Journal of Computer Security 0 (2018) 1
IOS Press

Secure Authentication in the Grid: A Formal Analysis of DNP3 SAv5

Cas Cremers [a], Martin Dehnel-Wild [b,*] and Kevin Milner [b]
[a] *CISPA Helmholtz Center i.G., Saarland Informatics Campus, Saarbrücken, Germany*
[b] *Department of Computer Science, University of Oxford, UK*
E-mails: cas.cremers@cispa.saarland, martin@dehnelwild.co.uk, kamilner@kamilner.ca

Abstract. Most of the world's power grids are controlled remotely. Their control messages are sent over potentially insecure channels, driving the need for an authentication mechanism. The main communication mechanism for power grids and other utilities is defined by an IEEE standard, referred to as DNP3; this includes the Secure Authentication v5 (SAv5) protocol, which aims to ensure that messages are authenticated.

We provide the first security analysis of the complete DNP3: SAv5 protocol. Previous work has considered the message-passing sub-protocol of SAv5 in isolation, and considered some aspects of the intended security properties. In contrast, we formally model and analyse the complex composition of the protocol's sub-protocols. In doing so, we consider the full state machine, the protocol's asymmetric mode, and the possibility of cross-protocol attacks. Furthermore, we model fine-grained security properties that closely match the standard's intended security properties. For our analysis, we leverage the TAMARIN prover for the symbolic analysis of security protocols.

Our analysis shows that the core DNP3: SAv5 design meets its intended security properties. Notably, we show that a previously reported attack does not apply to the standard. However, our analysis also leads to several concrete recommendations for improving future versions of the standard.

Keywords: DNP3, Secure Authentication, Power Grids, Network Protocols, Formal Analysis

1. Introduction

Most of the world's power grids are monitored and controlled remotely. In practice, power grids are controlled by transmitting control and monitoring messages, between authorised operators ('users') that send commands from control centers ('master stations'), and substations or remote devices ('outstations'). The messages may be passed over a range of different media, such as direct serial connections, ethernet, Wi-Fi, or un-encrypted radio links. As a consequence, we cannot assume that these channels guarantee confidentiality and authenticity.

The commands that are passed over these media are critical to the security of the power grid: they can make changes to operating parameters such as increases or decreases in voltage, opening or closing valves, or starting or stopping motors [1]. It is therefore desirable that an adversary in control of one of these media links should not be able to insert or modify messages. This has motivated the need for a way to authenticate received messages.

The DNP3 standard, more formally known as IEEE 1815-2012, the "Standard for Electric Power Systems Communications – Distributed Network Protocol" [2], is used by most of the world's power grids for communication, and increasingly for other utilities such as water and gas.

*Corresponding author. E-mail: martin@dehnelwild.co.uk.

Secure Authentication version 5 (SAv5) is a new protocol family within DNP3, and was standardised in 2012 (Chapter 7 of IEEE 1815-2012 [2], based on IEC/TS 62351-5 [3]). SAv5's goal is to provide authenticated communication between parties within a utility grid. For example, this protocol allows a substation or remote device within a utility grid to verify that all received commands were genuinely sent by an authorised user, that messages have not been modified, and that messages are not being maliciously replayed from previous commands.

Given the security-critical nature of the power grid, one might expect that DNP3: SAv5 would have attracted substantial scrutiny. Instead, there has been very little analysis, except for a few limited works. One possible explanation is the inherent complexity of the DNP3: SAv5 protocol, as it consists of several interacting sub-protocols that maintain state to update various keys, which results in a very complex state machine for each of the participants. Such protocols are notoriously hard to analyse by hand, and the complex looping constructions pose a substantial challenge for protocol security analysis tools. Moreover, it is not sufficient to analyse each sub-protocol in isolation. While this has been known in theory for a long time [4], practical attacks that exploit cross-protocol interactions have only been discovered more recently, e.g., [5, 6]. In general, security protocol standards are very hard to get right, e.g. [7–9].

Contributions

In this research, we provide the most comprehensive analysis of the full DNP3 Secure Authentication v5 protocol yet, leveraging automated tools for the symbolic analysis of security protocols. In particular:

- We provide the first formal models of two of the SAv5 sub-protocols that had not been modelled previously. In contrast to [10], the analysis in this work also includes the asymmetric mode of the *Update Key Change Protocol*.
- We provide the first analysis of the complex combination of the sub-protocols and all their modes, thereby considering cross-protocol attacks as well as attacks on any of the sub-protocols. The security properties that we model capture the standard's intended goals in much greater detail than any previous works.
- Despite the complexity of the security properties and the protocol, and in particular its complex state-machine and key updating mechanisms, and considering unbounded sessions and loop iterations, we manage to verify the protocol using the TAMARIN prover. We conclude that the standard meets its intended goals if implemented correctly, increasing confidence in this security-critical building block of many power grids.
- Notably, our findings contradict a claimed result by an earlier analysis; in particular, our findings show that an attack claimed by other work is not possible in the standard as defined.
- Our analysis naturally leads to a number of recommendations for improving future versions of the standard.

We start by describing the Secure Authentication v5 standard in Section 2. We describe the sub-protocols' joint modelling in Section 3, and their analysis and results in Section 4. We present our recommendations in Section 5, survey previous analyses of DNP3 in Section 6, before concluding this work in Section 7.

A public archive with our protocol models can be found at [11].

2. The DNP3 Standard

The DNP3 standard [2] gives both high level and semi-formal descriptions, to serve as an implementation guide, as well as providing an informal problem statement and conformance guidelines. The Secure Authentication v5 protocol is described in Chapter 7 of [2]. We give an overview of the system and its sub-protocols, before describing the threat model from SAv5.

2.1. System and Sub-Protocols

There are three types of actor in SAv5: the (single) **Authority**, the **Users** (operating from a Master station), and the **Outstations**. The Authority decides who are legitimate users, and generates new (medium-term) Update Keys for these users. Users send control packets to outstations, who act upon them if they are successfully authenticated. Outstations send back (similarly authenticated) monitoring packets. Each user can communicate with multiple outstations, and each outstation can communicate with multiple users. Users regularly generate new (short-term) **Session Keys** for each direction of this communication, and transport these keys to the outstations. These are updated and distributed using so-called **Update Keys**, which can be considered as medium-term keys. In the symmetric mode, update keys are distributed and updated using long-term **Authority Keys**. In the asymmetric mode, update keys are distributed and updated using long-term public/private key pairs for the Authority and each User and Outstation. These keys are used by four sub-protocols: the symmetric *Update Key Change Protocol*, the asymmetric *Update Key Change Protocol*, the *Session Key Update Protocol*, and the *Critical ASDU Authentication Protocol*. See Figure 1 for an overview of the sub-protocols' relationships.

Initial Key Distribution: Before any protocols are run, a long-term Authority Key and an initial medium-term update key must be pre-distributed to each party. If the parties support the optional asymmetric mode of the *Update Key Change Protocol*, public/private key pairs will be generated and the private halves securely distributed to their respective owners; the public keys are distributed to all involved parties. These keys are distributed "over a physical channel" (e.g. via USB stick) to the respective parties. N.B. Session Keys are *not* pre-distributed.

The *Session Key Update Protocol*: Before parties can exchange control or monitoring messages, the user and outstation must initialise session keys. This sub-protocol initialises (and later updates) a new, symmetric Session Key for each communication direction.

After $\sim$15 minutes or $\sim$1,000 critical messages (both configurable) the session keys will expire. The user and outstation run the *Session Key Update Protocol* again, where the user generates fresh symmetric session keys, and sends them to the outstation, encrypted with their current update key. These session keys *must* remain secret, but the secrecy of new keys importantly does not rely on the secrecy of previous session keys.

All sub-protocols use sequence numbers and freshly generated Challenge Data with the aim of preventing replay attacks.

The *Critical ASDU Authentication Protocol*: Outstations use this sub-protocol to verify that received control packets were genuinely sent by a legitimate user. Vice-versa, this sub-protocol allows a user to confirm that received monitoring packets were genuinely sent by a legitimate outstation. These packets are called 'Critical ASDUs', or Application Service Data Units. As this is an authentication-only protocol, **Critical ASDUs are not confidential.**

After this sub-protocol's first execution, the faster 'Aggressive Mode' may be performed: this cuts the non-aggressive mode's three messages to just one by sending the ASDU and a keyed HMAC in the same message.

The *Update Key Change Protocol*: After a longer time, the update key may expire. The user and outstation (helped by the Authority) will execute the *Update Key Change Protocol*. A new update key is created by the Authority, and sent to both the user and outstation.

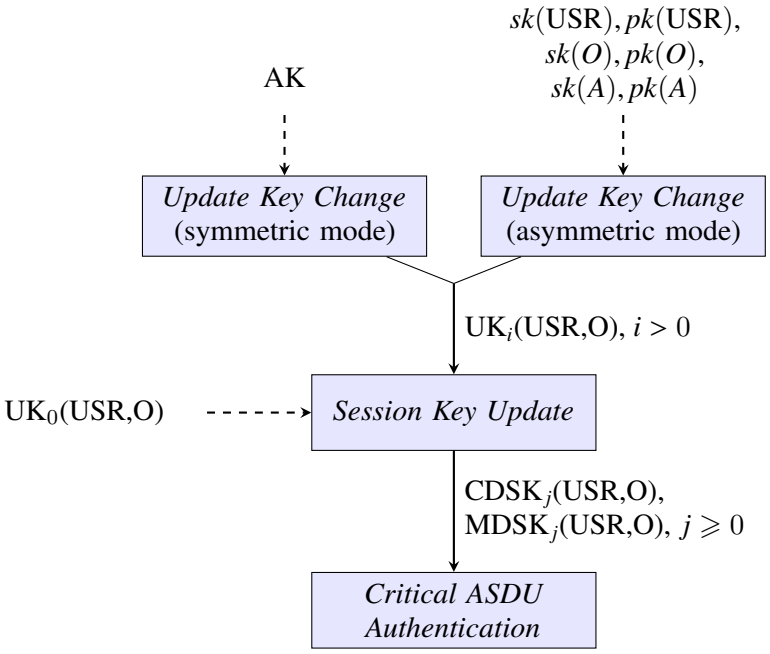

Fig. 1. Relationships between sub-protocols, the flow of keys between them (continuous lines), and required pre-shared keys (dashed lines).

2.2. Protocol Descriptions

We now give more detailed descriptions of the four sub-protocols in Secure Authentication v5. By way of notational preliminaries, $\{\!|\, m \,|\!\}^{s}_{k}$ denotes the symmetric encryption of term m under key k; similarly $\mathrm{HMAC}_k(m)$ denotes the HMAC of term m symmetrically keyed by k. Likewise, $\{\!|\, m \,|\!\}^{a}_{k}$ denotes the asymmetric encryption of term m under public key k, and $\{\!|\, m \,|\!\}^{asig}_{sk}$ denotes the asymmetric signature of term m under private key sk.

2.2.1. The Session Key Update Protocol*:*

See Figure 2. This is also the first sub-protocol run after a system restarts, to initialise the shared session keys.

S1. The user sends a Session Key Status Request. The user moves from "Init" to the state "Wait for Key Status".

S2. The outstation generates fresh challenge data CD_j, and increments its Key Change Sequence Number, KSQ. It sends a Session Key Status message (SKSM_j) to the user, containing the KSQ

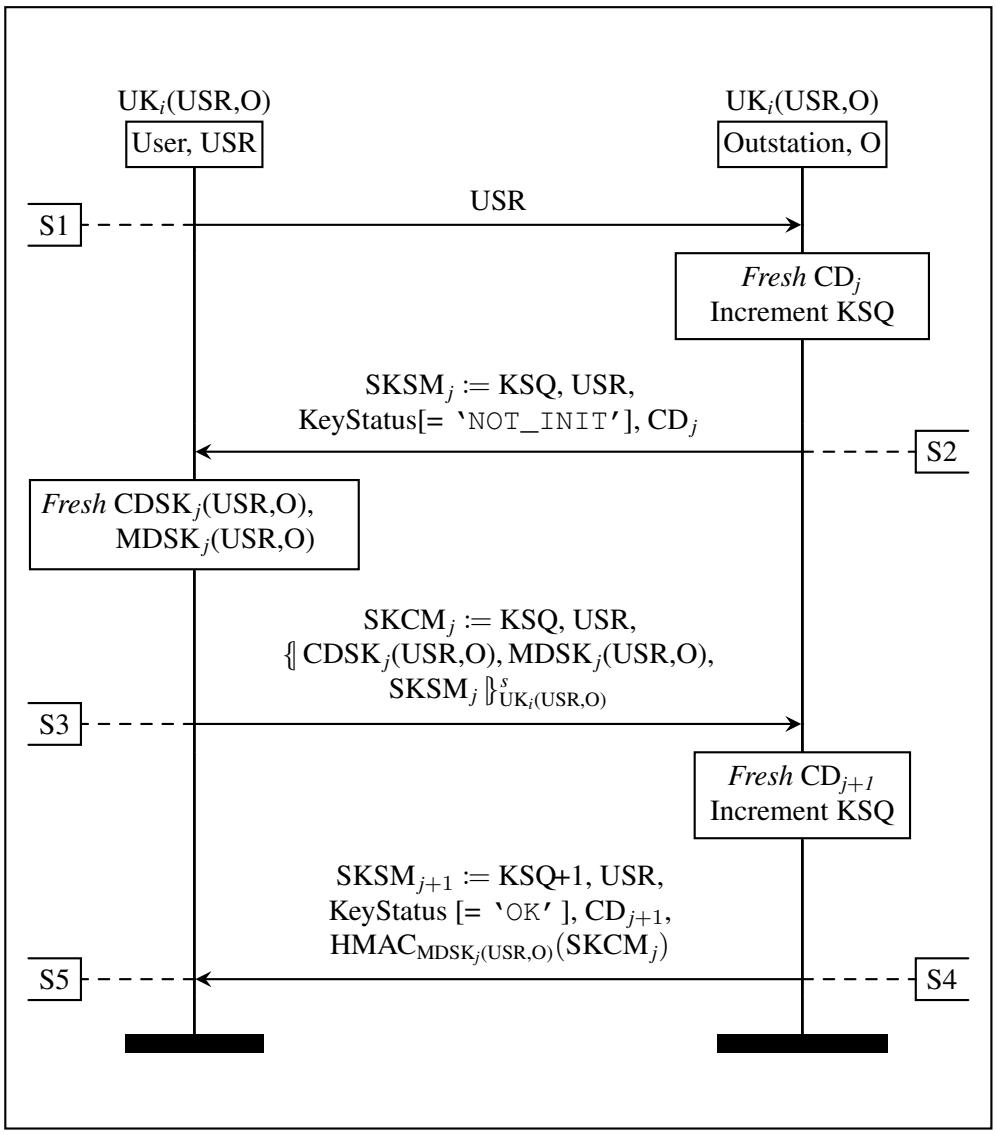

Fig. 2. The *Session Key Update Protocol*. The labels S1–5 identify the protocol rules described in Section 2.2.1

value, user ID, USR, Key Status, and CD$_j$. The outstation moves from "Start" to the state "Security Idle".

S3. The user generates two new session keys (one for each direction), CDSK and MDSK, and sends a Session Key Change Message to the outstation (SKCM$_j$). This contains the KSQ and USR values, and the encryption of the new keys and the previously received SKSM$_j$ message from the outstation, encrypted with the current symmetric update key. The user moves to the state "Wait for Key Change Confirmation".

S4. The outstation decrypts this with the shared update key, and checks that SKSM$_j$ is the same as it previously sent. If so, the outstation increments KSQ, and generates new challenge data, CD$_{j+1}$; it sends another Session Key Status Message (this time SKSM$_{j+1}$), but as session keys have been set,

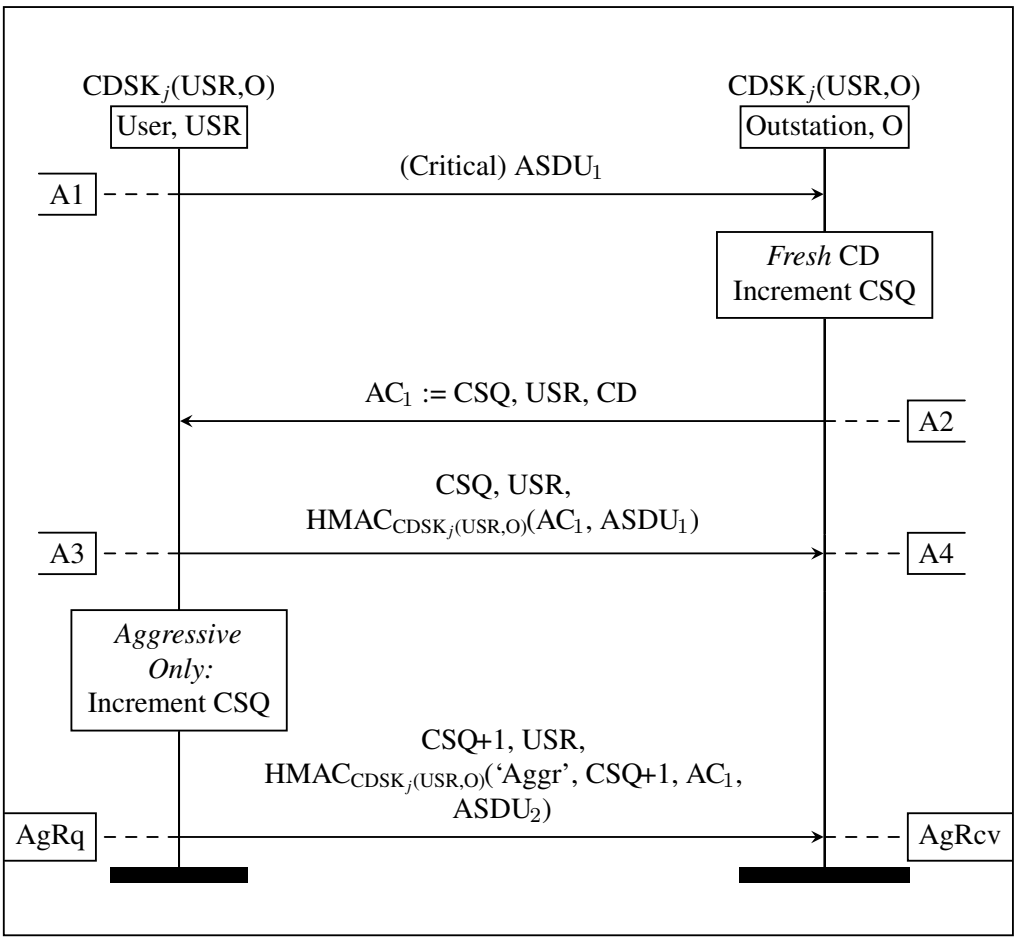

Fig. 3. The *Critical ASDU Authentication Protocol*, Control Direction, Non-Aggressive *and* Aggressive Modes. The labels A1–4 identify the protocol rules described in Section 2.2.2

the message now also includes an HMAC of SKCM$_j$, keyed with the MDSK.

S5. The user verifies that the received HMAC was generated from SKCM$_j$. If so, the user and outstation start to use the new session keys. If not, the user and outstation mark the keys as invalid, and retry the protocol. The user state moves to "Security Idle".

2.2.2. The Critical ASDU Authentication Protocol*:*

See Figure 3. This is the main data authentication protocol, and is used to verify the authenticity of critical ASDUs. This can only run *after* the first execution of the *Session Key Update Protocol*, and it can run in both the control and monitoring directions, User→Outstation and Outstation→User respectively. Here we present it in the control direction; the direction determines which key is used for the HMAC in the final message (A3), i.e. CDSK or MDSK. First, the non-aggressive mode; both parties start in the state "Security Idle":

A1. The user sends a critical ASDU, which the outstation must authenticate.

A2. On receipt of this ASDU, the outstation increments its Challenge Sequence Number, CSQ, and sends an Authentication Challenge (AC), which contains the user's ID, USR, fresh challenge data,

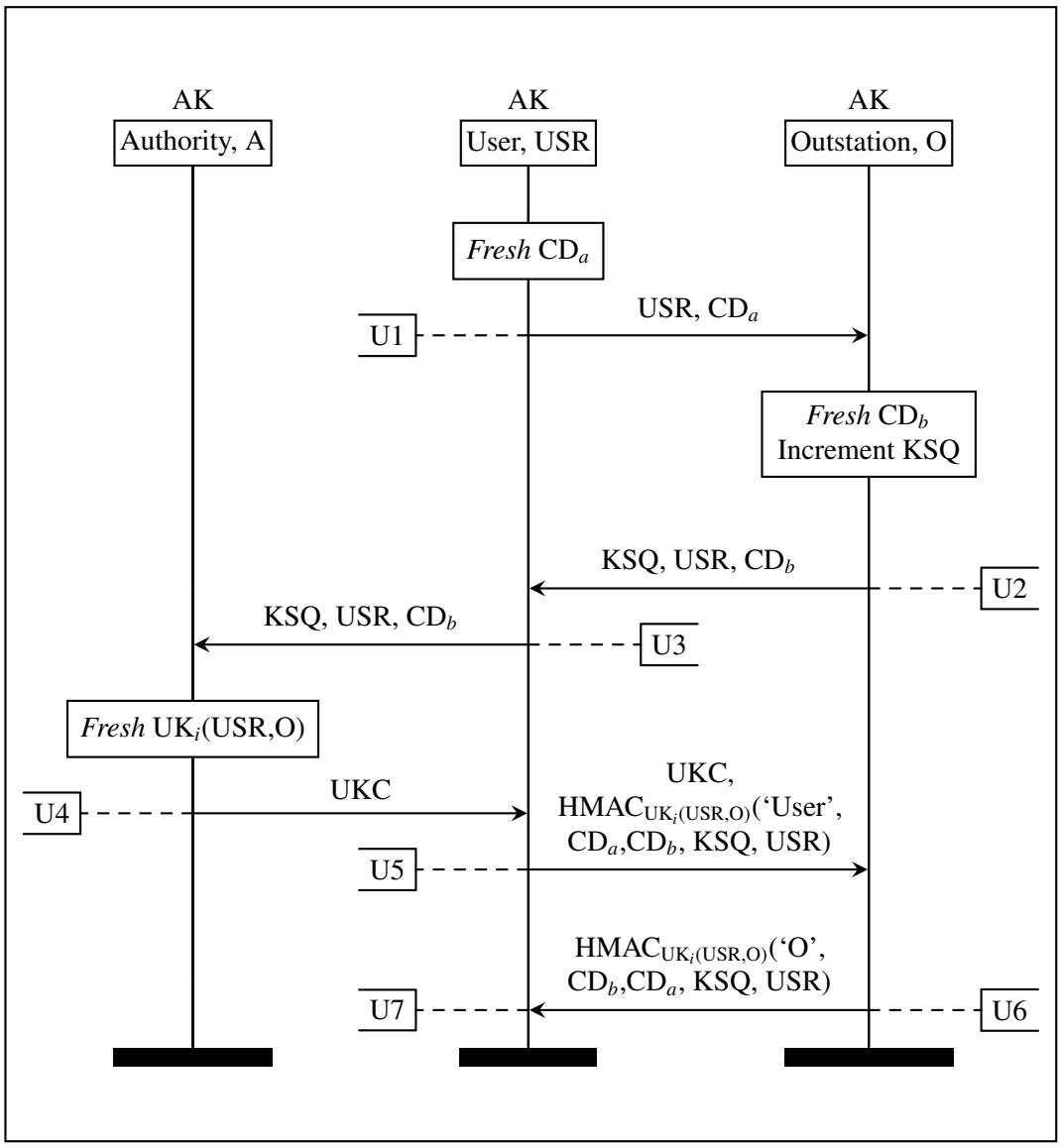

Fig. 4. The *Update Key Change Protocol*, symmetric mode. The labels U1–7 identify the protocol rules described in Section 2.2.3. In U4 and U5, UKC is the tuple KSQ, USR, $\{\!|\, \mathrm{USR}, \mathrm{UK}_i(\mathrm{USR,O}), \mathrm{CD}_b \,|\!\}^{s}_{\mathrm{AK}}$

CD, and the CSQ value. The outstation moves to the state "Wait for Reply".

A3. The user sends an Authentication Reply message, which contains the CSQ, USR, and an HMAC of the previously received Authentication Challenge message, AC, and the critical ASDU it seeks to authenticate. This HMAC is keyed with the Control Direction Session Key, CDSK.

A4. The outstation verifies that the HMAC was constructed with the AC message it sent, the critical ASDU, and keyed with the current CDSK. If it succeeds, the outstation acts upon this critical ASDU; if it fails, it does not execute it. Regardless of the outcome, the outstation returns to the state "Security Idle".

Aggressive Mode: Once the non-aggressive sub-protocol has run once, the user may send an Aggressive Mode Request ('AgRq' in Figure 3). This contains both the new ASDU to be authenticated, the incremented CSQ, and an HMAC in the same message. This HMAC is calculated over the last Authentication Challenge message the user received, and the entire preceding message it is being sent in.

The outstation then checks ('AgRcv' in Figure 3) that the HMAC was constructed with the last Authentication Challenge, and that the CSQ is incremented from the last message. If so, it accepts and acts upon the ASDU.

2.2.3. The Update Key Change Protocol *(Symmetric Mode):*

See Figure 4. This key-transport sub-protocol has two modes; symmetric (default) and asymmetric (optional; described in Section 2.2.4). Both allow users and outstations to change the symmetric update key used by the previous protocol. Both devices start in "Security Idle"; the outstation always remains here. The symmetric mode of this protocol proceeds as follows:

U1. The user sends an Update Key Change Request message, containing the user's ID, USR, and freshly generated challenge data, CD_a. The user moves to the state "Wait for Update Key Reply".

U2. Upon receipt of this message, the outstation increments its Key Change Sequence Number (the same variable as in the previous sub-protocol), and also generates fresh challenge data, CD_b. It sends the new value of KSQ, USR and CD_b to the user in an Update Key Change Reply message.

U3. The user forwards this message on to the Authority.[1]

U4. The Authority creates a new update key. It encrypts the key, USR, and CD_b with the Authority Key, and transmits it, KSQ, and USR back to the user.

U5. The user decrypts this, and forwards both this message (Update Key Change), and an Update Key Change Confirmation (UKCC) message to the outstation. This is an HMAC of the user's full name, both challenge data (CD_a and CD_b), KSQ, and USR, and it is keyed with the *new* update key. The user moves to the state "Wait for Update Key Confirmation".

U6. The outstation decrypts the first part of the message to learn the new update key, and verifies that the UKCC HMAC was created with the correct challenge data and KSQ from step U2. If so, it sends back its own UKCC message (also keyed with the new update key), but with the order of the challenge data swapped, and with its name, rather than the user's.

U7. If the user can validate this HMAC (by checking that it was created with the challenge data and KSQ values from this same protocol run, keyed with the new update key), then it accepts the message, and both parties start to use the new update keys. If this fails, the parties retry the protocol. Regardless of outcome (except timeout), the user moves back to the state "Security Idle".

2.2.4. The Update Key Change Protocol *(Asymmetric Mode):*

The asymmetric mode of the *Update Key Change Protocol* is very similar to the symmetric mode. See Figure 5 for the message sequence chart of the Asymmetric mode of the *Update Key Change Protocol*. The overall message flows are similar to the symmetric mode, and messages are effectively identical for U1–3. In message U4, the message transmitted is encrypted asymmetrically, not symmetrically. In message U5, the UKC object within the message (`g120v13`) is asymmetrically encrypted with the Outstation's public key. The overall U5 message also includes an Update Key Change Signature object (`g120v14`), rather than a (symmetrically encrypted) Update Key Change Confirmation object (`g120v15`). The Update Key Change Signature object is the signature over:

– Outstation Name

[1] U3 and U4 are technically out of scope for DNP3: SAv5.

– User Challenge Data (CD_a in the MSC)
– Outstation Challenge Data (CD_b in the MSC)
– Key Change Sequence Number (KSQ)
– User Number (USR)
– Encrypted Update Key Data (from UKC)

This object is signed by the User's private key, $sk(\mathrm{USR})$, which allows the Outstation to verify the signature with the User's public key, $pk(\mathrm{USR})$. In message U6, the Outstation then sends back to the User an object signed by the Outstation's private key, i.e. $sk(\mathrm{O})$. This signature is over all the same data as the previous signature (from U5), but as it is signed by the Outstation's private key, an adversary should not be able to forge this only from knowledge of the terms listed above and the previously seen signature (from U5).

2.3. Threat Model and Security Properties

In this section we describe how we arrived at the threat model and security properties that we formally analyse. This is not as straightforward as one might think, as security properties are often informally and minimally described in protocol standards. For transparency, we will quote the original standards where possible. We use colored boxes to denote verbatim quotations from other documents.

The standard has a "Problem description" section [2, p.13] that describes "the security threats that this specification is intended to address". We reproduce this section *in its entirety* below:

5.2 Specific threats addressed (from IEEE 1815-2012 [2] p. 13)

This specification shall address only the following security threats, as defined in IEC/TS 62351-2:

- spoofing;
- modification;
- replay
- eavesdropping — on exchanges of cryptographic keys only, not on other data.

Additionally, the general principles section contains a subsection "Perfect forward secrecy" that suggests an implicit security requirement. We could not determine any other sections that would imply security requirements.

The wording of the above section suggests that all listed terms are defined in IEC/TS 62351-2 [12]. This is not the case: [12] defines only some of these concepts. In particular, "modification" and (perfect) "forward secrecy" are not defined. We address the listed concepts in turn, starting from the ones which are defined.

Spoofing. The standard specifies that spoofing is defined through [12] as:

2.2.191 Spoof (from IEC/TS 62351-2 [12] p.39)

Pretending to be an authorized user and performing an unauthorized action.
[RFC 2828]

While this definition references RFC 2828 [13], there is a difference, in that [13] equates spoofing and masquerading, but does not reference unauthorized actions:

spoofing attack (from RFC 2828 [13])

(I) A synonym for "masquerade attack".

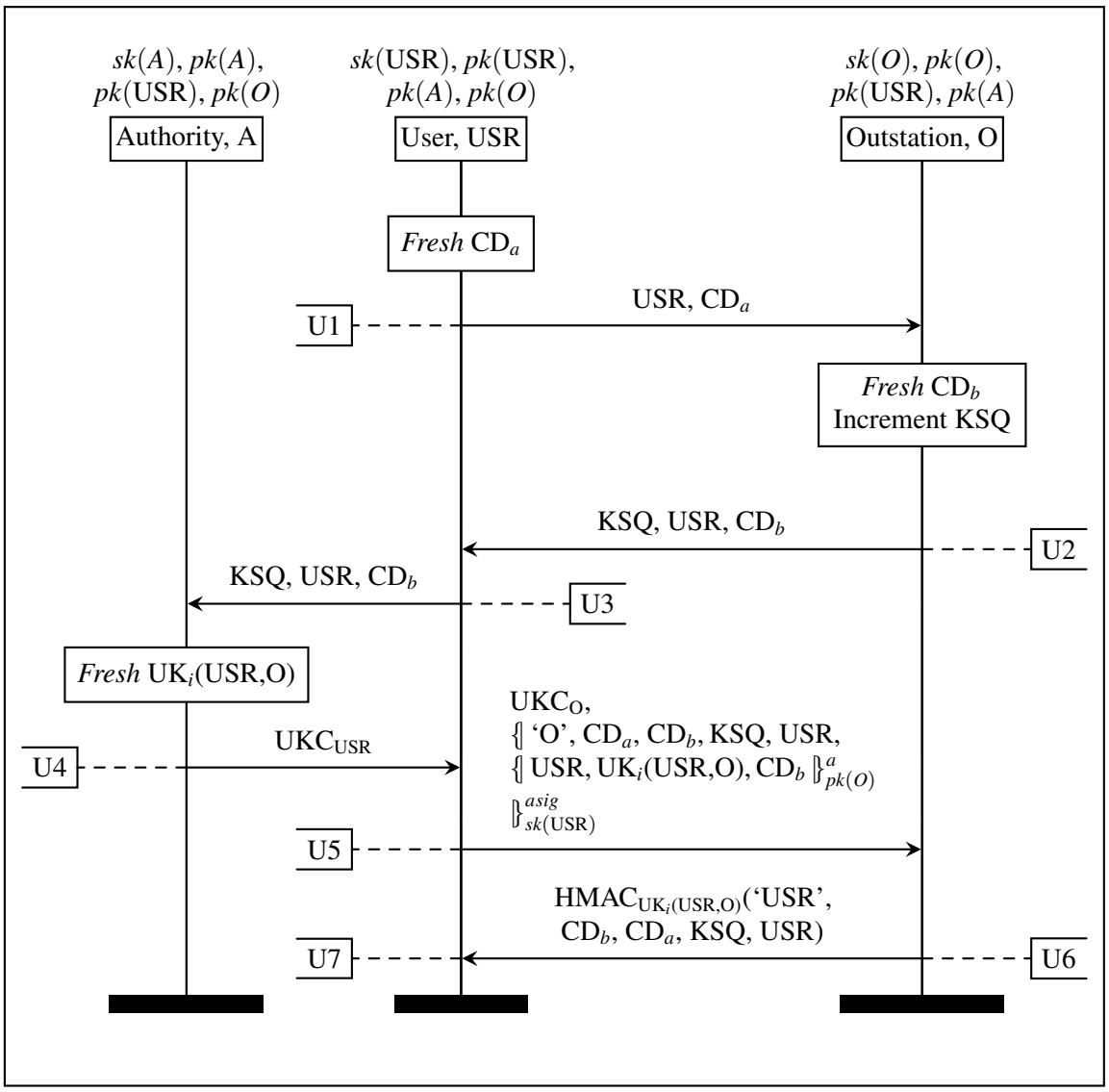

Fig. 5. The *Update Key Change Protocol*, asymmetric mode. The labels U1–7 identify the protocol rules described previously. In messages U4 and U5, UKC_x is equal to the tuple: KSQ, USR, $\{\!|\,\text{USR}, \text{UK}_i(\text{USR,O}), \text{CD}_b\,|\!\}^{a}_{pk(x)}$, where x is the intended recipient of the message (i.e. the User in message U4, and the Outstation in message U5).

where masquerade is defined in the RFC as

masquerade attack	(from RFC 2828 [13])
a type of attack in which one system entity illegitimately poses as (assumes the identity of) another entity. (see: spoofing attack.)	

Thus, the RFC equates spoofing and masquerading. Analogously, the DNP3 standard directly relies on [12], which defines masquerading as

2.2.131 Masquerade	(from IEC/TS 62351-2 [12] p.30
The pretence by an entity to be a different entity in order to gain unauthorized access. [ATIS]	

Here, ATIS [14] is a glossary from which this particular definition is taken. Hence it seems that within the context of DNP3, spoofing and masquerading are interchangeable, similar to the statements in RFC 2828. However, the definitions in the DNP3 standard [3] are closer to [14] than to [13], since they additionally include the aspect of unauthorized access/action. Note that the DNP3 standard has no explicit concept of authorization; this seems out of the standard's scope.

Replay

2.2.159 Replay Attack	(from IEC/TS 62351-2 [12] p.35)
1. A masquerade which involves use of previously transmitted messages. [ISO/IEC 9798-1:1997]	

This is a verbatim copy of a similar section in the reference ISO/IEC 9798-1:1997 [15], and suggests that replay is a special case of masquerading/spoofing.

Eavesdropping

2.2.92 Eavesdropping	(from IEC/TS 62351-2 [12] p.25)
Passive wiretapping done secretly, i.e., without the knowledge of the originator or the intended recipients of the communication. [RFC 2828]	

This is a verbatim copy from the definition in the reference RFC 2828 [13]. However, DNP3 adds the specific restriction to the confidentiality of keys, as the main purpose of the standard is to authenticate messages that are not confidential.

Modification There is no explicit definition: we interpret this as an integrity requirement: adversaries must not be able to modify transmitted messages.

Perfect Forward Secrecy The general design text contains:

5.4.10 Perfect forward secrecy	(from IEEE 1815-2012 [2] p. 16)
This specification follows the security principle of perfect forward secrecy, as defined in IEC/TS 62351-2. If a session key is compromised, this mechanism only puts data from that particular session at risk, and does not permit an attacker to authenticate data in future sessions.	

Surprisingly, IEC/TS 62351-2 [12] does not mention the concept of (perfect) forward secrecy. However, the informal explanation suggests that the loss of some session keys should not affect authentication of future sessions with, presumably, different session keys.

Adversary Capabilities The standard states that communications might be performed over insecure channels, and this suggests the threat model includes adversaries that can manipulate or insert messages.

The standard additionally states that "if update keys are entered or stored on the device in an insecure fashion, the entire authentication mechanism is compromised" [2, p.21]. This suggests that some forms of compromise might be considered (e.g., of session keys), but not the full compromise (in which all stored data is compromised) of a party involved of a session.

3. Formal Model of SAv5 in TAMARIN

Our modelling and analysis of Secure Authentication v5 used the TAMARIN security protocol verification tool [16]. TAMARIN is a symbolic tool which supports both falsification and unbounded verification of security protocols specified as multiset rewriting systems with respect to (temporal) first-order properties. We give a brief overview of TAMARIN in Section 3.3, and an example of its syntax in Section 3.3.1; for more detail on the theory and use of TAMARIN see [16, 17] and https://tamarin-prover.github.io.

3.1. Symbolic Modelling Assumptions

Symbolic analysis does not consider computational attacks on a protocol, instead focusing on the logic of protocol interactions. This requires us to make assumptions about the primitives used in the protocol, which restricts the power of the analysis. We make the following assumptions:

- Dolev-Yao Adversary: the adversary controls the network channels [18].
- Symbolic Representation: information is contained in terms. Any party (including the adversary) can either know a term in its entirety, or not know it at all. A party cannot learn e.g. a single bit of a term, or half a term.
- Perfect Cryptography: we assume that the cryptographic primitives used are perfect. This means that e.g. an adversary can learn the term m from the symmetrically encrypted $\{\!|\, m \,|\!\}^{s}_{k}$ term if and only if it knows the key, k.
- Hash Functions: we assume that hash functions are one-way and injective.
- Randomness: we assume all freshly generated random terms are unpredictable, and unique (no two fresh terms generated separately are equal).

3.2. Complexity of the Protocol

Each of the protocols within Secure Authentication v5 are individually straight forward; however, much more complexity becomes apparent when they interact. To give an indication of the state machines, see Figure 6 for a diagram showing the state transitions performed by the user. The system starts in state 0; each node is the state the user is in before it executes a rule along one of the outgoing edges. These edges are labelled with the name of the rule which the user executes during the transition into another state (these names are the same as in the Message Sequence Charts). This diagram demonstrates how multiple loops can occur in many different orders, with very little determined structure, and how little of the relevant state is represented by the standard's state machines. Each protocol can loop many times (below certain large thresholds), making the possible routes through the state machines and state-space very large and complex indeed. As there is stored data associated with each of these states, we do not get injective correspondence with the named states from the SAv5 specification.

Much of the complexity within Secure Authentication v5 comes from the transitions between different stateful sub-protocols as well as the multiple directions of the *Critical ASDU Authentication Protocol*. Each of these sub-protocols updates some part of agent state, and each of them rely on parts of state updated by other sub-protocols.

For example, the ability for an outstation to receive an aggressive mode request depends on:

(1) their pairing to a user, set at initialization and invariant across all sub-protocols;

(2) their current state machine state, which must be set to `SecurityIdle` or `WaitForReply`. This status is invariant through the both the symmetric and asymmetric versions of the *Update Key Change Protocol* but modified by the other two sub-protocols;

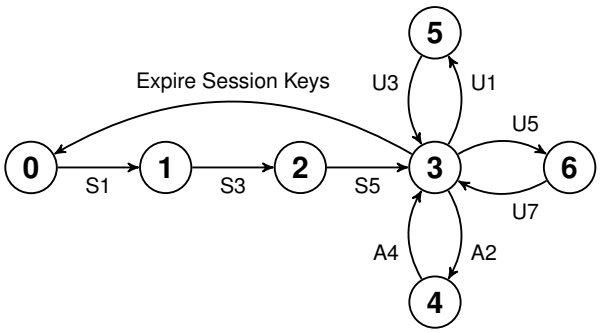

Fig. 6. A simplified version of the user's state machine as defined in the standard, excluding error transitions and the monitoring direction of the *Critical ASDU Authentication Protocol*. Note that although many transitions occur from the same state, they are conditional on additional state that is not represented in the state machine as described by the standard.

(3) their current session keys, set by the previously completed *Session Key Update Protocol*, and invariant across the other sub-protocols;

(4) the status of those session keys, which is invariant across all sub-protocols but can be changed at any time through a key expiry state transition;

(5) the 'current' challenge from the user, modified when a non-aggressive mode request was successfully verified by the outstation, which is invariant in the aggressive mode protocol variant and all other sub-protocols;

(6) and the number of aggressive mode requests received by the outstation since the current challenge was set, which is invariant in all other sub-protocols.

Each of these parts of state can be updated independently of each other, and each is invariant over several sub-protocols.

The state machine described in the protocol specification [2] captures very little of the protocol logic; the current state machine state is only one of the six dependencies listed above, and in fact is invariant for the outstation through both the *Update Key Change Protocol* and *Session Key Update Protocol* sub-protocols. An accurate TAMARIN model must include this much larger range of transitions, as well as their associated errors and timeouts; it is not possible to directly map the states and transitions in the specification to those in the model. This greatly increases the difficulty of interpreting the specification.

3.3. Protocol Modelling in TAMARIN

In TAMARIN, the possible executions of a protocol in the presence of an adversary are modelled as the possible executions of a labeled transition system. The state of the transition system is modelled as a multiset of so-called **facts**. Facts have **terms** as optional arguments, which in turn are used to represent messages symbolically. The initial state of the system is the empty multiset.

The possible transitions in the transition system are specified by multiset rewrite **rules**, of the form:

```
rule RULENAME:
    [ PREMISES ] --[ ACTIONS ]-> [ CONCLUSIONS ]
```

Each rule specifies three multisets of facts: the Premises, the Actions (or labels), and the Conclusions. The premises of a rule are *facts* that must exist in, and are removed from, the multiset prior to the rule's

execution; conclusions are facts that are added to the multiset by executing the rule. Facts within rules may contain variables. The Actions do not affect the possible transitions of the system - instead, they are logged into the action trace. Security properties are specified over these action traces, as we will see later.

By default, facts are linear, in the sense that if they appear only in the premise of a rule, they are consumed by performing the corresponding transition. TAMARIN additionally supports persistent facts, which are not consumed when such a rule is triggered, even when they only appear in the premise.

There are a few special facts used in TAMARIN. The Dolev-Yao adversary [18] is modelled through special `In` and `Out` facts that model the interaction between the protocol and the untrusted network. These facts respectively ask the adversary to provide, or provide to the adversary, the term argument of the fact. Additionally, there is a `Fr` fact to represent symbolically generating a fresh random value.

All four sub-protocols' rules and interactions were modelled as multiset rewriting rules in TAMARIN's specification language. Additional rules model the adversaries' capabilities, such as the ability to compromise certain keys. The final model comprises 36 multiset rewriting rules in ∼560 SLoC, with 10 types of invariants represented as actions within these rules, while others are captured in the state transitions themselves. The model and associated theorems are contained in the file `dnp3.m4`, which can be found at [11].

3.3.1. TAMARIN *Example Rule*

We now give an example of a SAv5 rule modelled in TAMARIN.

Example 3.1. The following rule is an example of a multiset rewriting protocol rule in TAMARIN's syntax. Note the use of '`let ... in ...`' for syntactic replacement, and angle brackets '`<..., ...>`' for tuples. The rule is an abstracted implementation of the A3 'Send Authentication Reply' message rule, in the control direction, from the *Critical ASDU Authentication Protocol* described in Section 2.2.2.

```
rule A3_C:
    let AC = < CSQ, USR, CD >
        AR = < CSQ, USR, hmac(<CSQ,AC,ASDU>, CDSK) >  in
    [ UserState(USR, OS, MDSK, CDSK, LastControlChallenge, [...],
                'SecurityIdle')
    , In(AC) ]
  --[ SentASDU(USR, OS, AR, 'NonAggressiveMode', 'ControlDirection')
    , UsingSessionKeys(CDSK,MDSK)
    ]->
    [ UserState(USR, OS, MDSK, CDSK, AC, [...], 'SecurityIdle')
    , Out(AR) ]
```

The A3 rule defines the behaviour when a user (identified by `USR`) receives a challenge in response to a critical ASDU sent to an outstation (here identified by `OS`). The challenge message `AC` contains a `CSQ` sequence number, the `USR` identifier, and some challenge data `CD`. The user generates a reply, `AR`, including an HMAC of the challenge message, `AC`, and the ASDU under the relevant session key.

Because TAMARIN's rules operate on the (global) multiset state, we need to keep track of the state of individual threads: in this model, we use the `UserState` facts to represent the current state of each user. In particular, the premises of the above rule require a fact `UserState` containing terms representing stored values of the user, and the user is required to be in the `SecurityIdle` state. They also require a message from the network, containing three terms, the second of which must be equal to the user identifier `USR` in the `UserState` fact.

The actions contain two labels to refer to later, one recording that the user `USR` sent an authenticated reply `AR` intended for the outstation `OS`, in the non-aggressive mode and in the control direction. The other records the session keys that the user had in their state at the time of the rule's execution.

The conclusions of the rule output an updated user state, in which the term `LastControlChallenge` used for aggressive mode is changed to the new challenge just received, as well as the message to the network `AR` defined above.

The state machines described in [2] (corresponding to the transitions discussed in Section 2.2) capture very little of the protocol logic, as the allowed transitions depend more on values in memory than on the current state machine 'state'. As an example, the outstation remains entirely in the named state "Security Idle" throughout the *Update Key Change Protocol*; however, the outstation can only respond to certain messages from the user dependent on data from previously sent or received terms. Our TAMARIN models include this much larger range of transitions, as well as their associated errors and timeouts.

3.3.2. Modelling Choices, and Issues with the Specification

The finer details of SAv5's sub-protocols in [2] and [3] are very often unclear, under-specified, and open to interpretation. We now describe the larger issues we encountered, and how we chose to model them.

Challenge Sequence Numbers: The specification states that parties should keep one CSQ per direction, and *not* on a per-user basis [2, p.211-g]. It also implies that parties should keep count of the number of Authentication Replies and Aggressive Mode Requests it has sent since the last Authentication Challenge it has received, on a per-user basis [2, p.211-d]. The purpose of the CSQ is to match messages, and to ensure that replays are not possible [2, pp.207 & 211].

Instead of modelling precisely as described, we keep one CSQ per user, per direction (control and monitoring). If we do not do this, the universal CSQ values in a model must depend on *all* of the state machines running from the same station, which makes analysis infeasible.

Modelling CSQs in this manner is analagous to the specification: both keep a single value which allows the Challenger to check whether received messages contain the correct CSQ or not; the specification keeps a universal total *and a difference* on a per-user basis, we simply keep a per-user total; both interpretations require this incrementing value to be in Authentication Replies or Aggressive Mode Requests, to prevent replay attacks.

The specification has clear rules for when the *sender* of a CSQ should increment this value [2, p.211], but nothing about when a recipient should accept the value; it is not clear whether a received CSQ (e.g. in an Authentication Challenge) can be any value, whether it must be strictly increasing, or whether it must be precisely one higher than its last seen value. In our model, recipients of CSQ values check that they are strictly increasing.

Sequence Number Rollovers: CSQs and KSQs explicitly rollover to 0 after they reach 4,294,967,295 ($2^{32} - 1$); what a recipient of a rolled-over CSQ does is not defined. If CSQs are not strictly increasing, this is not an issue. If not, the way rollovers are handled needs to be done so correctly, or this might allow (very slow) replay attacks. We avoid this issue by not modelling rollovers.

Challenge Data: When parties should set and erase certain values, and how many historic values parties should save is not clear. Challenge data from previous messages is saved to enable alternate modes, e.g. saving the last Authentication Challenge (AC) in the (non-aggressive) *Critical ASDU Authentication Protocol* so that its Challenge Data can be used in Aggressive Mode Requests.

It is not clear when this should be stored, and when each party should erase its previous 'last sent challenge'. As an illustration: after the outstation (here the Challenger) has sent an AC, the user might not receive this AC message; if the user gets bored of waiting for an AC (which might never appear), it might send an Aggressive Mode Request with a critical ASDU. The user knows that it must construct this request with the last AC it received, which will not be the same as the last AC the outstation sent:

should the outstation accept this Aggressive Mode Request? Figure 7-28 of [2] implies it should, (if in the 'WaitingForReply' state) but what about after an invalid reply is received, or a message times out?

How to construct HMACs from the user's side is clear (both for the non-aggressive and aggressive modes of CAAP), but it is not so clear how to work out what construes a valid HMAC from the outstation's point of view.

We modelled this as follows: after sending an Authentication Challenge message, the outstation saves the challenge it is currently expecting to receive in an Authentication Reply. Upon timeout, other error, or successful reply, this challenge gets saved as the last sent challenge. This means an outstation can receive an aggressive mode request between the Authentication Challenge and a timeout, and still prevent it from being over-written.

Asymmetric Mode of the *Update Key Change Protocol*: The DNP3 specification says [2, p.753] "If the Key Change Method is asymmetric, the Update Key Data shall be encrypted using the outstation's public key". Unfortunately, it does not specify whether the Authority is the one encrypting it, or the User. For our interpretation, we can choose one of two paths: either, we assume the Authority sends two copies of UKC to the User, one encrypted with the User's public key, and the other encrypted with the Outstation's public key; the User then forwards on the second one to the Outstation. Alternatively, the Authority could send one UKC message to the User, encrypted with the User's public key. The User could then decrypt this message, and re-encrypt its contents under the Outstation's public key, before transmission.

As the communications between Authority and User are considered 'out of scope' for the DNP3 SAv5 spec, it does not make any comment on which of these two situations is the case. Secondly, the specification makes various comments about keeping the required bandwidth to a minimum. As such, we model it as the latter case, i.e. the User decrypts the received message, and re-encrypts under the Outstation's public key before forward transmission.

We address many of these issues by making recommendations for improvements to the specification in Section 5; this section addresses these and other issues encountered through our modelling, analysis, and understanding of best cryptographic practice.

4. Analysis and Results

4.1. Modelling the Threat Model and Security Properties

In TAMARIN, security properties are modelled as (temporal) first-order logical formulae. These are evaluated over so-called action traces that are generated by the protocol model. Protocol rules have as their second parameter a multiset of *actions*; when the rewrite system makes a transition based on a ground rule instance, the rule's actions are appended to the action trace. Thus, the action trace can be considered to be a log of the actions defined by the transition rules, in a particular execution. The modeller chooses what is logged, and this enables us to log appropriate events that enable the specification of the desired properties.

Modelling Adversary Capabilities As described in Section 2.3, the standard assumes that communication channels are not secure, so we assume the worst: the adversary fully controls the network, i.e., it can drop and inject arbitrary messages, and eavesdrop all sent messages. This model is known within symbolic security verification as the network part of the Dolev-Yao attacker model.

Based on the general principle of perfect forward secrecy, we additionally provide the adversary with the ability to compromise some (but not all) keys. In particular, when considering authentication or confidentiality properties, we will allow the adversary to compromise all session keys except for the CDSK/MDSK used for this particular critical ASDU. As a result, our model also considers any attacks on the authentication property that are based on the compromise of (different) earlier session keys, as described in the standard.

Modelling the Security Properties We now revisit each of the properties defined in Section 2.3 and describe how we interpret them for modelling purposes, resulting in three properties called AUTH1, AUTH2, and CONF.

Spoofing: AUTH1 The main security goal of SAv5 seems to be to prevent spoofing, i.e. to ensure that all critical ASDUs originate from the intended parties. This is classically specified as an authentication property. However, there is no canonical notion of authentication; instead, there are many subtly different forms (see, e.g. [19]). In this particular case, we choose a form of agreement, i.e., if party A receives a critical ASDU, then this exact message was sent by some B who agrees on the message and some additional parameters. In particular, the additional parameters we include here are the mode ("aggressive" or "non-aggressive") and the direction ("control" or "monitoring").

One complication is that classical authentication properties link identities: if Alice receives a message, she associates the sender with an identity (say, Bob), and the authentication property then encodes that Bob sent the message. However, in the case of SAv5, there are not always clear identities for parties, e.g., outstations. Instead, pairs of users and outstations are effectively linked through their initial (pre-distributed) update keys. Thus, the best we can hope to prove is that upon receiving a message, apparently from someone that initially had update key k, then the message was indeed sent by someone whose initial update key was k.

We thus model the following (relatively weak) agreement property, which we refer to as **AUTH1**: if an outstation or a user receives an Authentication Reply or Aggressive Mode Request message m in a mode x (where x is either "aggressive" or "non-aggressive") in direction y (where y is "control" or "monitor"), then this message m was sent in mode x for direction y by a party that had the same initial (pre-distributed) update key.

We consider the following adversary capabilities for this property: the adversary can compromise all session keys (CDSK or MDSK) except for the one used in the message m. This covers the "perfect forward secrecy" general principle. Additionally, we allow the adversary to compromise all update keys other than that used to assign the current session keys. Furthermore, if the current update key (used to assign the current session keys) was assigned in the symmetric mode of the *Update Key Change Protocol*, we allow the adversary to compromise all (asymmetric) private keys; if the current update key was assigned in the asymmetric mode, we allow the adversary to compromise the (symmetric) Authority Key. If neither of these is the case, the current update key must be the initial, pre-distributed update key, and the *Update Key Change Protocol* will not yet have successfully run.

Replay: AUTH2 Classically, replay refers to multiplicity: if Bob apparently completes N sessions with Alice, then Alice in fact ran at least N sessions with Bob. Phrased differently, an adversary should not be able to complete more sessions with Bob than Alice actually ran with Bob. However, the definitions in the standard suggest that replay should be interpreted as a special case of masquerading (and thus spoofing), which uses previously transmitted messages. From this we infer that some form of multiplicity or recentness is intended to be part of the anti-spoofing guarantee. We encode this as AUTH2, which is strictly stronger than AUTH1.

Thus, **AUTH2** additionally models so-called *injective* authentication, which captures the classical notion of replay prevention. Informally, it states that for each received message, there is a unique message sent. Thus, an attack in which an adversary tricks Bob into receiving a message twice which Alice only sent once violates the property.

Eavesdropping: CONF Since the standard considers non-confidential ASDU messages, there is no clear confidentiality requirement. However, the authentication guarantees can only be satisfied against an active adversary if the relevant keys remain confidential. Hence, a subgoal is to require confidentiality of keys. This should in particular hold against weaker adversaries, such as eavesdroppers.

We note that the prevention of spoofing attacks (as per the first requirement) implies that all the relevant keys (Authority Key, Update Key, and MDSK or CDSK) are confidential with respect to eavesdroppers. If they are not, the active adversary can trivially use them to spoof a message. We can still model these confidentiality requirements separately. This is useful for protocols that do not satisfy the authentication guarantees directly.

If the user chooses, encrypts, and transmits a new Session Key (e.g., `CDSK_1`) it is important that the adversary does not learn it. However, it is equally important that the adversary cannot e.g. block the transmission of `CDSK_1`, impersonate the user, and transmit different, adversary-chosen keys (e.g. `CDSK_2`) to the outstation. In the second case, `CDSK_1` might still be secret, but the adversary can still issue 'authentic' commands to the outstation, HMAC'd with `CDSK_2`. Since there are different key types, **CONF** is modelled as a set of confidentiality properties, one of each type of key and each perspective (role).

Modification. As stated before, this is not defined in the standard, and we interpret it as an integrity requirement. As such, it will be covered by our authentication guarantees AUTH1 and AUTH2.

Perfect forward secrecy. As noted in Section 2.3, this general principle indicates an intended resilience against the compromise of other session keys, and is covered by our adversary capabilities for the three properties.

4.1.1. Properties in TAMARIN

We now explore the full properties from our TAMARIN analysis. With each lemma, TAMARIN attempts to construct a counter-example to the stated property; if it cannot, (assuming termination) it concludes that the property is upheld by the protocol model.

As described, lemmas are modelled as temporal first-order logical formulae evaluated over action traces, i.e. action facts and timepoints. The syntax for specifying security properties is defined as follows:

- `#i` indicates that the term `i` is of sort 'temporal'
- `All` for universal quantification, `Ex` for existential quantification
- `==>` for implication, `&` for conjunction, `|` for disjunction, `not` for negation
- `f @ #i` for action constraints (Action Fact `f` occurs at timepoint `#i`)
- `#i < #j` for temporal ordering
- `#i = #j` for equality between temporal variables '`i`' and '`j`'
- `x = y` for equality between message variables '`x`' and '`y`'
- `K(x)` states that the adversary knows the term '`x`'

See [17] for further information and detail on modelling properties within TAMARIN.

First, we consider the confidentiality properties. This first property models the secrecy of the Update Keys:

```
lemma update_key_secrecy:
"(All id uk #i.
    not(Ex #r. UpdateKeyReveal( uk ) @ #r)
    & NewUpdateKey( id, uk, 'Initial', 'usb_stick' ) @ #i
    ==> not(Ex #j. K( uk ) @ #j)
    )
& (All id uk ak #i.
    not(Ex #r. AuthorityKeyReveal( ak ) @ #r )
            & not(Ex #r. UpdateKeyReveal( uk ) @ #r )
    & NewUpdateKey( id, uk, 'Symmetric', ak ) @ #i
    ==> not(Ex #j. K( uk ) @ #j)
    )
& (All id uk oprk uprk #i.
    not(Ex #r. OutstationPrivateKeyReveal( oprk ) @ #r)
    & not(Ex #r. UpdateKeyReveal( uk ) @ #r)
    & NewUpdateKey( id, uk, 'Asymmetric', < oprk, uprk > ) @ #i
    ==> not(Ex #j. K( uk ) @ #j)
    )
"
```

This lemma is a conjunction of three properties: each of these three sections must be true for the overall lemma to be true. The lemma specifies:

- If the initial update key itself isn't explicitly revealed, this implies the adversary doesn't know it (`K(...)`), and,
- If an update key encrypted using symmetric cryptography (under '`ak`') isn't revealed directly, and the authority key '`ak`' used to encrypt it also wasn't revealed, this implies the adversary doesn't know it, and finally,
- If an update key transported using asymmetric cryptography isn't revealed directly, and the private key used (by the outstation) to decrypt it isn't revealed, this implies the adversary doesn't know it.

As described, TAMARIN attempts to construct a counter-example to this statement. If TAMARIN is able to conclude that it cannot construct a trace which would violate the secrecy of the update key, it concludes that the model of the protocol upholds this property.

The second confidentiality property models the secrecy of the Session Keys.

```
lemma session_key_secrecy:
"(All id uk cdsk mdsk #i.
      not( Ex #r . UpdateKeyReveal( uk ) @ #r )
    & not( Ex #r . CDSKReveal( cdsk ) @ #r )
    & not( Ex #r . MDSKReveal( mdsk ) @ #r )
    & Sourced_SKs( id, uk, cdsk, mdsk, 'Initial', 'usb_stick' ) @ #i
    ==> not(( Ex #j . K( cdsk ) @ #j ) | ( Ex #j . K( mdsk ) @ #j ))
    )
& (All id uk ak cdsk mdsk #i.
      not(Ex #r. AuthorityKeyReveal( ak ) @ r )
    & not( Ex #r . UpdateKeyReveal( uk ) @ #r )
    & not( Ex #r . CDSKReveal( cdsk ) @ #r )
    & not( Ex #r . MDSKReveal( mdsk ) @ #r )
    & Sourced_SKs( id, uk, cdsk, mdsk, 'Symmetric', ak ) @ #i
    ==> not(( Ex #j . K( cdsk ) @ #j ) | ( Ex #j . K( mdsk ) @ #j ))
    )
& (All id uk cdsk mdsk oprk #i.
      not(Ex #r. OutstationPrivateKeyReveal( oprk ) @ r )
    & not( Ex #r . UpdateKeyReveal( uk ) @ #r )
    & not( Ex #r . CDSKReveal( cdsk ) @ #r )
    & not( Ex #r . MDSKReveal( mdsk ) @ #r )
    & Sourced_SKs( id, uk, cdsk, mdsk, 'Asymmetric', oprk ) @ #i
    ==> not(( Ex #j . K( cdsk ) @ #j ) | ( Ex #j . K( mdsk ) @ #j ))
    )
"
```

This lemma is again a conjunction of three properties; each of these three sections must be true for the overall lemma to be true. We describe each of the three parts of the *Session Key* secrecy lemma in turn:

- Firstly, for when the current *Update Key* was distributed via the 'Initial' distribution method (i.e. USB stick): if the outstation accepts the Session Keys stated a `Sourced_SKs` Action Fact, then, assuming that the Update Key and the new Session Keys themselves haven't been directly revealed to the adversary in this trace, this implies that there is no time point `#j` in this trace when the adversary knows either of the new session keys.
- Secondly, for when the current update key was distributed via the *Symmetric Update Key Change Protocol*: if the outstation accepts the new session keys named in the `Sourced_SKs` action fact, then, assuming that the Authority Key ('`ak`') used to transmit the current update key, the Update Key, and the Session Keys themselves weren't compromised by (or revealed to) the adversary in this trace, then this implies that there is no time point `#j` in this trace when the adversary knows either of the new session keys.
- Finally, for when the update key was distributed via the *Asymmetric Update Key Change Protocol*: if the outstation accepts the new session keys named in the `Sourced_SKs` action fact, then, assuming that the private key ('`oprk`') of the private / public key-pair used to transmit the current Update Key, the Update Key, and the Session Keys themselves weren't compromised by (or revealed to) the adversary in this trace, then this implies that there is no time point `#j` in this trace when the adversary knows either of the new session keys.

We then prove AUTH1 and AUTH2 for both of these protocols, before proving it for the *Critical ASDU Authentication Protocol*. First, the agreement lemma for the *Update Key Change Protocol*:

```
lemma update_key_agreement:
"(All id id2 uk ak ak2 update_key_method #i #j.
          not(Ex #r. AuthorityKeyReveal( ak ) @ #r & #r < #i )
        & not(Ex #r. UpdateKeyReveal( uk ) @ #r & #r < #i )
        & Sourced_UpdateKey( id, uk, 'Symmetric', ak ) @ #i
        & NewUpdateKey( id2, uk, update_key_method, ak2 ) @ #j & #j < #i
        ==> ( id = id2 ) & ( ak = ak2 )
          & ( update_key_method = 'Symmetric' )
    )
& (All id id2 uk oprk oprk2 uprk uprk2 update_key_method #i #j.
         not(Ex #r. UserPrivateKeyReveal( uprk ) @ #r & #r < #i )
        & Sourced_UpdateKey( id, uk, 'Asymmetric',
                                        < oprk, uprk > ) @ #i
        & NewUpdateKey( id2, uk, update_key_method,
                                  < oprk2, uprk2 > ) @ #j & #j < #i
        ==> ( id = id2 ) & ( oprk = oprk2 )
          & ( uprk = uprk2 ) & ( update_key_method = 'Asymmetric' )
        )
"
```

As the initial UK_0(USR,O) is pre-distributed by USB key, we do not need to prove agreement on it (and its secrecy has already been proven by the lemma `update_key_secrecy`). This lemma models:

- For all '`Sourced_UpdateKey`' events for an update key encrypted using *symmetric* cryptography (under '`ak`'), where neither the authority key ('`ak`') used to encrypt it, or the update key itself ('`uk`') is revealed directly by the adversary, where there was a `NewUpdateKey` action fact (before the '`Sourced_UpdateKey`') with the same update key, they must agree on id, mode of encryption used to transmit the update key, and the authority key used to encrypt it.
- For all '`Sourced_UpdateKey`' events for a new update key encrypted using *asymmetric* cryptography, and the secret signing key '`uprk`' isn't revealed directly, where there was a

NewUpdateKey action fact (before the 'Sourced_UpdateKey') with the same update key, they must agree on id, mode of encryption used to transmit the update key, and the user's private key. N.B. This property does NOT require secrecy of the new Update Key for agreement to hold, making it strictly stronger than the symmetric property above.

Note we have also proven (before this lemma, in the lemma update_key_sourced) that for each 'Sourced_UpdateKey' action fact there exists at least one NewUpdateKey action fact which agrees on the same attributes. This second property (above) is now stronger, saying that for *all* pairs of NewUpdateKey and Sourced_UpdateKey with the same value 'uk', they must agree on all other listed attributes as well.

We then prove agreement on the *Session Key Update Protocol*:

```
lemma skiup_agreement:
"(All id id2 uk uk2 cdsk mdsk type source type2 source2 #i #j.
      not( Ex #r . UpdateKeyReveal( uk ) @ #r & #r < #i)
    & ( not(type = 'Asymmetric')
      | not(Ex #r. OutstationPrivateKeyReveal(source) @ #r))
    & ( not(type = 'Symmetric')
      | not(Ex #r. AuthorityKeyReveal(source) @ #r))
    & Sourced_SKs( id, uk, cdsk, mdsk, type, source ) @ #i
    & NewSKs( id2, uk2, cdsk, mdsk, type2, source2 ) @ #j & #j < #i
    ==> ( id = id2 ) & ( uk = uk2 )
      & (type = type2) & (source = source2)
)"
```

This lemma models that for all traces where the update key used to transmit the session keys is not revealed, the session keys themselves are not revealed, and either the outstation's private key or the authority key (depending on the method by which the update key was originally encrypted) weren't revealed, then where there was a Sourced_SKs action fact and a NewSKs action fact before this with the same session keys ('cdsk' and 'mdsk'), they must agree on id, update key, method of update key encryption ('type'), and authority key used to encrypt the update key ('source').

```
lemma asdu_agreement_implies_mode_agreement:
" not(Ex ak #r. AuthorityKeyReveal( ak ) @ #r )
& not(Ex oprk #r. OutstationPrivateKeyReveal( oprk ) @ #r )
& not(Ex uprk #r. UserPrivateKeyReveal( uprk ) @ #r )
==>
( All linkid ar mode direction linkid2 mode2 direction2 #i #j.
    ( All cdsk mdsk uk type source.
        UsingSessKeys( cdsk, mdsk, uk, type, source ) @ #i
    ==>
        not( Ex #kr. UpdateKeyReveal( uk ) @ #kr & #kr < #i )
        & ( direction = 'control' ==>
            not( Ex #skr. CDSKReveal( cdsk ) @ skr & #skr < #i ) )
        & ( direction = 'monitor' ==>
            not( Ex #skr. MDSKReveal( mdsk ) @ skr & #skr < #i ) ) )
    & AuthASDU( linkid, ar, mode, direction ) @ #i
    & SentASDU( linkid2, ar, mode2, direction2 ) @ #j & #j < #i
    ==> ( mode = mode2 )
      & ( direction = direction2 )
      & ( linkid = linkid2 )
)"
```

This lemma only considers traces where there were no Authority Key Reveal actions, and no private keys revealed. Then, for all traces where an ASDU was authorised (in the rule containing the AuthASDU action fact), using the session keys named in UsingSessKeys(...) (distributed by the named update key 'uk', which cannot have been revealed, and itself was distributed via the method in 'type', with the keys in 'source'), and where the session keys for the direction of the received message (i.e. 'control' or 'monitor') were not revealed, and where there was a SentASDU action fact with the same message

'ar' before the AuthASDU action fact, then they these two action facts must agree on 'linkid' (the internal ID of the communications channel), 'mode' (aggressive or non-aggressive), and 'direction' (control or monitor).

Informally, this models that all authorised ASDUs cannot have been originally transmitted over a different link, in a different direction, or in a different direction; this as yet makes no claim about injectivity (or whether it was replayed over the same link, in the same direction and mode). It is worth nothing that while this proves agreement, it doesn't yet prove injective agreement.

In the asdu_aliveness lemma we prove in almost exactly the same way that, with the same above conditions, wherever there is an AuthASDU action fact, there must exist at least one SentASDU action fact with the same message, linkid, mode, and direction.

Then, finally, we prove injective agreement in the asdu_injective_agreement lemma:

```
lemma asdu_injective_agreement:
"not( Ex ak #r. AuthorityKeyReveal(ak) @ #r )
& not(Ex oprk #r. OutstationPrivateKeyReveal( oprk ) @ #r )
& not(Ex uprk #r. UserPrivateKeyReveal( uprk ) @ #r )
==>
( All linkid ar mode direction #i #j.
    ( All cdsk mdsk uk type source.
        UsingSessKeys( cdsk, mdsk, uk, type, source ) @ #i
    ==>
        ( All uk #k. UpdateKeyUsedForSKs( linkid, uk,
                                cdsk, mdsk, type, source ) @ #k
            ==> not( Ex #kr. UpdateKeyReveal( uk ) @ #kr & #kr < #i ) )
        & ( direction = 'control' ==>
            not( Ex #skr. CDSKReveal( cdsk ) @ #skr & #skr < #i ) )
        & ( direction = 'monitor' ==>
            not( Ex #skr. MDSKReveal( mdsk ) @ #skr & #skr < #i ) ) )
    & AuthASDU( linkid, ar, mode, direction ) @ #i
    & SentASDU( linkid, ar, mode, direction ) @ #j & #j < #i
    ==> not( Ex #k. AuthASDU( linkid, ar, mode, direction ) @ #k
      & not( #k = #i ) )
)"
```

In this lemma, we prove (again, in almost exactly the same way) that with the same above conditions, whenever there is a AuthASDU action fact with matching SentASDU action fact and associated terms, then there does not exist another (different) AuthASDU action fact at any point in the trace with the same linkid, message, mode, and direction. We can be confident that any honestly transmitted ASDU will have an incremented counter (and challenge data), and so any actor receiving a message with these same terms can be confident it is a replay, and refuse to authorise it. Proving this series of lemmas now demonstrates that the protocol upholds this property of injective agreement on authorised ASDUs.

4.2. Analysis in TAMARIN

TAMARIN makes use of backwards reasoning, starting from trace constraints that correspond to the negation of the specified property, and building up further constraints from the possible solutions to an open proof goal. This has the invariant that all complete traces that fulfil the original constraints also fulfil at least one of the new sets of constraints. For example, if the current state contains a rule with an unsolved premise fact, then when TAMARIN solves this premise it splits the current state into several states, each containing one of the possible conclusions which may have been the source of that fact.

For example, to prove that a particular property holds in all traces (such as "In all traces, X is preceded by Y"), TAMARIN begins with the trace constraints from its negation ("There exists a trace in which X is not preceded by Y"). Goals are solved until either there is a case with no goals remaining, which is a completed trace and thus a counter-example to the property, or all possible states for this transition system

contradict the constraints. In the latter case, this returns a proof that no trace of the transition system can satisfy the constraints of the negated property, and thus the property holds in all traces.

This backwards reasoning makes TAMARIN very efficient in many protocols, but is ill-suited to a naïve model of the SAv5 protocol. The specification relies not only on shared state between each constituent sub-protocol, but also a shared state machine which dictates which transitions are allowable at particular times. Further, the majority of state transitions occur from and return to the same state, `Security Idle`.

The consequence is that the SAv5 model contains many unbounded loops. Naïvely, an attempt by TAMARIN to solve a premise requiring the `Security Idle` state may find that many rules are potential sources for the preceding state, and attempt to solve each of these possibilities separately. Worse, many may introduce new unsolved premises that also require the `Security Idle` state, creating a loop. One can see how this might cause TAMARIN to indefinitely unroll loops backwards if it cannot already conclude that the constraints cannot be met. Within TAMARIN's framework, this issue can be countered by making use of its forward inductive reasoning and proving properties that resemble loop invariants. For example, TAMARIN can prove inductively that each instance of a *Critical ASDU Authentication Protocol* rule of a particular role must have been preceded by an instance of the *Session Key Update Protocol* rule for the same role. After it proves such invariants inductively, TAMARIN can use them in the backwards search to avoid unrolling loops and instead reason about their starting point immediately.

More technically, the key to analysing a protocol like this is to identify invariants over particular transitions and prioritize solving for the origin point of these terms as necessary. For example, an outstation running the *Critical ASDU Authentication Protocol* is making use of session keys that were set during the last *Session Key Update Protocol* (rule S4, as labelled in Figure 2) and are invariant in all other rules. We therefore add a premise to any rule making use of the session keys so that it directly relies on the current "session key invariant", represented by a persistent fact that is output when the session keys are changed, along with a fresh identifier so that it cannot unify to any other session key invariant. In solving the premises, we can prioritize finding the origin point of the current invariants, as the properties of the current protocol often depend only on the circumstances around the relevant invariants.

In the *Critical ASDU Authentication Protocol* example, the authentication properties depend on the properties of the last *Session Key Update* and the original pairing of the user to outstation, and in the Aggressive Mode, on the last generated challenge data. Each of these is included as an invariant. When proving that all traces have the AUTH1 property, this allows TAMARIN immediately to solve for the origin of the invariants, which adds constraints to, for example, where the session keys were generated and assigned.

As described, the key to analysing a protocol like this successfully is to identify invariants over certain transitions, and to prioritize solving for the source of these. Invariants within the model include:

(1) the authority key and relevant identifiers for both the user and outstation as assigned during the initial key distribution,

(2) update key invariants for both the user and outstation,

(3) session key invariants for both the user and outstation,

(4) and the last challenge sent or received in each direction for both the user and the outstation.

Additionally, there is a 'last key status message' which is stored by the outstation in both the S2 and S4 rules. Although this is invariant through all other rules, it is only used in rules where it is also modified, so we can efficiently represent it with a linear fact consumed and output by those two rules.

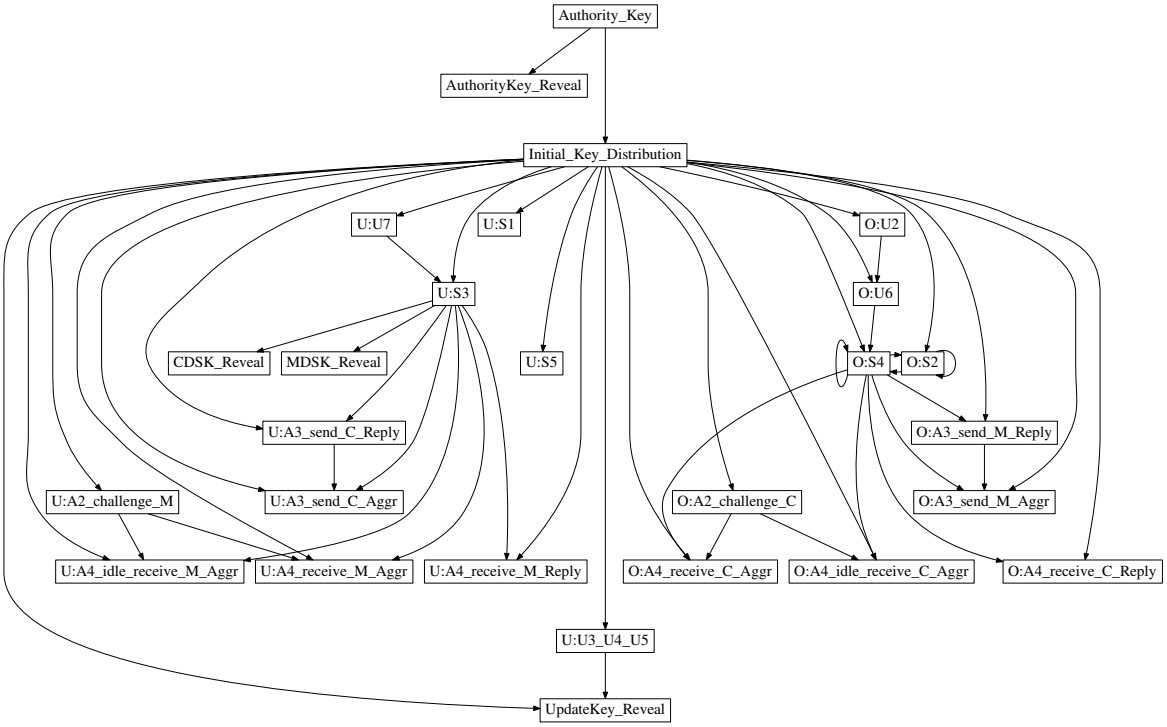

Fig. 7. Protocol rules and the structure of loop invariants in the DNP3 model. Rules executed by the user and outstation are prefixed by 'U:' and 'O:' respectively, and invariants are represented by edges from the rules that set their value to the rules that use them. For example, the U:A3_send_C_Aggressive rule uses the CDSK invariant from the U:S3 rule, the challenge data invariant since the last U:A3_send_C_Reply rule, and the identifiers set up in the Initial_Key_Distribution rule.

Finally, there are three 'keys to reveal' facts output whenever new keys are generated, which are used to model adversary compromise and represented with persistent facts. The combined invariant relations are shown in Figure 7.

4.2.1. Asymmetric Mode Analysis

Adding the Asymmetric mode of the *Update Key Change Protocol* to the overall protocol models had significant impact on the proof burden and time required to prove the required properties of the models. As we detail in Section 4.3, adding one extra sub-protocol (the asymmetric variant of the *Update Key Change Protocol*) causes nearly a 4.5-fold increase in the total CPU time required to prove these same properties.

The main cause of the increased CPU time is that all proofs now need to consider many more case distinctions and possible sources for obtaining messages. Since there is no compositional reasoning that can be directly applied here, and the additional sub-protocol can possibly precede most of the rule instances, the additional rules for the fourth sub-protocol exponentially increase the proof search space. Concretely, this meant we had to construct more complex invariants before TAMARIN could automatically prove the both the previous and newly added (i.e., specific to the asymmetric mode) security properties.

Rather than simply having to prove new properties just about the new asymmetric sub-protocol, we have additionally had to modify and update almost every lemma to take account of the fact that the

asymmetric sub-protocol is now modelled. Proving properties about the secrecy of the update key now requires considering three possible methods of distribution rather than the previous two (i.e. USB Stick, Symmetric, and Asymmetric). Then, as session keys set by the *Session Key Update Protocol* could be compromised *indirectly* by compromise of the update key, or even compromise of the keys used by the *Update Key Change Protocol* (i.e. either the authority key or the user and/or outstation's private keys), properties about the *Session Key Update Protocol* must now also consider each possible method by which the update key could have been set or potentially compromised. This extra burden of proof on the origin of all keys used to encrypt or authenticate messages creates significantly more work and complexity for each lemma.

4.3. Results

Section 4.1 described how the specification requires the protocol be resilient to Spoofing, Modification, Replay, and Eavesdropping, and how these properties translated into more formal security properties AUTH1, AUTH2, and CONF. Our analysis in TAMARIN has formally verified all three of these properties for our model of DNP3: Secure Authentication v5; in particular, they hold for any (unbounded) number of sessions and loop iterations. These results can be automatically verified by TAMARIN from the model and properties in `dnp3.m4`, which can be found at [11]. On a modern PC (2.8 GHz Intel Core i7 from 2014 with 16GB RAM), these theorems in total prove in ∼9m. We additionally proved several sanity checking properties, e.g., to show that our model correctly allows for expected behaviours.

It is worth noting the significant extra computational effort required to prove these properties when introducing even only one extra sub-protocol. The results from [10] proved the same properties on DNP3: SAv5 with only the symmetric mode of encryption available for the *Update Key Change Protocol*; introducing only one extra sub-protocol (the asymmetric variant of the *Update Key Change Protocol*) causes nearly a 4.5-fold increase in the total CPU time required to prove these same properties.

Table 1 details the time taken to achieve these results, and Table 2 details the results of each security property.

Model	**CPU Time**	**Wall clock time**
Symmetric only, from [10]	420.55s (∼7m 00s)	123s (2m 03s)
Asymmetric *and* Symmetric	1846.53s (∼30m 46s)	550s (9m 10s)

Table 1
Times for proving lemmas within the symmetric-only and combined models

Security Property	**Result**
AUTH1	verified
AUTH2	verified
CONF	verified

Table 2
Results of verification of security properties

As stated in the introduction, our results seemingly contradict an attack claimed in previous analysis; we will return to this in detail in Section 6.

5. Recommendations

Our analysis, while succesful in showing that the main properties hold, also naturally leads to several recommendations. To aid clarity of implementation, to avoid possible misinterpretation, and to allow the protocol to meet stronger security guarantees, we propose the following changes to future versions of the specification.

Recommendations Based upon Modelling and Analysis:

- Update Key Change messages (`g120v13`) should contain a clear indication of intended recipient (i.e. outstation ID). This would allow for a stronger authentication property that only relies on the secrecy of the Authority key, not additionally on the secrecy of the new update key.
  In the *Update Key Change Protocol*, the Update Key Change object (`g120v13`) contains the KSQ, User Name, update key, and outstation Challenge Data, but not an outstation identifier (in contrast to the asymmetric version in `g120v14`). Thus, an outstation cannot ensure the Authority agrees on the outstation identity when receiving a newly encrypted update key. It is only through the HMAC in the Update Key Confirmation message (`g120v15`) that the outstation can authenticate the destination of the update key, but this HMAC is computed under that same new update key being distributed. Concretely, there is potential to attack the *Update Key Change Protocol* without knowledge of the Authority's key using only knowledge of the new update key. The adversary can present a challenge from outstation *A* to the user as if it were from outstation *B*, receive an Update Key Change object intended for outstation *B* encrypted under the Authority key, and re-compute the Update Key Confirmation message so that it is incorrectly accepted by outstation *A*.
  This has only minor impact, as the update keys are assumed to be secret, and the attack requires two outstations to be running the *Update Key Change Protocol* with the same user concurrently. Nonetheless, it implies achieving agreement on a new update key requires a weaker adversary than is strictly necessary.
- The specification must clarify the use of Challenge Sequence Numbers:
  - It is not clear whether CSQ values (per direction) should be kept on a per Master-Outstation pair basis, or whether each device should keep one universal CSQ value (per direction).
  - The specification must clarify whether recipients of CSQ values from the network (whether Responder or Challenger) should expect CSQ values to be strictly increasing. The sender's behaviour (whether in an Authentication Challenge, Authentication Reply, or Aggressive Mode Request) is clear, but it is not clear under which conditions a device should accept a CSQ as valid from another party. If CSQ values are not required to be strictly increasing, then replay attacks of Aggressive Mode Requests become possible.
  - Further discussion and reasoning about the use of CSQs may be found in Section 3.3.2.

Recommendations Based upon Best Cryptographic Practice:

- The specification should strongly recommend (or even require) that devices support asymmetric authenticated key exchange, rather just than symmetric key-transport with an optional asymmetric key-transport mode for the *Update Key Change Protocol*. This should be recommended for **both** the *Update Key Change* and *Session Key Update* Protocols. Use of Elliptic Curve Cryptography (ECC) would allow stations to benefit from the added security of asymmetric cryptography, without significantly increasing the total amount of data transmitted. Asymmetric cryptography crucially only requires each private key to be in one location, and ECC is viable on low-power devices [20].
- Deprecate HMAC-SHA-1. The SHA-1 algorithm is dangerously weak, and a collision has been found [21]. HMAC-SHA-256 should be required at minimum.

- Within both the symmetric and asymmetric modes, the protocol should perform some form of key-exchange (incorporating randomness from all involved components), rather than (a)symmetric key-transport [22]. This would significantly reduce the protocol's dependence on the raw output of any one CSPRNG [23].

Other Recommendations:

- The standard must clarify how recipients of messages should parse them, and the standard must clearly and precisely state how recipients should calculate HMACs (e.g. to compare to received Authentication Replies and Aggressive Mode Requests). This must clarify which Sequence Numbers (for both Challenges and Key Changes) should be valid under which conditions, and which Challenge Data should be valid in which situations.
- The standard must clearly state when various data should be kept until (e.g. Challenge Data), when it should be overwritten, and how many previous instances of this data should be kept per User-Outstation pair.

6. Related Work

Previous work has considered the broader security of DNP3, or, in contrast, only analysed SAv5's *Critical ASDU Authentication Protocol* in isolation.

East et al. 2009 provide an interesting and thorough taxonomy of the different types of attack against DNP3 in [24], but as this paper was published before SAv5 was standardised, it does not consider Secure Authentication.

Tawde et al. 2015 propose a 'bump-in-the-wire' solution for the key-management and encryption of critical packets within IEC/TS 62351-5 (the protocol suite upon which DNP3: SAv5 is based), but provide no formal analysis of this addition or the existing protocols [25].

Attacks Claimed: Amoah et al., 2014 & 2016 use Colored Petri-Nets to model and analyse both the non-aggressive and aggressive modes of this sub-protocol, discovering a denial of service attack in the non-aggressive mode [26], and a "replay attack" when the aggressive and non-aggressive modes are combined [27]. Both papers only consider the *Critical ASDU Authentication Protocol* in isolation.

According to [27, p.353], the attack works as follows: after a non-aggressive critical ASDU request (A1 in Figure 3), the attacker blocks the Authentication Challenge message (A2) to the user, and sends a new one with the same challenge data, *but with an artificially incremented CSQ*. The user creates an Authentication Reply (A3, containing an HMAC) with this incremented CSQ value, which the outstation now rejects (A4). The attacker then replays this Authentication Reply with the critical ASDU prepended, to match the format of an Aggressive Mode Request (without modifying the HMAC), which, they claim, the outstation will now accept: valid Aggresive Mode Requests should have both the same challenge data as the last sent Authentication Challenge message, and a CSQ value incremented for each request sent since that challenge. As the user never sent an Aggressive Mode Request (only a non-aggressive request), [27] claims this violates agreement.

This attack does not work, as an outstation will not accept a non-aggressive mode message replayed into the Aggressive Mode. Our reasoning is as follows: HMACs within an Aggressive Mode Request must be calculated over "The entire Application Layer fragment that this object is included in, including the Application Layer header, all objects preceding this one, and the object header and object prefix for this object" [2, p.742, Table A-9]. An Aggressive Mode HMAC must therefore include the "Object Header `g120v3` Authentication Aggressive Mode Request", and the "Object Header `g120v9` Authentication

MAC"; these two object headers must both be included in the HMAC calculation [2, A.45.9, p.741]. In contrast, the calculation of an HMAC within an Authentication Reply message (`g120v2`) from a *non-aggressive mode request* contains no such Aggressive Mode objects or headers. Assuming the attacker cannot successfully modify the HMAC without access to the session key, an HMAC for an Aggressive Mode Request will never match one calculated from the non-aggressive mode, regardless of whether the CSQ values and challenge data match.

We modelled this 'attack' in the file `dnp3-aggressive-amoah-attack.spthy`. For this to succeed, we had to under-approximate the original model significantly compared to the specification. Notably, in this model, we had to remove anything from the specification stating or implying the mode in both HMACs, as well as removing checks on the relationship between the CSQ in the body of the Aggressive Mode Request, and the CSQ within the Authentication Challenge included in the HMAC [2, pp.211 & 742].

We conclude that this claimed attack is an artefact of a model that is too coarse, and is not possible in faithful implementations of the standard.

After the conference version of this work was accepted for publication [10], we contacted the authors of [27] (Amoah, Çamtepe, and Foo) with our discovery, asking for comment or clarification. Amoah and Foo both replied to our email confirming that they did not model the HMAC correctly, and that therefore "the previously reported replay attack identified on the non-aggressive to the aggressive mode of operation will not be possible".

Separately, Amoah et al. then make the novel contribution of a method for *Critical ASDU Authentication* within the Broadcast or Unicast setting, in [28]. Amoah's 2016 thesis [29] supplements these papers by providing greater detail of the modelling and analysis of the *Critical ASDU Authentication Protocol*.

7. Conclusions

In this research we have performed the most comprehensive symbolic modelling and analysis yet of the DNP3 Secure Authentication v5 protocol; this analysis has considered all of the constituent sub-protocols (both symmetric and asymmetric), including cross-protocol and cross-mode attacks. We make use of novel modelling techniques in TAMARIN, by identifying invariants in DNP3's state transitions to cope with analysis of the protocol's inherent complexity, extensive state, and unbounded loops and sessions.

Our findings notably contradict claimed results by earlier analyses; in particular, our findings show that the attack claimed in [27] is not possible in the standard as defined. While our analysis naturally leads to a number of recommendations for improving future versions of DNP3, we conclude that the core protocol of the standard meets its stated security goals if implemented correctly, increasing much-needed confidence in this security-critical building block of power grids.

References

[1] DNP Users Group, A DNP3 Protocol Primer (Revision A), 2005, https://www.dnp.org/AboutUs/DNP3%20Primer%20Rev%20A.pdf (Accessed April 2017).

[2] IEEE, 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3), *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)* (2012), 1–821, http://ieeexplore.ieee.org/document/6327578/.

[3] IEC, IEC/TS 62351-5:2013, Power systems management and associated information exchange – Data and communications security – Part 5: Security for IEC 60870-5 and derivatives, *International Electrotechnical Commission* (2013).

[4] J. Kelsey, B. Schneier and D.A. Wagner, Protocol Interactions and the Chosen Protocol Attack, in: *Security Protocols, 5th Workshop*, 1997, pp. 91–104.

[5] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti and P. Strub, Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS, in: *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 98–113.
[6] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov and B. Preneel, A cross-protocol attack on the TLS protocol, in: *ACM CCS'12*, 2012, pp. 62–72.
[7] J.P. Degabriele, V. Fehr, M. Fischlin, T. Gagliardoni, F. Günther, G.A. Marson, A. Mittelbach and K.G. Paterson, Unpicking PLAID - A Cryptographic Analysis of an ISO-Standards-Track Authentication Protocol, in: *Security Standardisation Research - First International Conference, SSR 2014*, 2014, pp. 1–25.
[8] K.G. Paterson and T. van der Merwe, Reactive and Proactive Standardisation of TLS, in: *Security Standardisation Research*, 2016, pp. 160–186.
[9] D.A. Basin, C. Cremers, K. Miyazaki, S. Radomirovic and D. Watanabe, Improving the Security of Cryptographic Protocol Standards, *IEEE Security & Privacy* **13**(3) (2015), 24–31.
[10] C. Cremers, M. Dehnel-Wild and K. Milner, Secure Authentication in the Grid: A Formal Analysis of DNP3: SAv5, in: *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, S.N. Foley, D. Gollmann and E. Snekkenes, eds, Lecture Notes in Computer Science, Vol. 10492, Springer, 2017, pp. 389–407. ISBN 978-3-319-66401-9. doi:10.1007/978-3-319-66402-6_23. https://doi.org/10.1007/978-3-319-66402-6_23.
[11] C. Cremers, M. Dehnel-Wild and K. Milner, DNP3 Secure Authentication v5 Tamarin Model (with Asymmetric mode of UKCP), 2018, https://github.com/tamarin-prover/tamarin-prover/tree/develop/examples/jcs18/.
[12] IEC, IEC/TS 62351-2:2008, Power systems management and associated information exchange – Data and communications security – Part 2: Glossary of terms, *International Electrotechnical Commission* (2008).
[13] R. Shirey, RFC 2828 – Internet security glossary, 2000, 2000, https://www.ietf.org/rfc/rfc2828.txt (Accessed April 2017).
[14] Alliance for Telecommunications Industry Solutions, Glossary, http://www.atis.org/glossary/definition.aspx?id=3961 (Accessed April 2017).
[15] ISO/IEC, ISO/IEC 9798-1:1997, Part 1: General, 1997, https://www.iso.org/standard/27743.html (Accessed April 2017).
[16] S. Meier, B. Schmidt, C. Cremers and D. Basin, The TAMARIN Prover for the Symbolic Analysis of Security Protocols, in: *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, Springer-Verlag, 2013, pp. 696–701. ISBN 978-3-642-39798-1. doi:10.1007/978-3-642-39799-8_48. http://dx.doi.org/10.1007/978-3-642-39799-8_48.
[17] D. Basin, C. Cremers, J. Dreier, S. Meier, S. Radomirovic, R. Sasse, L. Schmid and B. Schmidt, The Tamarin Prover Manual, 2016, https://tamarin-prover.github.io/manual/book/001_introduction.html, Creative Commons: Attribution-NonCommercial-ShareAlike 4.0 International License.
[18] D. Dolev and A.C. Yao, On the Security of Public Key Protocols, *Information Theory, IEEE Transactions on* **29**(2) (1983), 198–208.
[19] G. Lowe, A Hierarchy of Authentication Specifications, in: *Proceedings 10th Computer Security Foundations Workshop*, 1997, pp. 31–43, ISSN 1063-6900.
[20] N. Gura, A. Patel, A. Wander, H. Eberle and S.C. Shantz, Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs, in: *CHES 2004*, 2004, pp. 119–132.
[21] M. Stevens, E. Bursztein, P. Karpman, A. Albertini and et al., Announcing the first SHA1 collision, 2017, https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html (Accessed April 2017).
[22] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, Information Security and Cryptography, Springer, 2003, https://doi.org/10.1007/978-3-662-09527-0. ISBN 978-3-642-07716-6. doi:10.1007/978-3-662-09527-0.
[23] D.J. Bernstein, T. Lange and R. Niederhagen, Dual EC: A Standardized Back Door, 2015, https://eprint.iacr.org/2015/767.pdf.
[24] S. East, J. Butts, M. Papa and S. Shenoi, A Taxonomy of Attacks on the DNP3 Protocol, in: *Critical Infrastructure Protection III - Third Annual IFIP WG 11.10 International Conference on Critical Infrastructure Protection.*, 2009, pp. 67–81.
[25] R. Tawde, A. Nivangune and M. Sankhe, Cyber security in smart grid SCADA automation systems, in: *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 2015, pp. 1–5.
[26] R. Amoah, S. Suriadi, S.A. Çamtepe and E. Foo, Security analysis of the non-aggressive challenge response of the DNP3 protocol using a CPN model, in: *IEEE International Conference on Communications, ICC 2014*, 2014, pp. 827–833.
[27] R. Amoah, S.A. Çamtepe and E. Foo, Formal modelling and analysis of DNP3 secure authentication, *J. Network and Computer Applications* **59** (2016), 345–360.
[28] R. Amoah, S.A. Çamtepe and E. Foo, Securing DNP3 Broadcast Communications in SCADA Systems, *IEEE Trans. Industrial Informatics* **12**(4) (2016), 1474–1485.
[29] R. Amoah, Formal security analysis of the DNP3-Secure Authentication Protocol, PhD thesis, Queensland University of Technology, 2016.