Formally Verifying the Newest Versions of the GNSS-centric TESLA Protocol

Short Paper

Ioana Boureanu
University of Surrey
i.boureanu@surrey.ac.uk

Stephan Wesemeyer
University of Surrey
s.wesemeyer@surrey.ac.uk

***Abstract*—Global Navigation Satellite Systems (GNSS) are critical for infrastructure like energy, telecommunications, and transportation, making their accuracy vital. To enhance security especially against location spoofing, in 2024, the Galileo GNSS system adopted the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) protocol, for Navigation Message Authentication (NMA). However, past and present TESLA versions have lacked formal verification due to challenges in modelling their streaming and timing mechanisms. Given the importance of formal verification in uncovering protocol flaws, this work addresses that gap by formally modelling and verifying the latest TESLA protocol used in Galileo; we verify Galileo's TESLA protocol in the well-known `Tamarin` prover. We discuss our findings and, since this is work-in-progress, we contextualise them in terms of next steps for us, as well as for future Navigation Message Authentication protocols inside GNSS systems.**

I. Introduction

Over the past 25 years, *global navigation satellite systems (GNSS)* have become essential components of various critical infrastructures, such as energy distribution, telecommunications, financial services, and transportation. In recent times, state-level adversaries have increasingly underscored the importance of maintaining GNSS accuracy [1]. This highlights the need for *Navigation Message Authentication (NMA)*, that is authentication of navigation data broadcast by GNSS.

Indeed, in 2024, the Galileo system, that is the European service for GNSS, incorporated a protocol for NMA [2], [3]. This protocol is called *Timed Efficient Stream Loss-Tolerant Authentication (TESLA)* and its original version [4] has been adapted since 2016 [5], specifically to provide enhanced NMA capabilities for the Galileo I/NAV E1-B Open Service signal, such as to maximise both availability and robustness.

Simply put, in TESLA, one or several *satellites* are sending a series of timed (location) messages to a ground verifier or *user*. The authentication of the messages and the sender occurs based on a certified initial key sent by the satellite and subsequent keys generated from one another, in a sequence. These interdependent keys are computed using one-way functions and are released to the user with some delay, to avoid pre-computation, impersonation and timing attacks.

One important matter is that these new versions of TESLA have *not* been formally verified with computer-aided provers or tools for systems or security analysis. Even older versions of TESLA, from the early 2000, have *not* been formally verified [6]. To a great extent, this is because the protocol uses a continuous "stream" of authentication messages, as well as timing, with authentication occurring if multiple checks pass over an interval of time over a subset of messages in that stream. These aspects of streaming and timing are hard to model in computer-aided verification tools, and –in general– ingenious approximations would be needed therein, to carry out the modelling of the protocol and the verification of its correctness, let alone security. However, it is arguably important and necessary to formally model and verify security protocols and systems. Indeed, history and experience shows that formal verification uncovered important and subtle flaws in security protocols of all kinds (e.g., in versions of TLS [7], [8], in WPA2 [9], in payments [10]), sometimes decades after they were proposed; IETF now asks for formal verification [11]).

In this vein, in this work, we formally model and verify the newest version of the GNSS-centric TESLA protocol, implemented as for early 2024 in Galileo, the European GNSS.

II. Background

A. The Galileo-adopted version of TESLA

We now explain the new version of the TESLA protocol [5], as adopted in Galileo [2], [3].

Let S be a sender/satellite, R be a receiver/user, n, d be some parameters. The protocol is given on Figure 1.

Timestamped TESLA Keys. The protocol starts with S generating a random *TESLA key* K_n. Then, S iteratively generates the list $K_{n-1}, K_{n-2}, \ldots, K_i, \ldots, K_0$ of *TESLA keys*, where $K_i := F^{n-i}(K_n, GST_i)$, F is a one-way[1] function, F^j denotes applying F j times, and GST_i is the GNSS time, as held by S, at the time of the generation. First, the sender sends K_0 signed[2] to the receiver. Then, S uses the keys

[1] In [5], F is taken to be $trunc(hash(...))$, that is a truncation of a hash. The GST-based timestamping is used inside F to protect against pre-computation attacks. In Galileo, SHA-256, SHA3-256 are used as hashes.

[2] In Galileo, ECDSA P-256/SHA-256 or P-521/SHA-512 are used [2], [3].

Workshop on Security of Space and Satellite Systems (SpaceSec) 2025
24 February 2025, San Diego, CA, USA
ISBN 979-8-9919276-1-1
https://dx.doi.org/10.14722/spacesec.2025.23009
www.ndss-symposium.org

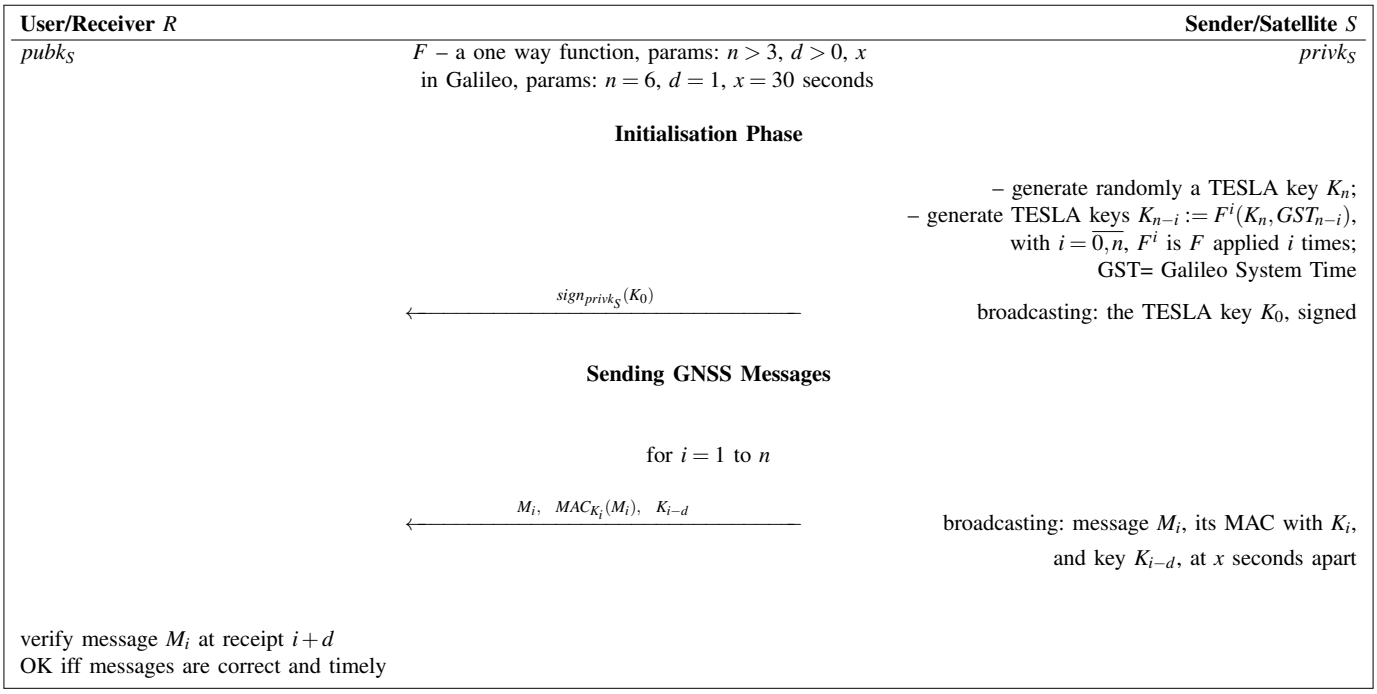

Figure 1. The TESLA protocol as adapted for Galileo

in the order $K_1,\ldots,K_{n-2},K_{n-1},K_n$ to MAC[3] location/GNSS messages $M_1,\ldots,M_n$.

Timed, Chained Location Messages. As shown in Figure 1, S sends a *chain* of GNSS messages $M_1,\ldots,M_n$ (and their MAC), at regular intervals: e.g., M_i is sent x seconds before $MAC_{K_i}(M_i)$, which is sent x seconds before K_{i-d}, which is sent x seconds before M_{i+1}.

Receiver's Verification. At the ith message-receipt within a chain, the receiver R gets message M_i , but cannot verify it against the sender's data, as –by then– the receiver R has only obtained keys $K_0,\ldots,K_{i-d}$ from the sender. It is at receipt $i+d$ that the receiver can verify M_i, as it is then that the key K_i, which MAC-ed M_i, is received. Note that, due to the one-wayness of F, releasing the K_i does not reveal any information of the "later-to-be-released" keys K_{i+j}, for any $i+j>i$ (i.e., $K_i = F^j(K_{i+j}, GST_i)$). In Galileo's current implementation, d=1.

TESLA's Requirements. To be able to verify the messages, the receiver and the sender have to have *synchronised clocks*, and the sender needs to send timing data alongside the GNSS payloads. In this way, the receiver can estimate time intervals during which the keys K_i would have been generated, and time intervals when messages M_i would have legitimately left the sender's side. The former also allows the receiver to compute $K_i := F^{n-i}(K_n, e^j - GST_i)$ for some estimations $e^j - GST_i$ of the timestamps GST_i, and thus check the messages and authenticate them and the sender.

Overall, this also allows the receiver to class the sender and their messages, as valid and timely with some probability. Tuning the false rejection rate clearly impacts (in)security and efficiency, as to amplify the receiver's certainty longer chains and tighter parameters would be needed. Concretely, the process depends on sender-receiver time synchronisation, parameters n, d and the delay of x seconds between the frames M_i, $MAC_{K_i}(M_i)$, K_{i-d}, etc. In Galileo trials, n was taken to be 6, d to be 1, x to be 30 seconds [2], [3].

[3]In Galileo, HMAC-SHA-256 and CMAC-AES are used as MACs [2], [3].

B. Symbolic Verification & The `Tamarin` Protocol-Prover

Since the community of space security may not necessarily be familiar with symbolic verification, we introduce it here.

Symbolic Verification. In *symbolic or Dolev-Yao analysis* [12], [13], [14], protocols/systems are encoded via "possibilistic" (rather than probabilistic) measures, cryptography is assumed to be perfect/correct [15], yet it allows for a computationally unbounded *Dolev-Yao (DY)* [15] attacker which can hijack all protocol sessions and communication, and corrupt all parties modulo the perfect-cryptography assumption, and a set of abstract, algebraic rules that encapsulate these powers [15]. This makes symbolic analysis amenable to mechanisation into (often automatisable) computer-aided verifiers, which have been fruitfully used in the analysis of cryptographic systems [7], [8], [9], [10].

`Tamarin` Models. `Tamarin` [13] is a Dolev-Yao [15] protocol-verifier, which supports the analysis of an unbounded number of protocol sessions. `Tamarin` models are transition systems (TS), whose transitions are modelled via *rules* containing logical predicates (called *facts*) over user-defined protocol variables.

`Tamarin` Proofs & Oracles. In `Tamarin`, one writes *lemmas* in a fragment of first-order logic over some of these

facts, in order to inspect (all or one) possible executions/*traces* of the transition system of the protocol model.

The trace-inspection via said lemmas constitutes the *proof*, which is a backward search over the space of all possible Dolev-Yao executions/traces of the model. `Tamarin` supports various heuristics to cover this search-space. These heuristics determine which Dolev-Yao deduction or "protocol-executing" rules should be prioritised during the proof search. Users can also create bespoke search heuristics or *"oracles"* [16], to overrule `Tamarin`'s default heuristics in case of specific proofs.

III. Modelling Galileo's TESLA in Tamarin

Note: All our `Tamarin` files are at http://people.itcarlson.com/ioana/tesla/.

We now explain how we modelled the Galileo-adopted TESLA in `Tamarin`, and what the challenges around this are.

Older versions of the TESLA protocol [4] have been simplistically modelled in `Tamarin` in around 2012 [17], [18], [19]; not only are these for an older version of TESLA, but they are much simplified [17], and some are labelled as not working [18]. In fairness, these simple models were not aiming to verify TESLA per se, but were mainly created to illustrate how loops can be constructed in `Tamarin` (i.e., the fact that the satellite keeps sending messages continuously), and how loops posed certain modelling challenges in the tool.

A. Challenges in Modelling & Verifying TESLA

Due to the limitations of existing Dolev-Yao/symbolic provers, there are three main challenges around modelling and verifying TESLA in `Tamarin`, and, in fact, in any Dolev-Yao/symbolic protocol verifier:

1) modelling of time, that is: (a) the timestamps inside the functions F; (b) the fact that navigation messages are sent at regular time intervals; (c) the fact that the receiver must to do time-based verification and discern between "old' and "time-valid" messages;
2) modelling (a) the length of the TESLA chain of GNSS messages; (b) the (continuous) repetition of these chains;
3) dealing with exponential growth in the size of the models, increasing with: the length of chains and their repetition for each satellite, and the number of satellites taking part in a TESLA execution sending messages to a receiver; this affects the tractability of verifying larger models.

B. Our Concrete Modelling & Verification of TESLA

We implemented in `Tamarin` the sender and receiver behaviours within the TESLA protocol, just as described in Figure 1 and in Section 2, allowing for a full Dolev-Yao attacker that exists in `Tamarin` (i.e., injecting, replaying, etc.). The most important modelling and verification aspects lie in how we overcame the challenges above, as described below.

Challenge 1: Modelling Time. Tamarin does not model real/continuous time, and one can only simulate time via discrete steps. One therefore needs to be ingenious about this and there is no standard approach to this.

We model time/acceptability intervals not of a fixed length (e.g., 30 seconds), but dictated by the "discretely-timestamped" actions of the satellite, as explained next.

We encode timing (constraints) using a "$timestamp" variable, and all the `Tamarin` rules modelling time-sensitive TESLA exchanges have a fact, $NewTimestamp(\$timestamp)$; in other words, whenever that rule executes it is "tagged" with a fresh value; this emulates, e.g., the time it was when the sender/satellite sent something and it placed this time in the meta-data that comes with the navigation payloads (i.e., I/NAV clock and status in ADKD tags [2], [3]).

We also created a global restriction that enforces that each created timestamp is unique.

```
rule Receiver_Valid_Message:

    [
          !VerifiedKey($R, $S, %chain, key)
        , Receiver_VerifyContent($R, $S, %chain, content, signature, $timestamp)
    ]
    --[
          _restrict(IsTrue(checkMac(signature,content, key)))
        , ContentIsValid($R, $S, %chain, content)
        , _restrict(WithinInterval($S, %chain, key, $timestamp))
    ]->
    [
    ]
```

Figure 2. Implementing Time-restrictions in Message Checks

We use our timestamp artefact for the receiver to verify that the appropriate time intervals are adhered to, e.g., keys are only valid within a given chain and within their allowed time intervals. Figure 2 shows our Tamarin rule called *Receiver_Valid_Message*: it implements messages' verification by the receiver, and includes a restriction using our predicate *WithinInterval*, utilising our timestamp artefacts. In Figure 3, we show this predicate. It validates that a time-value $\$timestamp$ is indeed within its allowed interval of a given session. Concretely, the predicate checks that when the user receives a message signed with key_i at $\$timstamp$, then $\$timstamp$ must lie within the interval of when the satellite sent its message signed with key_i and its subsequent message signed with key_j ($key_i = F(key_j, \ldots)$).

```
predicates:
  IsTrue(x) <=> (x = true),
  IsFalse(x) <=> (not(x = true)),
  IsSame(x,y) <=> (x = y),
  WithinInterval(S, %chain, key1, timestamp) <=>
  (Ex S %chain key2 ts1 ts2 ts3 #t01 #t02
    .
    UsedKey(S, %chain, key1, ts1) @ t01 // msg was sent
    &
    UsedKey(S, %chain, key2, ts2) @ t02 // subsequent msg was sent
    &
    key1=f(<key2, ts3>) // the relation between keys during an interval
    &
    //message was received after it was sent
    (All #i #j . NewTimestamp(ts1) @ i & NewTimestamp(timestamp) @ j ==> #i < #j)
    &
    //message was received before the next one was sent
    (All #i #j . NewTimestamp(timestamp) @ i & NewTimestamp(ts2) @ j ==> #i < #j)
  ),
  EarlierThan(ts1,ts2) <=>
  (All #i #j .
      NewTimestamp(ts1) @ i
    & NewTimestamp(ts2) @ j
```

Figure 3. Enforcing Acceptable Time Intervals

Challenge 2: Modelling Chains. Tamarin 1.8 was the first version to introduce natural numbers; we use this feature in our modelling, to restrict the length n of a chain (i.e., the

number of TESLA keys inside the model) and to count the number of chains per satellite.

To simplify the management of TESLA chains, we also modelled the case where the TESLA parameter d is equal to n: i.e., the MAC-ing keys are released as late as possible, that is all at the end. This allows for most attempts of timing and integrity attacks, so it is a security-safe modelling. The model can be easily adapted to release one MAC-ing key at a time.

Challenge 3: Tractability Challenges. Full proof for protocols that rely on a constant stream of messages being generated, as is the case with TESLA, could be done in the absolute via induction, but this is not fully supported in `Tamarin` for models that use natural numbers, like ours[4]. So, constraints or bounds need to be put in place to do our proofs.

To make verification tractable, we place restrictions in our generic model:
(i) the length n of message chains is restricted to 5;
(ii) the number of satellites – to 2;
(iii) the number of repeated chains per satellite – to 4.

Restriction (i) is not unrealistic, since the tested length of the chain in Galileo was 6 [2], [3]. We show how this is achieved in `Tamarin` in Figure 4; note that this is easily adjustable to other values. The last two restrictions are not stumbling blocks w.r.t. proofs, either: to prove most correctness and security properties coming from interleaving chains, a value of 2 chains per satellite (one honest and one hijacked, in part or in total) would suffice; similarly, one honest and one dishonest/to-be-impersonated satellite suffice to show attacks.

```
// limit the number of generated keys
restriction key_gen_to_n:
"All %x #t01 .
  LessThanN(%x) @ t01  //this predicate is used in our rules
  ==>
  %x << %1 %+ %1 %+ %1 %+ %1 %+ %1 %+ %1 //this means max 5 key for one chain
"
```

Figure 4. Restriction for tractable verification

As discussed in Section II-B, we carried out our proofs first by hand and, based on the strategy found, we built a `Tamarin` oracle that automates the proving and makes it more efficient: i.e., it tells `Tamarin` how to explore the proof space. For instance, the snippet in Figure 5, partly shows which `Tamarin` transitions, be it Dolev-Yao attacker actions (starting the "KU(") or specific protocol steps (starting with something other than "KU(")), to apply first, in order to *efficiently* prove a lemma on the receiver being able to detect delays.

IV. Verification

a) ***Properties Proven****:* We encoded a total of 19 lemmas checking correctness and security. Of these, 11 lemmas check the correctness or the soundness of our model. For instance,

[4](a) If we could write lemmas in `Tamarin` quantifying over natural numbers (e.g., for all n number of satellites), then (b) a non-expert in `Tamarin` could image we could prove – by induction – lemmas for arbitrary-size models. But, one cannot do (a) in `Tamarin`, and even if one could do (a), then one cannot do (b), that is induction proofs over natural numbers in `Tamarin` (i.e., induction in `Tamarin` exists, but over "re-appearing" facts, not natural numbers).

```
elif "detecting_delays" in lemma:
    print ("applying oracle to "+lemma)
    if 'Receiver_VerifyContent' in line or 'SatelliteMessage' in line :
        l2.append(num)
    elif 'StartSeed( $S, %1' in line or 'SendKey' in line:
        l3.append(num)
    elif 'KeyN(' in line:
        l4.append(num)
    elif 'KU( sign(<' in line or 'KU( MAC' in line:
```

Figure 5. Example Oracle

we show that a user can receive valid messages, early, as well as late. Then, security-wise, via 8 lemmas, we show:
– that an Dolev-Yao attacker can inject messages in various ways into the system, including within chains on which the receiver gets valid messages, and that would not pass the receiver's checks;
– integrity of the messages: the receiver will reject invalid messages, which lack integrity;
– timeliness of the messages: the receiver will reject messages (even when they have cryptographic integrity) if they were received outside of their validity time-interval;
– authentication of satellites: any valid, accepted message must have been sent by a satellite (and not the attacker);
– authentication of satellites and messages, or no hijacking of message chains: i.e., if the attacker does not have the satellite's private key, then it cannot impersonate it or forge its chains;
– replay-related security: i.e., replay attacks are possible, but only if the acceptability time interval is not violated (i.e., messages replayed too late will be rejected).

b) ***Verification Results****:* The model has about 800 lines of code. It was executed on a stand-alone laptop with an AMD Ryzen 7 PRO 7840U (8 cores/16 threads at 3.3 GHz/5.1 GhZ Turbo Speed) with 64GB or RAM. All lemmas can be proven automatically, with the supplied oracle, in about 70 minutes and needing cca. 6GB of memory.

c) ***Significance****:* Our models and results mean the following: Via our modelling of time, combined with our correctness and soundness lemma, we showed that we now have a first correct and functional model for formally verifying the security of the TESLA protocol, which so far has proven hard to model formally in computer-aided tools. Via our oracles and (reasonable) restrictions, the verification is tractable even on a laptop. These two aspects together give the research community a significant head-start in any future models of TESLA, its future incarnations, or likely adaptations to other GNSS services beyond Galileo. Via our security lemmas, we *formally* show that authentication and integrity of navigation messages is achieved, if –as per TESLA's assumptions– the sender and receiver are time-synchronised and the private key of the sender is not leaked.

V. Discussions & Conclusions

We put forward the first model able to formally reason on the correctness and security of the Galileo-adopted TESLA protocol, for authenticating navigation/GNSS messages sent by TESLA satellites.

There are other GNSS-centric versions of TESLA, not adopted in Galileo or in other GNSS services, primarily stemming from [20]: TESLA versions where in the same message-chain the messages come from several satellites: e.g., these satellites share the initial TESLA key k_0 and/or have a joint/group signature. Depending on how much they share and how the navigation messages are formed, there are variations of how the authentication works, and whether a group/constellation or a single satellite is authenticated. These versions were introduced with efficiency in mind. It is clear there are some attacks associated with them (e.g., if k_0 is shared, and satellite 1 sends it first, this can be used to spoof some satellite 2, for some period of time, at least). So, these not-yet-adopted versions of TESLA show the classic trade-off between security and efficiency.

However, whilst these versions are not adopted, the future may favour their efficiency slant. So, looking into formally modelling them and thus at ascertaining a provably acceptable trade-off between security and efficiency, is a future direction.

At the same time, we are working on mechanised *cryptographic* models [21] (i.e., not Dolev-Yao models) for TESLA, which –similarly to this line– are hindered by inabilities to accurately capture timing-based threats and continuous streaming inside message-chains in computational tools such as Squirrel [22].

REFERENCES

[1] T. Westbrook, "Trojan spoofing: A threat to critical infrastructure," *Security and Defence Quarterly*, vol. 42, no. 2, pp. 1–15, 2023.

[2] European Union Agency for Space Programme (EUSPA), "Navigation message authentication schemes," https://www.gsc-europa.eu/galileo/services/galileo-open-service-navigation-message-authentication-osnma, 2023.

[3] E. U. A. for Space Programme (EUSPA), "OSNMA Developments," https://www.euspa.europa.eu/sites/default/files/expo/osnma_public_testing_javier_simon_v3.pdf, 2023.

[4] A. Perrig and J. D. Tygar, *Secure Broadcast Communication: In Wired and Wireless Networks*. Springer Science & Business Media, 2003.

[5] I. Fernández-Hernández, V. Rijmen, G. Seco-Granados, J. Simon, I. Rodríguez, and J. D. Calle, "A navigation message authentication proposal for the galileo open service," *NAVIGATION: Journal of the Institute of Navigation*, vol. 63, no. 1, pp. 85–102, 2016.

[6] S. Meier, "Last commit of a TESLA model in Tamarin 2012-not tractable," https://github.com/search?q=repo3Atamarin-prover%2Ftamarin-prover%20tesla&type=code, 2012.

[7] A. Kurmus, "TLS renegotiation vulnerability (CVE-2009-3555)," 2009.

[8] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 483–502.

[9] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1313–1328.

[10] A.-I. Radu, T. Chothia, C. J. Newton, I. Boureanu, and L. Chen, "Practical EMV relay protection," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1737–1756.

[11] K. G. Paterson and T. van der Merwe, "Reactive and proactive standardisation of tls," in *Security Standardisation Research: Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5–6, 2016, Proceedings 3*. Springer, 2016, pp. 160–186.

[12] A. Armando, D. Basin, Y. Boichut, and et al., "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications," in *CAV*, 2005.

[13] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *CAV*, 2013, pp. 696–701.

[14] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *IEEE CSFW*, 2001.

[15] D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *IEEE Trans. Inf. Theory 29*, vol. 29, no. 2, 1983.

[16] The Tamarin Team, "Tamarin prover manual," 2024. [Online]. Available: https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf

[17] The Tamarin Team, "Tamarin Examples: Loops - TESLA (Scheme1)," 2024. [Online]. Available: https://projects.cispa.saarland/c01yaiv/tamarin-prover/-/blob/master/examples/loops/TESLA_Scheme1.spthy

[18] The Tamarin Team, "Tamarin Examples: Loops - TESLA (Scheme2)," 2024. [Online]. Available: https://projects.cispa.saarland/c01yaiv/tamarin-prover/-/blob/master/examples/loops/TESLA_Scheme2.spthy

[19] The Tamarin Team, "Tamarin Examples: Loops - TESLA (Scheme1)," 2024. [Online]. Available: https://projects.cispa.saarland/c01yaiv/tamarin-prover/-/blob/master/examples/loops/TESLA_Scheme2_lossless.spthy

[20] I. F. HERNANDEZ, "Method and system to optimise the authentication of radionavigation signals," Aug. 4 2020, uS Patent 10,732,290.

[21] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, Nov. 2004, available at http://eprint.iacr.org/2004/332.

[22] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and J. Lallemand, "The Squirrel Prover and its Logic," *ACM SIGLOG News*, vol. 11, no. 2, Apr. 2024. [Online]. Available: https://inria.hal.science/hal-04579038