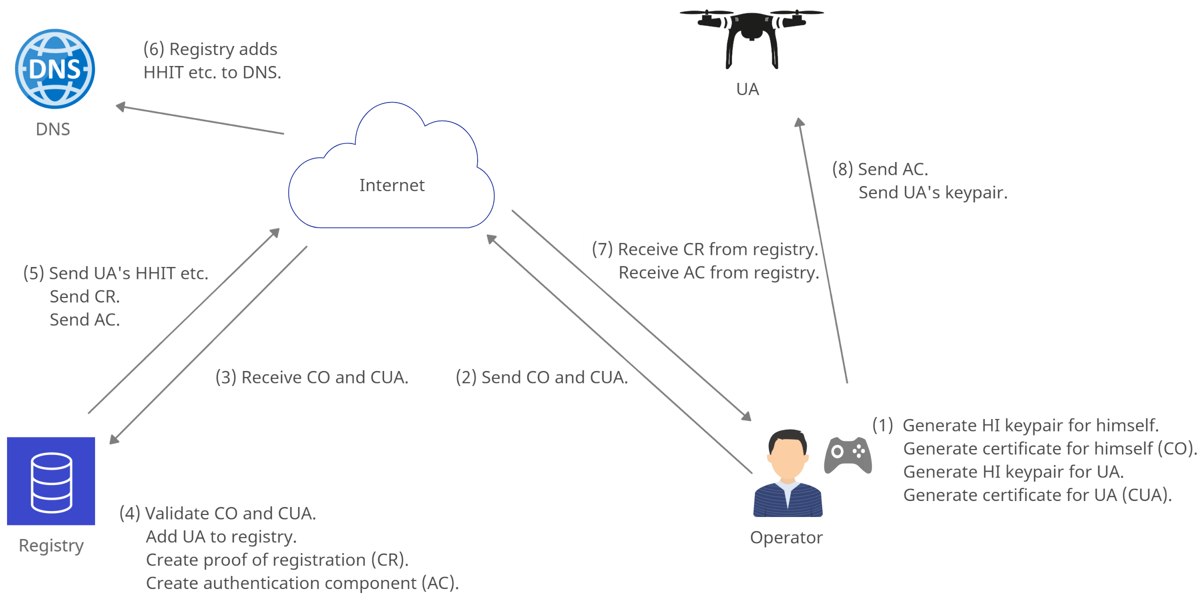

(6) Registry adds
HHIT etc. to DNS.
DNS
UA
Internet
(8) Send AC.
Send UA's keypair.
(7) Receive CR from registry.
Receive AC from registry.
(5) Send UA's HHIT etc.
Send CR.
Send AC.
(3) Receive CO and CUA.
(2) Send CO and CUA.
(1) Generate HI keypair for himself.
Generate certificate for himself (CO).
Generate HI keypair for UA.
Generate certificate for UA (CUA).
(4) Validate CO and CUA.
Add UA to registry.
Create proof of registration (CR).
Create authentication component (AC).
Registry
Operator

Figure 4.1: A detailed view of the registration process of a new UA to the registry.

First off, the owner/pilot of a drone needs to generate HI, HIT and private/public keys both for itself and the UA. From these keys, a certificate for both the operator and the UA are created. These certificates are then sent to a registry, where they are verified and subsequently stored for future lookup. The registry then creates a new certificate which it sends back to the operator, who then gives it to the unmanned aircraft system (UAS). The DET is at this point also registered to DNS to allow for lookups by observers receiving broadcasts. Lastly, the DET is broadcast from the UA to observers in the airspace. Said observers can also use the received DET to look up information about the drone in either a public or a private registry. To start, certificate generation at the operator and subsequent registration at a registry using HIP Base Exchange (BEX) was attempted. However, there are not many resources for certificates in Tamarin, resulting in the model registering the drones via the underlying public key used to generate the certificates instead.

Several more rules were written in to model the BEX in Tamarin. The safety of registration of a new drone was a big worry specified in the DRIP drafts, namely the leakage of a drone's private key from the manufacturer. This would mean that a drone could be imitated by someone else and that they could use the drone's private key to encrypt DRIP broadcasts and pose as the registered authentic drone. When modeling BEX as a Tamarin script, four rules were established. Two of these rules, called Initatior 1 and Initator 2, describe what the initiator does and what it sends to the responder. The other two rules, Responder 1 and Responder 2, describe the actions taken by the responder. Furthermore, a lemma was written to verify secrecy of session keys, this was done to specify a case where someone had obtained the session keys and could as such pose a threat to secure communications. Such an attack could lead to leakage of a UAs private key, which would in turn enable impersonation attacks on the UA.

Next, rules to specify the broadcasting of a DET were written. The rules let a drone broadcast its DET and observers receive the tag. Another rule was written to register a drone's keypair to a registry over a secure connection established via BEX. Since the tag is encrypted with the drone's private key, the observer would then receive the public key of the drone from the registry and subsequently be able to decrypt the identification message. To ensure that this worked, a lemma was created to test that an observer could successfully look up a drone's identity in this way and that no other entity could have received the drone's private key and thus impersonate it.

4.3 Formal verification

In order to formally verify the Tamarin scripts, one also has to write lemmata for Tamarin to test. The lemmata can represent certain types of attacks such as man-in-the-middle (MITM) attacks or desirable security properties of the protocol such as making sure that a shared secret is not leaked. The written lemmata focused on proving two main security properties, session key secrecy and DET authenticity. Proving these two properties also indirectly addresses MITM attacks and impersonation attacks via leakage of a drones private key. One more lemma was written which tested the executability of the rules. It is of utmost importance that all rules are run correctly. Upon execution, Tamarin collects all rules and turns them into a set of states. The execution traces that Tamarin generates are transition relations between these states. If one or more of the rules are malformed, the underlying algorithms and logic of Tamarin may choose to skip over some states while assessing the models security properties. The risk of this occurring is especially high when using using Tamarin's automatic proving mode, which is what was used to verify DRIP. The Tamarin team specifices ways to counteract this problem though, with one such method being the construction of an additional lemma to ensure that all rules are run to completion [12]. This method has previously been used to great success in the previously mentioned paper which formally verified LDACS in Tamarin [24] and is also the method used in this thesis.

An attempt was also made to write lemmata specifying the different forms of authentication as described by G. Lowe in [22]. This however proved difficult, and was ultimately non-successful. The different forms of authentication described by Lowe were attempted as a means to not only verify protection against some attacks, but also verify the absence of whole classes of attacks on the protocol. Aliveness, weak agreement and injective agreement were all attempted but could not be proven to be true.

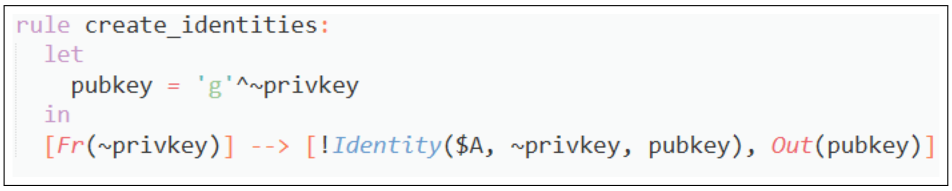
After all rules and lemmata were written, the model was run through the Tamarin Prover application by issuing terminal commands entered into the Ubuntu kernel hosted by Windows subsystem for Linux. This opens a local web server, where the user has the option of what lemmata should be attempted to prove. Using the auto-prove feature allows Tamarin to run the code and look for vulnerabilities in the architecture. If one is found, this results in Tamarin marking the code as red as well as showing a graph of where the weakness occurred. If a weakness is found, the lemma in question is classified as "false". If no weaknesses are found, the lemma is classified as "true" and the code is colored green instead. In such cases, a graph will still be generated but it will instead only show the flow between states of the model.

5 Results

The results presented in this section showcase the Tamarin scripts written, in the form of rules and lemmata for testing. The output from Tamarin Prover is also presented to graphically show what the tool gives when running a proof.

5.1 Tamarin model of the Drone Remote Identification Protocol

The protocol is modeled using Tamarin's symbolic rule-based language. Rules can produce facts and events based on the inputs and outputs. The events can later be used for examining the security via lemmata. As per the Dolev-Yao model, Tamarin can instantiate the state-facts infinitely many times, creating a situation with an arbitrary number of drones, receivers and registries. The first rule models the public key infrastructure and allows entities to be generated. See figure 5.1.



```
rule create_identities:
  let
    pubkey = 'g'^~privkey
  in
  [Fr(~privkey)] --> [!Identity($A, ~privkey, pubkey), Out(pubkey)]
```

Figure 5.1: Rule for creating identities and generating keypairs

At this point, the rules are divided into two parts. The first part focuses on establishing a secure connection with a HIP Base Exchange while the second part focuses on implementing DRIP-specific features such as broadcasting a remote ID, receiving a HHIT and communicating with registries. Only the DRIP specific sections of the Tamarin model will be discussed here. The HIP BEX, while important to the functionality of DRIP, has been formally verified before [33]. The full source code for the Tamarin model is accessible on GitHub [1].

The first DRIP specific rule, shown in figure 5.2, represents an unmanned aircraft broadcasting its entity tag in the form of a HHIT. We first generate a fresh ID and hash it to form our

[1]https://github.com/Jakaho/DRIP-tamarin, accessed 26-04-2022

HIT. To convert the HIT into a HHIT we asymmetrically encrypt it using the UA's private key. The hashed and encrypted ID is then broadcast over the network for everyone, including the adversary, to see.

```
rule UA_broadcast_DET:
let hit = h(~id)
hhit = aenc(hit, privkeyDrone)
in
[Fr(~id), !Identity($Drone, privkeyDrone, pubkeyDrone),
!Identity($Observer, privkeyObs, pubkeyObs)]
--[CreateUA($Drone, ~id)]->
[Out(<'broadcast', hhit>)]
```

Figure 5.2: Rule for generation and transmission of HHITs from drones.

The next logical continuation would be an observer receiving the broadcast HHIT, which must also be written in the form of a rule. When an observer receives a HHIT they cannot immediately gain any relevant knowledge on their own and must instead contact the registry to learn the drones public key. Only once the observer possesses both the drone's HHIT and public key can they decrypt the HHIT using the public key. See figure 5.3. Since the decryption process relies upon a registry sending the drones public key, this must also be represented in a rule, see figure 5.4.

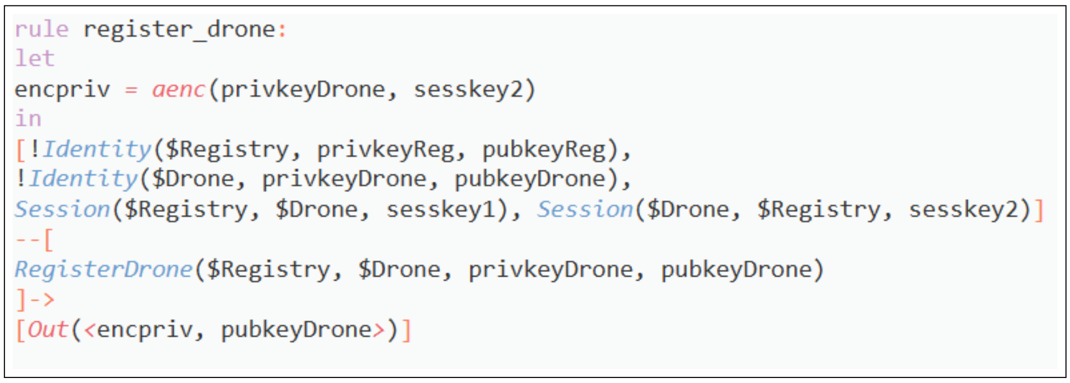
```
rule observer_receive_HHIT:
[!Identity($Observer, privkeyObs, pubkeyObs), In(<'broadcast', hhit>)]
--[
ObserverLookup($Observer, hhit),
Out(hhit),
In(pubkeyDrone)
]->
[DecryptHHIT($Observer, privkeyObs, pubkeyObs, adec(hhit, pubkeyDrone))]
```

Figure 5.3: Rule representing an observer receiving the broadcast HHIT and decrypting it using the drones public key acquired via a registry.

```
rule registry_send_pubkey:
[!Identity($Registry, privkeyDrone, pubkeyDrone), In(hhit)]
--[
RegistrySendPubkey($Registry, privkeyDrone, pubkeyDrone, hhit)
]->
[Out(pubkeyDrone)]
```

Figure 5.4: Rule for instantiating the registry and relaying a drones public key after receiving its HHIT.

Finally, there must also be a rule for the first time registration of a new drone to the registry. This is arguably the most dangerous part of the protocol that is prone to vulnerabilities, as the drones actual private key must be broadcast. Before the private key can be broadcast from the UA, there must exist an established secure connection between the UA and the registry. In terms of Tamarin rules, this is done by using previous session facts from the HIP base exchange. The modeled registration rule can be seen in figure 5.5.

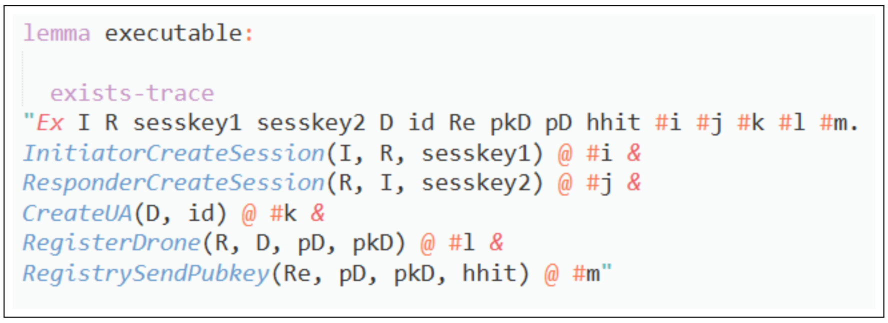


```
rule register_drone:
let
encpriv = aenc(privkeyDrone, sesskey2)
in
[!Identity($Registry, privkeyReg, pubkeyReg),
!Identity($Drone, privkeyDrone, pubkeyDrone),
Session($Registry, $Drone, sesskey1), Session($Drone, $Registry, sesskey2)]
--[
RegisterDrone($Registry, $Drone, privkeyDrone, pubkeyDrone)
]->
[Out(<encpriv, pubkeyDrone>)]
```

Figure 5.5: Rule for registering a drone to a registry in the form of its keypair.

5.2 Tamarin Lemmata

To prove that the protocol modeled with rules is secure, there must also be lemmata to test desired security properties. The first lemma ensures that the code can actually be run all the way through without skipping any steps. This is essentially an insurance that boosts the credibility of the other two lemmata, meaning that security is not only achieved through impossible scenarios. See figure 5.6.

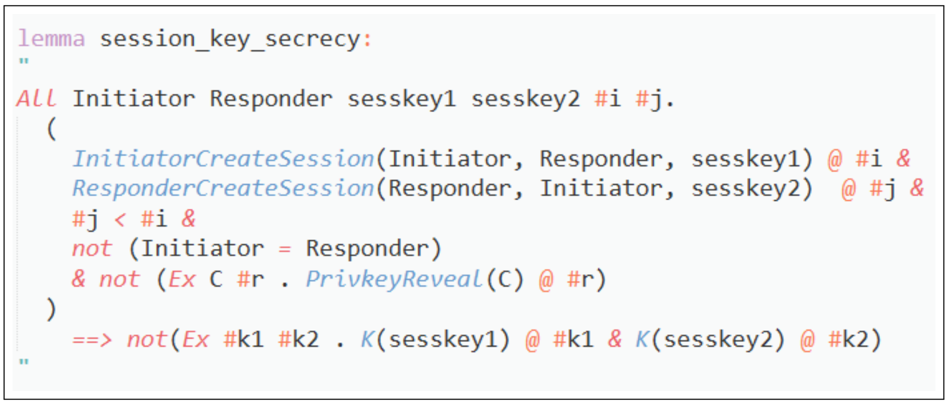


```
lemma executable:

  exists-trace
"Ex I R sesskey1 sesskey2 D id Re pkD pD hhit #i #j #k #l #m.
InitiatorCreateSession(I, R, sesskey1) @ #i &
ResponderCreateSession(R, I, sesskey2) @ #j &
CreateUA(D, id) @ #k &
RegisterDrone(R, D, pD, pkD) @ #l &
RegistrySendPubkey(Re, pD, pkD, hhit) @ #m"
```

Figure 5.6: Lemma for ensuring that the code can be run.

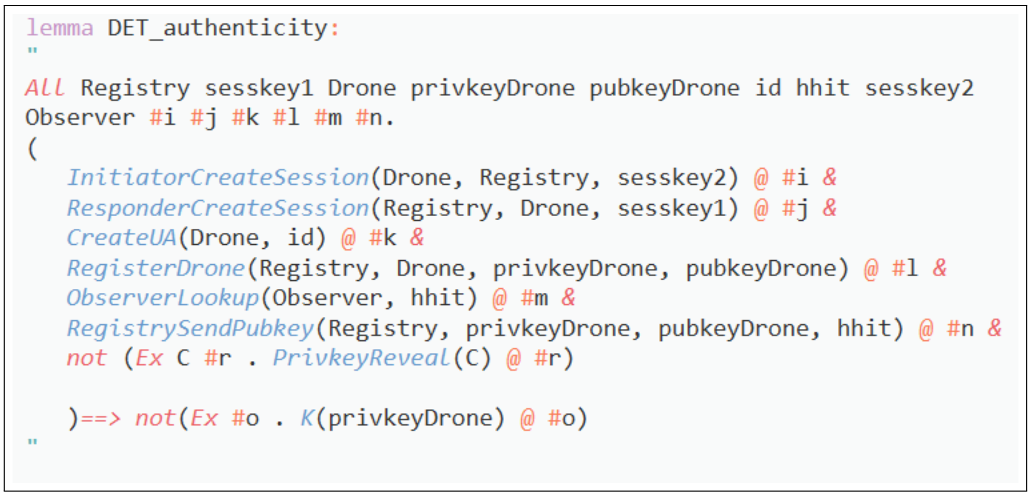
The first real property to examine is session key secrecy. This ensures that the session keys cannot be learned by anyone else when two parties establish a connection during the HIP Base exchange. In cryptography, secrecy is achieved when an adversary's knowledge of the contents of a message is the same both before and after inspecting the message. Simply put, it is encrypted well enough that the adversary cannot learn anything from the message even if it gets intercepted. There are several types of secrecy, such as computational secrecy which is achieved if the adversary cannot learn anything from the message within a practical time even with the fastest computer on earth. Perfect secrecy is when the adversary cannot learn anything even with infinite time and computing power [23]. Since Tamarin is based on Dolev-Yao models, the degree of secrecy is not examined further as any type of encryption is assumed to be unbreakable. Instead, secrecy is achieved if the message is encrypted at all and the adversary cannot gain access to the correct decryption key. If secrecy is not achieved, several attacks become possible, for example MITM attacks. The lemma that guarantees the secrecy of session keys can be seen in figure 5.7.



```
lemma session_key_secrecy:
"
All Initiator Responder sesskey1 sesskey2 #i #j.
  (
    InitiatorCreateSession(Initiator, Responder, sesskey1) @ #i &
    ResponderCreateSession(Responder, Initiator, sesskey2)  @ #j &
    #j < #i &
    not (Initiator = Responder)
    & not (Ex C #r . PrivkeyReveal(C) @ #r)
  )
    ==> not(Ex #k1 #k2 . K(sesskey1) @ #k1 & K(sesskey2) @ #k2)
"
```

Figure 5.7: Lemma for ensuring session key secrecy.

Lastly, a lemma for verifying the integrity of Drip Entity Tags is created. Maintaining the integrity proves that if a DET broadcast is received by an observer, they can be sure that the information received is in fact correct. The protocol is structured in a way where all lookups are done without the observer knowing the drone's public key must therefore contact the registry to learn it after receiving a DET broadcast. Being able to impersonate a registered drone can be done gaining access to the drone's private key, and this lemma aims to prove that the private key cannot be leaked from the broadcasts. The lemma uses facts from almost every rule, and tests basically the whole DRIP process from creation and registration of UA's to the broadcast and observation of HHITs. See figure 5.8.



```
lemma DET_authenticity:
"
All Registry sesskey1 Drone privkeyDrone pubkeyDrone id hhit sesskey2
Observer #i #j #k #l #m #n.
(
   InitiatorCreateSession(Drone, Registry, sesskey2) @ #i &
   ResponderCreateSession(Registry, Drone, sesskey1) @ #j &
   CreateUA(Drone, id) @ #k &
   RegisterDrone(Registry, Drone, privkeyDrone, pubkeyDrone) @ #l &
   ObserverLookup(Observer, hhit) @ #m &
   RegistrySendPubkey(Registry, privkeyDrone, pubkeyDrone, hhit) @ #n &
   not (Ex C #r . PrivkeyReveal(C) @ #r)

   )==> not(Ex #o . K(privkeyDrone) @ #o)
"
```

Figure 5.8: Lemma for ensuring that drones are not impersonated.

Together, these three lemmata allow Tamarin to examine the specified protocol in search of traces where execution of the code is not possible, where session key secrecy is not established or if drone lookup can result in impersonation attacks. When compiling the code with Tamarin, the user must choose what lemma to prove, one at a time. The user also has the option of manual or automatic proving. Tamarin Prover version 1.6.1 in the automatic proving mode is used for the evaluation of these three lemmata. All lemma results and constraint systems presented in this section are from the same compilation, ensuring that all lemmata are true simultaneously.

An overview of the results can be seen in table 5.1. The scope column in the table specifies whether the given property holds for at least one trace or for all traces. The Tamarin manual recommends that one lemma should be proven in exists-trace mode before other security properties are tested using all-traces, to make sure everything runs smoothly [32]. This is the role of the "executable"-lemma. All three lemmata were proven to hold, meaning that the symbolic model of DRIP also hold under the tested properties.

Table 5.1: Results of the Tamarin proof.

Lemma	Scope	Result
Executable	Exists-trace	Verified
Session_key_secrecy	All-traces	Verified
DET_authenticity	All-traces	Verified

5.3 Symbolic security proof

When running the code with Tamarin Prover in the automatic proving mode, a constraint system is generated as output. The constraint system visualizes how the protocol flows from state to state and highlights any potential weaknesses in the system. The constraint system showing the HIP Base Exchange can be seen in figure 5.9. The constraint system showing the broadcasting of DETs is found in figure 5.10. This is the result of what was generated by Tamarin when proving all three lemmata. As Tamarin was not able to generate a specific counterexample of when one of these lemmata do not hold, this can instead serve as a symbolic proof showing that the modeled protocol is secure. In the constraint system we can see what information is accessible from where, and deduce where the adversary may try to attack. The three lemmata used in this proof mostly focused on making sure that the private keys are not leaked. If Tamarin had managed to find a case where the adversary managed to gain access to a private key, parts of the constraint system would be colored red illustrating how the key was obtained.

The constraint system for the HIP Base exchange in figure 5.9 shows the process of an initiator setting up a secure connection to a responder by exchanging key information and negotiating session keys. There was also a similar constraint system generated that represents the responder counterpart of the HIP base exchange, but no figure is included as it is very similar to figure 5.9. The attempts of the adversary are also visible in the system, where the adversary's knowledge is represented by the K-fact in the figure below. The adversary accesses the initiator's public key but is not able to sign the message as he does not know the initiator's private key and cannot gain access to it. As the responder is not able to validate the signature sent by the adversary, the exchange between them is halted.

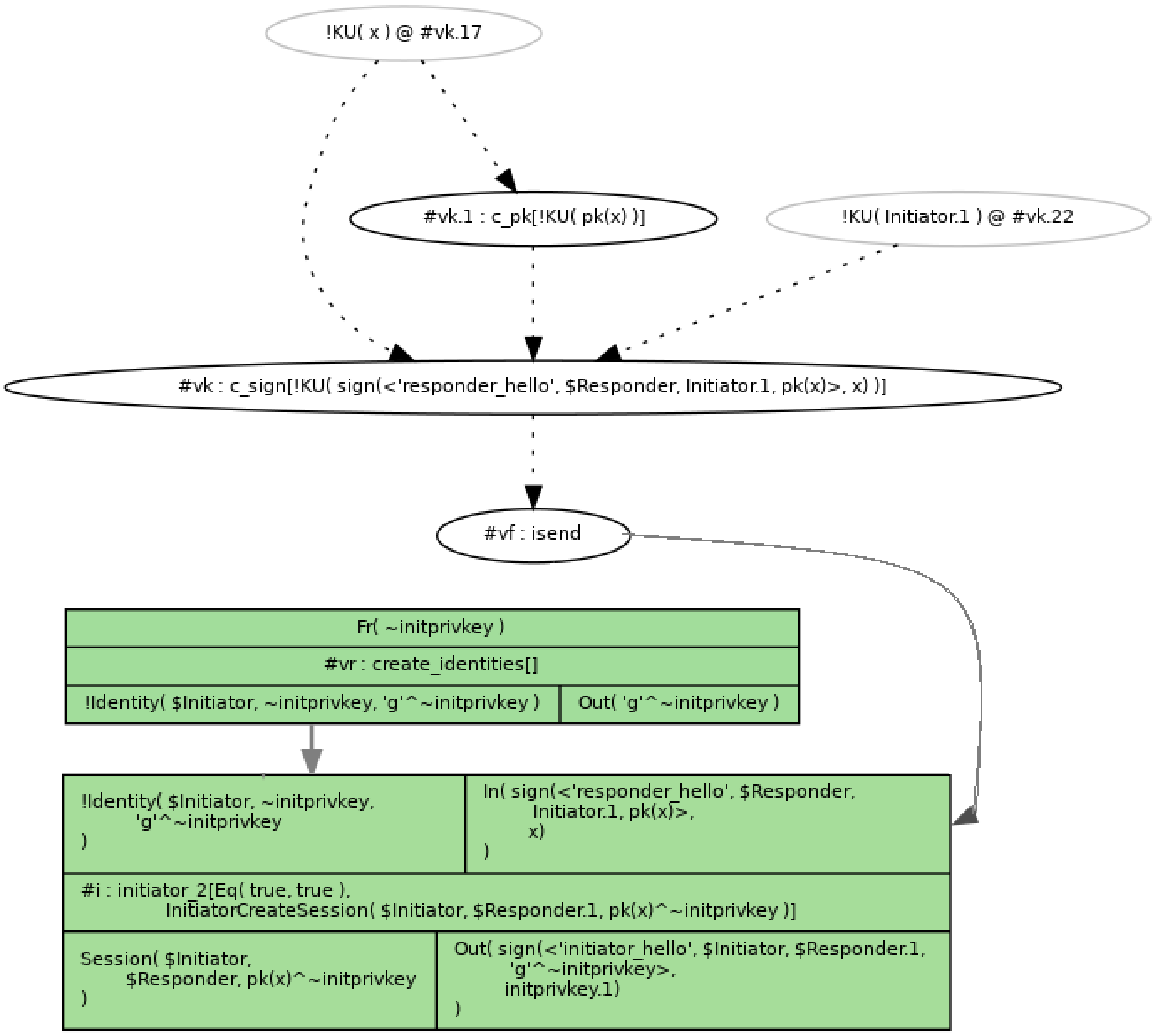

!KU(x) @ #vk.17
#vk.1 : c_pk[!KU(pk(x))]
!KU(Initiator.1) @ #vk.22
#vk : c_sign[!KU(sign(<'responder_hello', $Responder, Initiator.1, pk(x)>, x))]
#vf : isend
Fr(~initprivkey)
#vr : create_identities[]
!Identity($Initiator, ~initprivkey, 'g'^~initprivkey)
Out('g'^~initprivkey)
!Identity($Initiator, ~initprivkey, 'g'^~initprivkey)
In(sign(<'responder_hello', $Responder, Initiator.1, pk(x)>, x))
#i : initiator_2[Eq(true, true), InitiatorCreateSession($Initiator, $Responder.1, pk(x)^~initprivkey)]
Session($Initiator, $Responder, pk(x)^~initprivkey)
Out(sign(<'initiator_hello', $Initiator, $Responder.1, 'g'^~initprivkey>, initprivkey.1))

Figure 5.9: Tamarin Constraint System for HIP Base Exchange

The constraint system for DET broadcasting (figure 5.10) shows how two identities are created (the two green boxes). They look the same with the exception of the names, as these boxes represent two instances of the same rule. These two identities are then used for broadcasting a DET (shown in the yellow box). Which identity relates to drone or observer is shown by the two gray arrows pointing from each identity to their counterpart in the DET broadcast rule. As the broadcast of a drone's DET does not open up possibilities for any attacks no adversary attempts to access information are shown.

In the constraint system generated by Tamarin, there was also a third graph which represents the registration of drones and the subsequent verification of DETs. This is not included here, as the graph looked very similar to figure 5.9 since the HIP Base Exchange is utilized in the registration rule in order to guarantee a secure connection when sending a drone's private/public keypair to a registry.

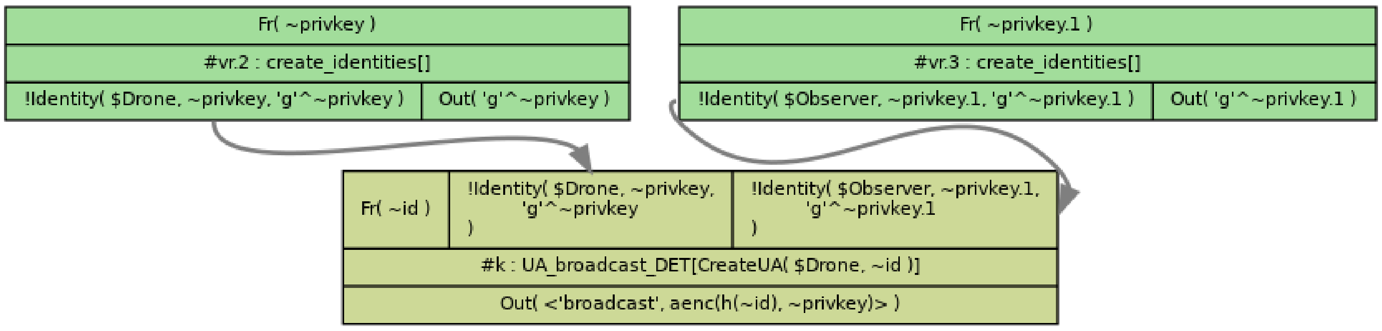

Fr(~privkey)
#vr.2 : create_identities[]
!Identity($Drone, ~privkey, 'g'^~privkey)
Out('g'^~privkey)
Fr(~privkey.1)
#vr.3 : create_identities[]
!Identity($Observer, ~privkey.1, 'g'^~privkey.1)
Out('g'^~privkey.1)
Fr(~id)
!Identity($Drone, ~privkey, 'g'^~privkey)
!Identity($Observer, ~privkey.1, 'g'^~privkey.1)
#k : UA_broadcast_DET[CreateUA($Drone, ~id)]
Out(<'broadcast', aenc(h(~id), ~privkey)>)

Figure 5.10: Tamarin Constraint System for DET broadcast

While Tamarin's constraint systems can be interesting, they are mostly useful if a lemma is disproven, meaning that the security property does not hold for one or more traces. If that is the case, one can examine the constraint system to get a better idea of how the adversary managed to exploit the system. A similar strategy has been used previously to detect and confirm attacks, for instance in 2021 the Tamarin team managed to prove the existence of a card brand mix-up attack involving MasterCard and Visa credit cards [6]. The attack was similar to MITM attack and revolved around tricking the payment terminal to bypass the PIN requirement of MasterCard cards during contactless payment. There was already suspicion as to how this attacked worked, but by using Tamarin they were able to confirm how the attack works and suggest proven countermeasures.

If a lemma is proven when compiling the code with Tamarin Prover, the generated constraint system may look very similar to if it is disproven. If the user is only interested in whether the security property holds up or not, there are easier ways to tell than by looking at the constraint system. For instance, in the Tamarin graphical UI, the entire code of the lemma will be colored green if it is secure, and red if it is not. The resulting constraint system can be used as some form of symbolic proof though, to show that the system is secure.

6 Discussion

In this section the implications of the results will be analysed and discussed. The results will also be evaluated in terms of trustworthiness. Furthermore, the method will be assessed and critiqued. We will also look at the work in a wider, ethical and societal context as well as present thoughts about future work in the area.

6.1 Results

The results presented in this paper are a start to complete formal verification of the Drone Remote Identification Protocol. The HIP Base Exchange, which much of the DRIP architecture relies on was tested and proven secure to the extent of the tests that were carried out. The lemmata were written to protect against leakage of session keys resulting in drone private keys being compromised. As these keys are used for encryption of DRIP Entity Tags, leakage of the private key would allow for these tags to be imitated, meaning that someone could falsely pose as a drone matching a legitimately registered drone. Through these lemmata, the integrity of the DET broadcast is stated to be upheld and thus most attacks where a bad entity imitates a drone should not be possible. However, this does not address replay attacks, where authentic messages are replayed in order to confuse an observer as to where the drone is actually located. Currently, the proposed fix to this problem is a short lifespan of the DET broadcast. The inclusion of a not-after timestamp set only a couple of minutes after the current time could make these replay messages trivial to detect. However, implementing this in Tamarin proved difficult and ultimately no such rules and lemmata were constructed. Even without a formal verification this should be secure, though, as a simple timestamp check should not open up the protocol to any new attacks. Furthermore, as long as a drone's private key is secure, no meddling in messages should be possible and thus all timestamps included in replay attack messages will be authentic and accurate.

The rules and lemmata resulting from the work are currently able to be executed from start to finish securely. However, as Tamarin does not verify the functionality this does not necessarily mean the code is one hundred percent correctly written. It proved hard to find anyone knowledgeable in both DRIP and Tamarin Prover, as such, proofreading of the Tamarin scripts was left up to the writers. As no apparent errors were found, the results are valid to the best of our knowledge. In Tamarin, the underlying cryptographic algorithms of the protocol such as the encryption of the HHIT or the signature algorithm is always assumed to be true. This implies that the computational soundness of these algorithms cannot be verified in Tamarin as they are assumed to hold, and serve as a baseline for the purposes of the proof.

The attempted authentication lemmata, based on Lowe's hierarchy, are not present in this thesis, but can be found in the authors' GitHub repository [1]. The result of these lemmata can be attributed to a multitude of factors. Seeing as the protocol should provide authentication through BEX, there is likely some form of error in the Tamarin script. What the error is proved diffucult to identify, seeing as the authors just recently started working with the tool. Ultimately, these authentication lemmata were left unproven due to a lack of time to identify whether the error lies in the lemmata or the rules they are based upon.

6.2 Method

Working with this thesis has been rather difficult. Tamarin is a new, state of the art tool and as such, there is not much material online detailing how to use it for your own specific needs. Getting a deeper understanding for both DRIP and Tamarin Prover required much reading, trial-and-error testing and discussion. Furthermore, while understanding the Tamarin language proved do-able, the proofs generated while autoproving include a lot of information about every step taken which turned out to be a lot harder to understand. An even deeper understanding of Tamarin Prover would have helped immensely, especially in the case of attempting to write new functions and equations in order to implement new algorithms. However, given the conditions in terms of former knowledge and available material, we feel that we have found relevant information which helped us reach a sufficient understanding of both DRIP and Tamarin Prover for the purpose of searching for weaknesses in the current drafts. When running into problems, seeing as we had no one experienced in Tamarin to consult, the only option was to find a suitable solution ourselves. For this reason, some compromises had to be made. One such compromise was the exclusion of the HIP Base Exchange puzzle, which exists to mitigate denial of service attacks. This was due to the fact that no suitable way to include the puzzle generation algorithms was found. Completely stopping denial of service attacks is also generally not possible and as such Tamarin is not equipped for testing against these types of vulnerabilities. Furthermore, instead of using EdDSA for signing, we utilized the existing signature functionality in Tamarin as implementing new algorithms proved difficult and was deemed unnecessary. Since the important part of EdDSA is to provide speed and security, a slower but just as secure signature algorithm should provide the same result in formal verification. Another important detail is that the lemmata were tested separately at first and then all together. This was done in order to rule out any edge-cases where secure procedures carried out in one would result in vulnerabilities in another. When running the Tamarin script, only the security is considered and not the effectiveness of the proof as done by Boeira and Asplund [7]. We do not measure execution time, memory usage or other resource consumption when conducting the proofs. This due to the fact that the DRIP model is relatively small scale and therefore the time it takes to run is inconsequential. While the Tamarin proof created in this thesis might be significantly smaller and test for less properties than the model created by Boeira and Asplund, it does still remain the size of other comparable Tamarin verification models, such as the one created for LDACS by Mäurer et al [24].

When searching for information about DRIP and Tamarin, some scientific papers were found. Most information that exists about DRIP is in form of internet drafts published by the Internet Engineering Task Force and papers written at Linköping University. These internet drafts are often updated, and as such might be made redundant after some time has passed. Therefore, only the latest drafts were studied, as well as recent papers relating to features still present in current drafts. Regarding Tamarin, the manual proved to be a great way to learn.

[1]https://github.com/Jakaho/DRIP-tamarin, accessed 12-05-2022

Several academic papers utilizing Tamarin Prover are listed on their website, however these did not provide much useful information for the purposes of this thesis. However, GitHub repositories containing Tamarin scripts, specifically from Mäurer et al. [24] [25], provided information about how to write certain features in the Tamarin language. YouTube tutorials by independent creators were also of great help when initially trying to understand the syntax of Tamarin language and how to generate keys for later usage.

6.3 The work in a wider context

Proving security of DRIP means that if the protocol becomes widely used in the future, concerns about security and integrity can be mitigated. Drone identification has received some mixed reactions, as drone pilots might not want to be "monitored" and feel that this is an intrusion of their privacy. When information about the pilot is also registered to the drone, even more questions about integrity might be raised. Others feel like registering a drone and its owner is not so different from registering a car with a license plate to an owner. Verified security of DRIP would mean that the concerns of personal information leaking from the registries could be addressed, as only authorized personnel should be able to see all information contained about a person. It would also mean a more secure and safe society, as no one should be able to frame someone else's drone for flying in a restricted area or over private property. If a drone constantly broadcast its position and owner, they would also not be able to be used for spying on neighbors without them knowing about it. Once again, this would result in people feeling safer when seeing a drone fly by. The question, however, is if all drones would need to abide by the protocol regardless of who the owner is. If government owned drones would require less information than personal ones, this could instill a debate about personal integrity versus national security.

6.4 Future work

The work presented in this thesis aims to start the process of formally verifying the safety of DRIP. While the presented tests do not completely cover all possible attacks on a network, they do however address the most detrimental ones. This being the leakage of a UA's private key to an adversary and compromise of session keys. Further work needs to be carried out in the area, and as such the tests and code created during our work could be of use when implementing more parts of DRIP or even new parts introduced in future drafts. Seeing as a lot of DRIP features rely on the Host Identity Protocol, the HIP Base Exchange created for the purposes of this paper might be of value in order to ease the process for future Tamarin testing. For future work in the field it is important to implement more features of DRIP and if possible, the specific signature algorithms such as EdDSA. More testing of additional scenarios would also be of relevance, to cover all possible types of attacks on the protocol. When writing the test scripts, it would also be beneficial if knowledgeable individuals both about DRIP and Tamarin Prover were consulted. This would mean that the task force working on DRIP could verify that the models are correct and the Tamarin experts would make sure that the code was correctly written.

As previously stated, a complete formal verification of the finished version of DRIP would give credibility to the protocol and showcase just how much of an improvement upon former standards DRIP would supply. In order to achieve this, the current issues in the Tamarin scripts would need to be identified and handled. Further implementation of different types of authentication should rule out types of attacks not proven secure as of this thesis.

7 Conclusion

This thesis set out to improve the trustworthiness of the underlying security structure of DRIP via a formal verification of the protocol using Tamarin Prover. Formally verifying a security protocol in the development stages allows for possible vulnerabilities to be detected and rectified before they can do any actual damage. The HIP Base exchange and key features of DRIP were modeled, including drone broadcast and an observer receiving the broadcast and performing a lookup. The security properties examined were session key secrecy and authenticity of the broadcasts. These properties were proven to be true, meaning attacks such as man-in-the-middle attacks and impersonation attacks via a drones leaked private key should not be possible. The results indicate that the current proposed version of DRIP is secure to the extent mentioned above. The results are not entirely conclusive though, in the sense that not all DRIP functionality was added to the Tamarin model and only a few security properties were tested. There may also have been slight errors in the Tamarin model due to the authors' inexperience with DRIP and Tamarin Prover. Certain parts of DRIP were left out of the model intentionally due to them not having relevance to the proofs, and some parts could not be accurately modeled due to the limited resources and libraries in Tamarin. Future work may include confirming the correctness of our model, while also including more details to it. Correctly modeling additional parts of DRIP to get a more complete verification of the full DRIP process could also be useful. Specifically, functionality relating to registries, certificates and first-time registration of drones are currently lacking and could be added to the model. Writing lemmata for other types of security properties may also be of interest to extend the verification further. Possible security properties to examine are injective agreement, recentness, as well as others from Lowe's taxonomy [22]. Properties related to post-compromise security could also be relevant [11]. As Tamarin is mostly suited for key exchanges and similar processes, it currently has quite limited support for some of parts of DRIP. If further verification is needed, some of those parts may have to be examined using other tools.

Bibliography

[1] Moskowitz et al. *DRIP Entity Tag (DET) for Unmanned Aircraft System Remote Identification (UAS RID)*. Internet-Draft draft-ietf-drip-rid-17. IETF, Mar. 2022. URL: https://datatracker.ietf.org/doc/draft-ietf-drip-rid/.

[2] Harald T. Alvestrand. *A Mission Statement for the IETF*. RFC 3935. Oct. 2004. DOI: 10.17487/RFC3935. URL: https://www.rfc-editor.org/info/rfc3935.

[3] Jari Arkko and Börje Ohlman. "Host identity indirection infrastructure (Hi3)". In: *Proceedings of the 2nd Swedish National Computer Networking Workshop SNCNW*. Vol. 4. 2004, pp. 1–4.

[4] ASTM. *ASTM F3411-19, Standard Specification for Remote ID and Tracking*. standard. ASTM, Feb. 2020, pp. 1–67. URL: https://www.astm.org/f3411-19.html.

[5] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. "Formal verification of security protocol implementations: a survey". In: *Formal Aspects of Computing* 26 (Jan. 2014), pp. 99–123. DOI: https://doi.org/10.1007/s00165-012-0269-9.

[6] David Basin, Ralf Sasse, and Jorge Toro-Pozo. "Card Brand Mixup Attack: Bypassing the PIN in non-Visa Cards by Using Them for Visa Transactions". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 179–194. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/basin.

[7] Felipe Boeira and Mikael Asplund. "Exploiting Partial Order of Keys to Verify Security of a Vehicular Group Protocol". In: *ArXiv* abs/2105.02664 (2021).

[8] Christian Bunse and Sebastian Plotz. "Security Analysis of Drone Communication Protocols". In: *Engineering Secure Software and Systems*. Ed. by Mathias Payer, Awais Rashid, and Jose M. Such. Cham: Springer International Publishing, 2018, pp. 96–107. ISBN: 978-3-319-94496-8.

[9] 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.

[10] Stuart W. Card, Adam Wiethuechter, Robert Moskowitz, Shuai Zhao, and Andrei Gurtov. *Drone Remote Identification Protocol (DRIP) Architecture*. Internet-Draft draft-ietf-drip-arch-22. Work in Progress. Internet Engineering Task Force, Mar. 2022. 27 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-drip-arch-22.

[11] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. "On Post-compromise Security". In: *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 2016, pp. 164–178. DOI: 10.1109/CSF.2016.19.

[12] Véronique Cortier, Stéphanie Delaune, and Jannik Dreier. "Automatic generation of sources lemmas in Tamarin: towards automatic proofs of security protocols". In: *ESORICS 2020 - 25th European Symposium on Research in Computer Security*. Vol. 12309. Lecture Notes in Computer Science. Guilford, United Kingdom, Sept. 2020, pp. 3–22. DOI: 10.1007/978-3-030-59013-0_1. URL: https://hal.archives-ouvertes.fr/hal-02903620.

[13] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1773–1788. ISBN: 9781450349468. DOI: 10.1145/3133956.3134063. URL: https://doi.org/10.1145/3133956.3134063.

[14] Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". In: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)* (1981), pp. 350–357.

[15] Sandra Dünki. "Modelling and Analysis of Web Applications in Tamarin". Master's thesis. Swiss Federal Institute of Technology Zurich, Oct. 1, 2019. URL: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/ma-19-duenki-webmodel.pdf.

[16] Google. *Wi-Fi Aware overview*. Oct. 27, 2021. URL: https://developer.android.com/guide/topics/connectivity/wifi-aware (visited on 07/03/2022).

[17] Bluetooth Core Specification Working Group. *Bluetooth Core Specification*. July 13, 2021. URL: https://www.bluetooth.com/specifications/specs/core-specification-5-3/ (visited on 07/03/2022).

[18] Yousef Hashem, Elmedin Zildzic, and Andrei Gurtov. "Secure Drone Identification with Hyperledger Iroha". In: *Proceedings of the 11th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 11–18. ISBN: 9781450390811. URL: https://doi.org/10.1145/3479243.3487305.

[19] Simon Josefsson and Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032.

[20] Suleman Khan, Markus Loborg, Robin Lidekrans, and Lukas Rajala. *TDDE21 - DRIP Project Report*. Dec. 2021. URL: https://www.ida.liu.se/~TDDE21/info/tdde21_droneid_2021.pdf.

[21] Julien Laganier and Francis Dupont. *An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2)*. RFC 7343. Sept. 2014. DOI: 10.17487/RFC7343. URL: https://www.rfc-editor.org/info/rfc7343.

[22] G. Lowe. "A hierarchy of authentication specifications". In: *Proceedings 10th Computer Security Foundations Workshop*. 1997, pp. 31–43. DOI: 10.1109/CSFW.1997.596782.

[23] Ueli M. Maurer. "Conditionally-Perfect Secrecy and a Provably-Secure Randomized Cipher". In: *J. Cryptol.* 5.1 (Jan. 1992), pp. 53–66. ISSN: 0933-2790.

[24] Nils Mäurer, Christoph Gentsch, Thomas Gräupl, and Corinna Schmitt. "Formal Security Verification of the Station-to-Station based Cell-attachment Procedure of LDACS". In: *SECRYPT*. 2021.

[25] Nils Mäurer, Thomas Gräupl, Christoph Gentsch, Tobias Guggemos, Marcel Tiepelt, Corinna Schmitt, and Gabi Dreo Rodosek. "A Secure Cell-Attachment Procedure of LDACS". In: *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. 2021, pp. 113–122. DOI: 10.1109/EuroSPW54576.2021.00019.

[26] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The Tamarin Prover for the Symbolic Analysis of Security Protocols". In: *Lecture Notes in Computer Science, vol 8044*. CAV 2013. Springer, Berlin, Heidelberg, 2013, pp. 696–701. ISBN: 978-3-642-39799-8. URL: https://doi.org/10.1145/3133956.3134063.

[27] Robert Moskowitz, Tobias Heer, Petri Jokela, and Thomas R. Henderson. *Host Identity Protocol Version 2 (HIPv2)*. RFC 7401. Apr. 2015. DOI: 10.17487/RFC7401. URL: https://www.rfc-editor.org/info/rfc7401.

[28] Federal Register. *Remote Identification of Unmanned Aircraft*. Jan. 2021. URL: https://www.federalregister.gov/documents/2021/01/15/2020-28948/remote-identification-of-unmanned-aircraft.

[29] Juliano Rizzo and Thai Duong. "Practical padding oracle attacks". In: *4th USENIX Workshop on Offensive Technologies (WOOT 10)*. 2010.

[30] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties". In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012, pp. 78–94. DOI: 10.1109/CSF.2012.25.

[31] Vincent Stettler. "Formally analysing the TLS 1.3 proposal". Bachelor's thesis. Swiss Federal Institute of Technology Zurich, July 3, 2016. URL: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/TLS-1.3_thesis_vincent_stettler.pdf.

[32] The Tamarin Team. *Tamarin-Prover manual*. The Tamarin Team. Sept. 1, 2021. 107 pp. URL: https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf.

[33] Hannes Tschofenig, Andrei Gurtov, Jukka Ylitalo, Aarthi Nagarajan, and Murugaraj Shanmugam. "Traversing Middleboxes with the Host Identity Protocol". In: July 2005, pp. 17–28. ISBN: 978-3-540-26547-4. DOI: 10.1007/11506157_2.

[34] European Union. *COMMISSION DELEGATED REGULATION (EU) 2019/945 of 12 March 2019 on unmanned aircraft systems and on third-country operators of unmanned aircraft systems*. Aug. 2020. URL: http://data.europa.eu/eli/reg_del/2019/945/2020-08-09.

[35] European Union. *COMMISSION IMPLEMENTING REGULATION (EU) 2021/664 of 22 April 2021 on a regulatory framework for the U-space*. Apr. 2021. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32021R0664&from=SV.