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

Extended Version, 25th June 2017

Cas Cremers, Martin Dehnel-Wild, Kevin Milner

Department of Computer Science, University of Oxford.
{cas.cremers,martin.dehnel-wild,kevin.milner}@cs.ox.ac.uk

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 three sub-protocols. In doing so, we consider the full state machine, 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.

1 Introduction

Most of the world's power grids are are monitored and controlled remotely. In practice, power grids are controlled by transmitting monitoring and control 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 or 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 [13]. 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" [3], is used by most of the world's power grids for communication, and increasingly for other utilities such as water and gas.

Secure Authentication version 5 (SAv5) is a new protocol family within DNP3, and was standardised in 2012 (Chapter 7 of IEEE 1815-2012 [3], based on IEC/TS 62351-5 [4]). 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 three 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 [17], practical attacks that exploit cross-protocol interactions have only been discovered more recently, e.g., [11, 19]. In general, security protocol standards are very hard to get right, e.g. [10, 12, 21].

Contributions In this work, we perform 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.
- We provide the first analysis of the complex combination of the three sub-protocols, 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 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.

Paper Structure 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 in Section 7. We further illustrate modelling issues, choices, and examples in the appendices.

2 The DNP3 Standard

The DNP3 standard [3] 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 [3]. 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. Session keys are distributed and updated using long-term **Authority Keys** and medium-term **Update keys**. These three different keys are used by three sub-protocols: the *Session Key Update* protocol, the *Critical ASDU Authentication* protocol, and the *Update Key Change* 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. These keys are distributed "over a secure 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 ~15 minutes or ~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.

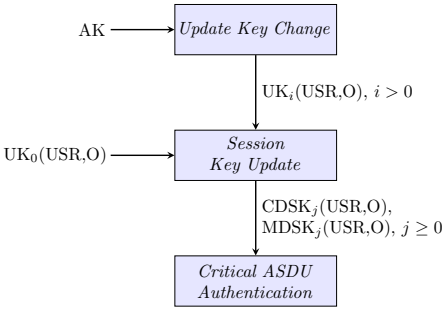

Figure 1. Relationships between sub-protocols, the flow of keys between them (vertical), and required pre-shared keys (horizontal).

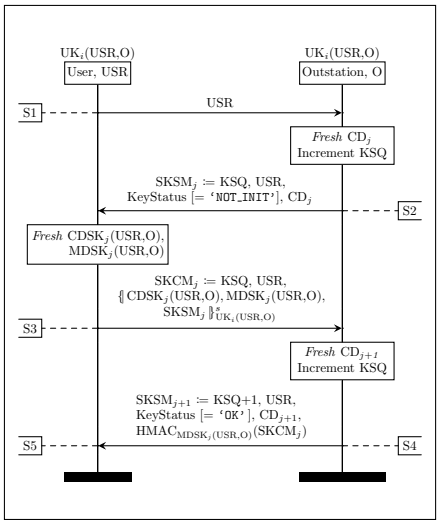

Figure 2. The *Session Key Update Protocol*. The labels S1–5 identify the protocol rules described in Section 2.2.1

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. 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.

2.2 Protocol Descriptions

We now give more detailed descriptions of the three symmetric-key sub-protocols in Secure Authentication v5. We consider the optional asymmetric mode out of scope for this analysis. $\{\!|\, m \,|\!\}^s_k$ denotes the symmetric encryption of term m under key k; similarly $\text{HMAC}_k(m)$ denotes the HMAC of term m keyed by k.

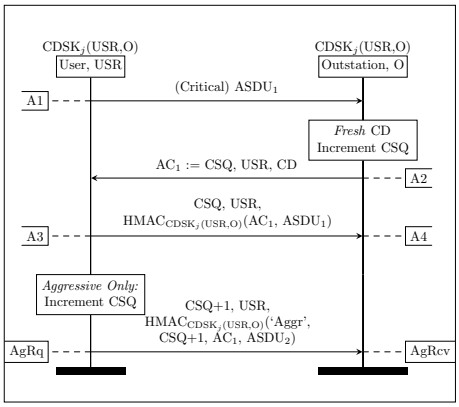

Figure 3. The *Critical ASDU Authentication Protocol*, Control Direction, Non-Aggressive *and* Aggressive Modes. The labels A1–4 identify the protocol rules described in Section 2.2.2

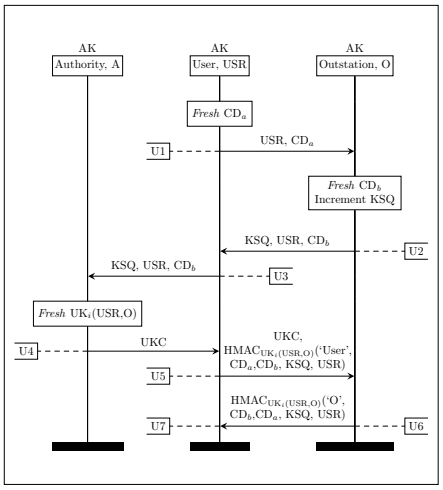

Figure 4. The *Update Key Change Protocol*. The labels U1–7 identify the protocol rules described in Section 2.2.3. In U4 and U5, UKC is the tuple KSQ, USR, $\{\!|\, \text{USR}, \text{UK}_i(\text{USR,O}), \text{CD}_b \,|\!\}^s_{\text{AK}}$

2.2.1 ***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 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 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, 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 ***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, 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, 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 "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 ***Update Key Change Protocol*:** See Figure 4. This allows 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.

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.

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

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".

2.3 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 [3, 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 [3] 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 [2]. This is not the case: [2] 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 [2] as:

> **2.2.191 Spoof** (from IEC/TS 62351-2 [2] p.39)
>
> Pretending to be an authorized user and performing an unauthorized action. [RFC 2828]

While this definition references RFC 2828 [22], there is a difference, in that [22] equates spoofing and masquerading, but does not reference unauthorized actions:

spoofing attack	(from RFC 2828 [22])
(I) A synonym for "masquerade attack".	

where masquerade is defined in the RFC as

masquerade attack	(from RFC 2828 [22])
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 [2], which defines masquerading as

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

Here, ATIS [5] 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 [4] are closer to [5] than to [22], 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 [2] 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 [16], and suggests that replay is a special case of masquerading/spoofing.

Eavesdropping

2.2.92 Eavesdropping	(from IEC/TS 62351-2 [2] 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 [22]. 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 [3] 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 [2] 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" ([3, 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 [20]. 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 Appendix A.1; for more detail on the theory and use of Tamarin see [20] 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.
- Symbolic Representation: information is contained in terms. Any party (including the adversary) can either know a term in its entirety, or not know it, a party cannot learn e.g. a single bit of a term.
- Perfect Cryptography: we assume that the cryptographic primitives used are perfect. This means that e.g. an adversary can only learn the term m from the symmetrically encrypted $\{\!|\, m \,|\!\}^s_k$ term 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).

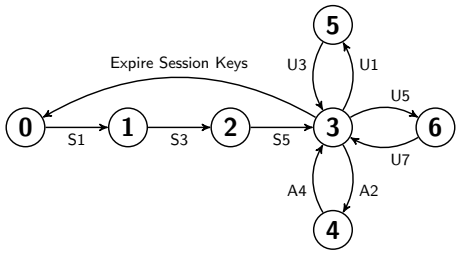

Figure 5. A simplified version of the user's state machine as defined in the standard, excluding error transitions and the monitoring direction of the *Critical ASDU Authentication Protocol*. Note that although many transitions occur from the same state, they are conditional on additional state that is not represented in the state machine as described by the standard.

3.2 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 5 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.

3.3 Protocol Modelling in Tamarin

In Tamarin, protocols are modelled as a collection of labelled multiset rewriting rules; these consist of Premises, Actions (or labels), and Conclusions. The premises of a rule are *facts* which must exist in the multiset prior to the rule's execution, and conclusions are facts which are added to the multiset by executing the rule. Individual facts may be either linear or persistent; if a fact is linear then it is consumed when used in the premise of a rule. Actions are used to label execution traces: when a rule is executed at a particular point, the actions are associated to that time point, and can be referenced to describe properties of traces.

All three sub-protocols' rules and interactions were modelled as rules in Tamarin's operational semantics; the final model comprises 30 multiset rewriting rules in ~450 SLoC. The model and associated theorems are contained in the file `dnp3.m4`, which can be found at [1]. We give an example of a SAv5 rule modelled in Tamarin in Appendix A.1.

The state machines described in [3] (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.

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. [18]). 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 "monitoring"), 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.

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. 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).

We now give an example of a confidentiality property from our analysis; this property models the secrecy of Session Keys from the outstation's point of view:

```
lemma sessionkey_secrecy_outst:
  "not ( Ex AK #r . AuthorityKeyReveal( AK ) @ r )
  ==>
  ( All id UK CDSK MDSK #i.
      not ( Ex #r . UpdateKeyReveal( UK ) @ r )
      & not ( Ex #r . CDSKReveal( CDSK ) @ r )
      & not ( Ex #r . MDSKReveal( MDSK ) @ r )
      & received_sess_keys( id, UK, CDSK, MDSK ) @ i
       ==>  not ( Ex #j . K( CDSK ) @ j ) & not ( Ex #j. K( MDSK ) @ j ) )"
```

Informally this says, "assuming no authority keys have been compromised, if the outstation has received some new un-revealed session keys encrypted under an un-revealed update key, then the adversary cannot derive those new session keys". Most key-secrecy lemmas are of this form.

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.

4.2 Analysis in Tamarin

Tamarin makes use of backwards reasoning, starting from trace constraints, 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.

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 are contradictory. In the latter case, this returns a proof that no trace 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`. Naïvely, an attempt to solve a premise requiring the `Security Idle` state may find that many rules are potential sources, 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.

The key to analysing a protocol like this is to identify invariants over particular transitions and prioritize solving for the source of these 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 the sources 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 source of the invariants, which adds constraints to, for example, where the session keys were generated and assigned.

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 [1]. On a modern PC (2.6 GHz Intel Core i7 from 2012 with 8GB RAM), these theorems in total prove in ~1m 33s. We additionally proved several sanity checking properties, e.g., to show that our model correctly allows for expected behaviours.

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

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. We discuss the reasoning behind these recommendations in more detail in Appendices A.2 and A.3.

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.
- 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.

Recommendations Based upon Best Cryptographic Practice:

- The specification should strongly recommend that devices support asymmetric cryptography, rather just than symmetric key-transport. 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 secret key to be in one location, and ECC is viable on low-power devices [15].
- Deprecate HMAC-SHA-1. The SHA-1 algorithm is dangerously weak, and a collision has been found [23]. HMAC-SHA-256 should be required at minimum.

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 [14], 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 [24].

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 [9], and a "replay attack" when the aggressive and non-aggressive modes are combined [7]. Both papers only consider the Critical ASDU protocol in isolation.

According to [7, 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), [7] 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" [3, 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 [3, 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 [3, 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.

Amoah et al. then make the novel contribution of a method for *Critical ASDU Authentication* within the Broadcast or Unicast setting, in [8]. Amoah's 2016 thesis [6] supplements these papers by providing greater detail of the modelling and analysis of the *Critical ASDU Authentication Protocol*.

7 Conclusions

In this work, 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, including cross protocol 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 [7] 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. DNP3 Secure Authentication v5 Tamarin Model, `https://www.cs.ox.ac.uk/people/cas.cremers/tamarin/dnp3/dnp3.zip`
2. 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)
3. IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3). IEEE Std 1815-2012 pp. 1–821 (Oct 2012)
4. 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)

5. Alliance for Telecommunications Industry Solutions: Glossary, http://www.atis.org/glossary/definition.aspx?id=3961 (Retrieved April 2017)
6. Amoah, R.: Formal security analysis of the DNP3-Secure Authentication Protocol. Ph.D. thesis, Queensland University of Technology (2016)
7. Amoah, R., Çamtepe, S.A., Foo, E.: Formal modelling and analysis of DNP3 secure authentication. J. Network and Computer Applications 59, 345–360 (2016)
8. Amoah, R., Çamtepe, S.A., Foo, E.: Securing DNP3 Broadcast Communications in SCADA Systems. IEEE Trans. Industrial Informatics 12(4), 1474–1485 (2016)
9. Amoah, R., Suriadi, S., Çamtepe, S.A., Foo, E.: Security analysis of the non-aggressive challenge response of the DNP3 protocol using a CPN model. In: IEEE International Conference on Communications, ICC 2014. pp. 827–833 (2014)
10. Basin, D.A., Cremers, C., Miyazaki, K., Radomirovic, S., Watanabe, D.: Improving the Security of Cryptographic Protocol Standards. IEEE Security & Privacy 13(3), 24–31 (2015)
11. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.: Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113 (2014)
12. Degabriele, J.P., Fehr, V., Fischlin, M., Gagliardoni, T., Günther, F., Marson, G.A., Mittelbach, A., Paterson, K.G.: Unpicking PLAID - A Cryptographic Analysis of an ISO-Standards-Track Authentication Protocol. In: Security Standardisation Research - First International Conference, SSR 2014. pp. 1–25 (2014)
13. DNP Users Group: A DNP3 Protocol Primer (Revision A) (2005), https://www.dnp.org/AboutUs/DNP3%20Primer%20Rev%20A.pdf (Retrieved April 2017)
14. East, S., Butts, J., Papa, M., Shenoi, S.: 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. pp. 67–81 (2009)
15. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C.: Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In: CHES 2004. pp. 119–132 (2004)
16. ISO/IEC: ISO/IEC 9798-1:1997, Part 1: General (1997), https://www.iso.org/standard/27743.html (Retrieved April 2017)
17. Kelsey, J., Schneier, B., Wagner, D.A.: Protocol Interactions and the Chosen Protocol Attack. In: Security Protocols, 5th Workshop. pp. 91–104 (1997)
18. Lowe, G.: A Hierarchy of Authentication Specifications. In: Proceedings 10th Computer Security Foundations Workshop. pp. 31–43 (Jun 1997)
19. Mavrogiannopoulos, N., Vercauteren, F., Velichkov, V., Preneel, B.: A cross-protocol attack on the TLS protocol. In: ACM CCS'12. pp. 62–72 (2012)
20. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Computer Aided Verification - 25th International Conference, CAV. pp. 696–701 (2013)
21. Paterson, K.G., van der Merwe, T.: Reactive and Proactive Standardisation of TLS. In: Security Standardisation Research. pp. 160–186 (2016)
22. Shirey, R.: RFC 2828 – Internet security glossary, 2000 (2000), https://www.ietf.org/rfc/rfc2828.txt (Retrieved April 2017)
23. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., et al.: Announcing the first SHA1 collision (2017), https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html (Retrieved April 2017)
24. Tawde, R., Nivangune, A., Sankhe, M.: Cyber security in smart grid SCADA automation systems. In: 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS). pp. 1–5 (2015)

Appendix A

A.1 Tamarin example rule

There are a few special facts used in Tamarin. The Dolev-Yao adversary is modelled through special `In` and `Out` facts, which respectively ask the adversary to provide, or provide to the adversary, the term in the fact. Additionally, there is a `Fr` fact to represent symbolically generating a random value.

Example 1. The following rule is an example of a multiset rewriting protocol rule in Tamarin's syntax. This 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.

In the Tamarin implementation of this rule above, the premises require a linear 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 `LastControlChallenge` used for aggressive mode is overwritten by the new challenge just received, as well as the message to the network `AR` defined above.

A.2 *Update Key Change Protocol* Outstation ID.

Update Key Change messages (`g120v13`) should contain a clear indication of intended recipient (i.e. outstation ID).

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 recompute 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.

A.3 Modelling choices, and issues with the specification

The finer details of SAv5's sub-protocols in [3] and [4] are very often unclear, under-specified, and open to interpretation. We give a couple of indicative examples of 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 [3, 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 [3, p.211-d]. The purpose of the CSQ is to match messages, and to ensure that replays are not possible [3, 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 [3, 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 is to 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 [3] 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 side.

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.