Extending the Authentication Hierarchy with One-Way Agreement

Johannes Wilson
Sectra Communications, Sweden
Dept. of Computer and Information Science
Linköping University, Sweden
Email: johannes.wilson@sectra.com

Mikael Asplund
Dept. of Computer and Information Science
Linköping University, Sweden
Email: mikael.asplund@liu.se

Niklas Johansson
Sectra Communications, Sweden
Dept. of Electrical Engineering
Linköping University, Sweden
Email: niklas.johansson@sectra.com

***Abstract*—Providing authenticated interactions is a key responsibility of most cryptographic protocols. When designing new protocols with strict security requirements it is therefore essential to formally verify that they fulfil appropriate authentication properties. We identify a gap in the case of protocols with unilateral (one-way) authentication, where existing properties are poorly adapted. In existing work, there is a preference for defining strong authentication properties, which is good in many cases but not universally applicable. In this work we make the case for *weaker* authentication properties. In particular, we investigate one-way authentication and extend Lowe's authentication hierarchy with two such properties. We formally prove the relationship between the added and existing properties. Moreover, we demonstrate the usefulness of the added properties in a case study on remote attestation protocols. This work complements earlier work with additional generic properties that support formal verification of a wider set of protocol types.**

I. Introduction

Cryptographic protocols are necessary building blocks of the Internet and the services we take for granted in today's society. We rely on these protocols to uphold basic security properties such as secrecy (confidentiality) and authenticity. It has been argued [1] that formal verification of security protocols should be adopted early on in development to prevent logical design flaws. We support this viewpoint, and that more work is needed to make formal verification attainable for practitioners not trained in formal methods.

We focus on authentication properties, whose definition and meaning varies between different works and can be difficult to map to a particular use-case or scenario for non-experts. Making protocol verification accessible means providing reusable and generic primitives and properties that can be applied to many different protocols. However, authentication is not a one-size-fits-all property. Different protocols will need different levels of authentication depending on the intended goal of each protocol. Using authentication specifications that are precise is important during formal verification, since using a definition that is too weak will miss feasible attacks, while using a definition that is too strong may cause false-positives.

Lowe's authentication hierarchy [2] provides an ordered set of generic properties and has been widely used to verify authentication properties, especially in works using the Tamarin prover [3]. For example, it has been used for verification of 5G authentication [4][5] and Direct Anonymous Attestation [6][7]. The hierarchy provides a powerful comparison between protocols, as it allows reasoning about relative strength of different authentication properties. Unfortunately, Lowe's hierarchy in its current form has some gaps. In particular, it only considers mutual authentication, neglecting the many situations where only one-way authentication is needed. For example, important internet protocols such as TLS will only perform authentication of the server, not the client, when unilateral authentication is chosen (the terms unilateral and one-way authentication are synonymous). A crucial aspect of one-way authentication is that the authenticated agent will not make any distinction between non-authenticated agents.

In this paper we define two one-way authentication properties that complement Lowe's hierarchy where already the one-way dimension differentiates the properties *aliveness* and *weak agreement*, but lacks the corresponding one-way properties for non-injective and injective agreement. We show how the hierarchy can be intuitively extended with the proposed properties and prove that they form a strict hierarchy. We also show that the proposed one-way properties are applicable in real-world scenarios by examining remote attestation as a case-study. We therefore believe that the amended hierarchy would be a useful resource for those who seek to find appropriate verification properties to test their protocols against.

Previous works has often focused on finding the strongest possible definition for authentication [8][9]. Granted, verifying against such strong properties offers a higher confidence that a protocol is secure. However, in practice it is often impractical since not all protocols are designed to prevent every type of adversarial behaviour. Rather, we argue that weaker properties are often needed as these correspond to the actual guarantees provided by the protocol. One-way authentication has been used and discussed in an ad-hoc fashion in previous works (e.g., [10][11]) but not properly defined as generic properties and placed in an authentication hierarchy with associated strictness proofs. Another related strand of work has discussed unilateral key exchange. While key exchange requires authentication, the opposite is not true, and we consider key establishment as out of scope for this paper.

We argue that this work fills an important gap in the literature by formally stating and analysing one-way authentication properties. Proving that these properties strictly differ from

existing properties is important since the apparent strength of a property (based on how it is expressed as a logical formula) is not necessarily reflected by the set of feasible cryptographic protocols that exhibit the property.

We also formally show the suggestion by Lowe that unique and unpredictable nonces can provide both injectivity and recency, thus protecting against replay and preplay attacks. In fact, we introduce a sufficient criterion for protocols fulfilling injective agreement to also fulfil recent injective agreement. This potentially reduces the need of stronger authentication properties such as synchronisation.

The contributions of this paper can be summarized as follows.

- Extending Lowe's authentication hierarchy with two authentication properties according to a two-dimensional categorisation.
- Proving strictness of the extended hierarchy by showing the relationships between the sets of admissible protocols for the different properties.
- Providing a sufficient criterion for recency given an injective protocol.
- Demonstrating the applicability of the proposed properties in a case study based on remote attestation.

The rest of this paper is organised as follows. Section II describes the formalism and notation used, briefly introduces Lowe's hierarchy, and discusses the meaning of strictness for protocol properties. Section III describes the need for additional properties, and introduces our proposed extended hierarchy. Section IV shows a complete proof of the relationships between properties. In Section V, we illustrate that nonces are a sufficient tool to provide recentness given a protocol that is already injective. Section VI is a case study on remote attestation protocols which illustrates cases where our proposed properties may be useful. Section VII discusses related work, and Section VIII presents some conclusions of this work.

II. Preliminaries

In this section we explain notation, and briefly introduce the original hierarchy by Lowe. Finally, we highlight an important distinction between trace properties and protocol properties, which affects reasoning about strictness for protocol properties.

A. Notation and Terminology

We build on Schmidt et al. [12] for the formalisation of protocols and traces. Protocols and adversaries are modelled as a labelled transition system, where labelled multiset rewrite rules describe protocol steps and adversary capabilities. A trace is a sequence of sets of actions (ground facts). For the sake of presentation, we only consider rewrite rules with a single action, so a trace is thus a sequence of actions. We write the set of all traces of a protocol *P* as *traces(P)*.

Properties are specified as trace formulas in a two-sorted first-order logic, with one sort for timepoints and one sort for message contents. Traces may contain free variables, and so a valuation θ is an instantiation of all free variables in a trace T. For a given trace T, a valuation θ and a trace formula φ, we write $(T, \theta) \models \varphi$ to mean that the trace formula is satisfied by the trace with the given valuation. If all possible valuations of T satisfy a trace formula, we write $T \models \varphi$. We say a security property φ holds for a protocol P, written $P \models \varphi$, iff $T \models \varphi$ for every possible trace $T \in traces(P)$.

As an example, consider the trace formula for the authentication property *aliveness* in Formula 1.

$$\begin{aligned} &\varphi_{alive} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i \\ &\quad \rightarrow (\exists\, I_2\, n_2\, j.\, Running(R, I_2, n_2)@j \end{aligned} \tag{1}$$

In Formula 1, *Commit* and *Running* are actions, and i and j are timepoints. We use the notation $f(...)@i$ to signify that an action f occurred at timepoint i. Terms *I*, *R*, *n* and *i* are all examples of variables, and are not bound to any particular value by a valuation. For the formula to hold for a trace it must hold for any valuation.

B. Lowe's Hierarchy

We give a brief introduction to the authentication hierarchy as specified by Lowe. In Lowe's hierarchy, four main authentication properties are introduced: *aliveness*, *weak agreement*, *non-injective agreement* and *(injective) agreement*. Following Lowe's definitions [2], we assume the existence of at least two roles in each protocol, an *initiator* and a *responder*. We further assume that that the final step of the protocol performed by the initiator role is labelled by a *Commit* action, and the final step of the protocol performed by the responder role is labelled by a *Running* action. The actions *Commit* and *Running* have the following form. The first argument describes the name of the acting agent, while the second argument takes the name of the intended recipient. For example, *Commit(I,R,n)* means that *Commit* was performed by *I*, and the expected responder was *R*. The remaining argument contains a term representing data items that are sent as part of the protocol.

Consider again Formula 1 for aliveness shown previously. Intuitively, Formula 1 describes that whenever *Commit* has occurred, then *Running* has occurred at some point, and was performed by the responder *R* expected by the initiator *I*. Excluding the responder term *R*, all other terms used in the action *Running* are existentially quantified, signifying that they do not need to agree with the values seen by the initiator *I* performing *Commit*.

Commit can be seen as a claim made by the initiator. Once *Commit* has occurred, we expect there to have been some *Running* action. Typically, we assume that *Commit* is performed by the initiator of the protocol, but this need not be the case in practice. It is possible to generalise these definitions to other kinds of interactions as well. A symmetric authentication property can be formulated from the perspective of the responder role, or for any other role involved in the protocol. Moreover, the reasoning in this paper can be applied

to a specific protocol phase that considers only a part of the protocol.

Aliveness describes a very weak form of authentication. For the stronger specifications for authentication, the initiator and responder need to both agree on roles and data. The following formulas show formulas for weak agreement, non-injective agreement, and injective agreement, respectively.

$$\begin{aligned}&\varphi_{w-agree} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I,R,n)@i \\ &\quad \rightarrow (\exists\, n_2\, j.\, Running(R,I,n_2)@j)\end{aligned} \tag{2}$$

$$\begin{aligned}&\varphi_{ni-agree} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I,R,n)@i \\ &\quad \rightarrow (\exists\, j.\, Running(R,I,n)@j)\end{aligned} \tag{3}$$

$$\begin{aligned}&\varphi_{i-agree} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I,R,n)@i \\ &\quad \rightarrow (\exists\, j.\, Running(R,I,n)@j \\ &\qquad \wedge \neg(\exists\, I_2\, R_2\, i_2.\, Commit(I_2,R_2,n)@i_2 \\ &\qquad\quad \wedge\, i_2 \neq i))\end{aligned} \tag{4}$$

It can be shown using the formulas that each successive property in the hierarchy necessarily implies the previous, and is thus at least as strong of a requirement. It can also be shown that they are strictly stronger. Given property φ, we define the set of protocols that fulfil each property as $S_\varphi \triangleq \{P : P \models \varphi\}$ where P is a protocol. The following statement, which we state without a proof (follows from Lowe's work), will be used for proving our extended hierarchy.

Assumption 1. *The relations in the hierarchy are strict, meaning that for each pair of properties there is a protocol for which the stronger property does not hold but the weaker does.*
$S_{i-agree} \subset S_{ni-agree} \subset S_{w-agree} \subset S_{alive}$

C. Strictness of Protocol Properties

Showing that trace properties are strict for traces can be done by providing traces that distinguish the properties. However, there is a subtle but important difference between trace properties and protocol properties. A trace property holds for a specific trace, while a protocol property has to hold for every possible trace of the protocol. The original definitions by Lowe contain a temporal requirement: the *Running* action must have occurred before *Commit*. However, it is not necessary to specify as part of the property. Consider the example of a modified formula for injective agreement specified as follows.

$$\begin{aligned}&\varphi_{m-i-agree} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I,R,n)@i \\ &\quad \rightarrow (\exists\, j.\, Running(R,I,n)@j \\ &\qquad \wedge\, j < i \\ &\qquad \wedge \neg(\exists\, I_2\, R_2\, i_2.\, Commit(I_2,R_2,n)@i_2 \\ &\qquad\quad \wedge\, i_2 \neq i))\end{aligned} \tag{5}$$

In Formula 5, the requirement for $j < i$ has been added to Formula 4, meaning we additionally disallow traces where *Running* happened after *Commit*. Trivially $\varphi_{m-i-agree} \Rightarrow \varphi_{i-agree}$. It may seem like the modification gives rise to a different property, since we can easily find a trace that fulfils $\varphi_{i-agree}$ but not $\varphi_{m-i-agree}$ using a trace where *Commit* occurs before *Running*. However, the formulas give rise to the exact same condition for protocols under a Dolev-Yao [13] model where messages can be arbitrarily dropped.

Proof. Assume there is a protocol P so that $P \models \varphi_{i-agree}$ and $P \not\models \varphi_{m-i-agree}$. Then there must exist at least one trace $T \in traces(P)$ so that $T \models \varphi_{i-agree}$ and $T \not\models \varphi_{m-i-agree}$. For this trace T, the only possibility is that there was some *Commit* at a time i and a *Running* at a time j so that $j >= i$. Importantly, there cannot be any matching *Running* actions that occurred before *Commit*, or else the trace would also fulfil $\varphi_{m-i-agree}$.

Then there must be a subtrace T_s consisting of the sequence of actions in T leading up to and including all actions that happened before or at timepoint i, but excluding all actions that happened after. This is because our Dolev-Yao adversary is able to drop all messages sent after this point[1]. Such a trace contains a *Commit* action at i, but no matching *Running* action. Such a trace does not fulfil $\varphi_{i-agree}$. Such a trace must be a valid trace for protocol P, since it only contains states in an order that was possible during trace T. But this is a contradiction, since $P \models \varphi_{i-agree}$. □

The same line of reasoning can be used to justify removal of the temporal condition also from the other properties. This example illustrates that strictness for trace properties does not equate to strictness for protocol properties. In order to show that two protocol properties are distinctly different, we must first show that there exists a protocol fulfilling the first property, but for which there exists a trace which violates the other property.

III. One-Way Authentication

A. Motivation

Because injective agreement is defined as a strictly stronger property relative to weak agreement, proving injectivity using these definitions requires also proving weak agreement. However, weak agreement already requires agreement on initiator

[1]If we allow *Running* and *Commit* to occur at the same time, we could drop the message needed for *Running* but not for *Commit* since we assume they are performed by different agents.

identity. For mutually authenticated key exchange this is reasonable, since we may want to ensure that the key is always established between two known agents before we ensure that old sessions are not replayable. On the other hand, for unilateral key-exchange, or for any type of one-way authentication, it is typically not possible to prove that the authenticated agent knows the identity of the unauthenticated agent, and so weak agreement fails. For such cases, the specifications defined in the hierarchy are not sufficient, as we may still want to show that unilateral authentication holds injective properties in order to prevent replay attacks. Injective agreement is too strong of a property for unilateral authentication in these cases.

Protocols that are unilaterally authenticated can still be considered secure, however only one of the agents can consider the other to be authenticated. This does not in general lead to an attacker-in-the-middle vulnerability, since agreement on data unique to the client ensures to the client that server session was performed with the client, regardless of who the server believed was the intended recipient.

We argue for the relevance of unilateral authentication using TLS as an example. As noted by Cremers et al. during formal verification of a draft of TLS 1.3 [10], the client cannot guarantee that the server agrees on the client's identity during a unilateral handshake. This means that the standard formulation for non-injective agreement could not be used, even though there was agreement on all data. The TLS protocol establishes a session between a server and a client, and allows application level protocols, such as HTTPS, to send data securely using the established session key. All modes in TLS require the server to be authenticated to the client, while it is optional for the client to be authenticated to the server [14].

Fig. 1 shows a simplified TLS 1.3 handshake. The TLS handshake establishes shared symmetric session keys between the server and the client, and additionally authenticates the server using a certificate or pre-shared key. If the client is also able to authenticate using a certificate, then the server can request client authentication. In this scenario the client would send *Certificate* and *CertificateVerify* messages before sending the *Finished* message. The case where client authentication is performed is known as mutual TLS.

Connections between servers and clients are often unilaterally authenticated, as the client can know and authenticate the server, while the server has no prior knowledge of the client identities.

B. Proposed Properties

While the definition of non-injective agreement is strictly stronger than weak agreement, the requirements they relate to do not depend on each other. It is possible to agree on data, while not agreeing on participant identities. This is not reflected in the hierarchy, since a proof of non-injective agreement necessarily proves weak agreement, therefore also requiring agreement on participant identities. This is illustrated in Table I.

Table I shows the authentication properties separated into two columns, mutual and one-way. Mutual properties require

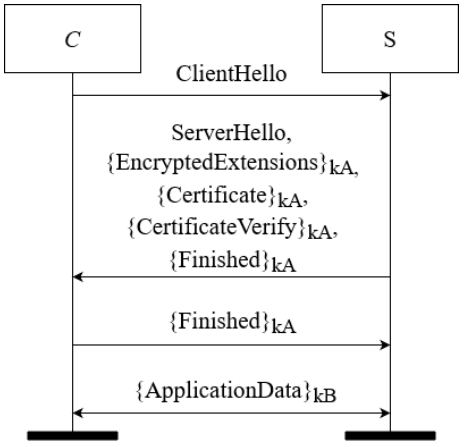

Fig. 1. TLS 1.3 Handshake

TABLE I
AUTHENTICATION HIERARCHY. PROPERTIES DEFINED IN THIS WORK ARE SHOWN IN BLUE.

Surj.	Integ.	Inj.	Mutual		One-Way
•	•	•	Injective Agreement	⇒	One-Way Injective Agreement
			⇓		⇓
•	•		Non-injective Agreement	⇒	One-Way Non-Injective Agreement
			⇓		⇓
•			Weak Agreement	⇒	Aliveness

Surj. = Surjectivity, Integ. = Integrity, Inj. = Injectivity

agreement also on initiator identity, while one-way do not. In the vertical axis we show three supported properties: surjectivity (there is at least one *Running* for each *Commit*), integrity (the two roles have the same view on the shared data) and injectivity (there is at most one *Commit* for each *Running*). Every authentication property at least describes a surjective property from *Running* to *Commit*. In principle, one can also imagine the case where there is injectivity but not integrity, but it is not clear whether such protocols are feasible or practical.

We propose that the hierarchy is extended to include two weaker alternatives to non-injective and injective agreement, called *one-way non-injective agreement* and *one-way injective agreement*. We define one-way non-injective agreement and one-way injective agreement as follows.

a) Definition: one-way non-injective agreement: A protocol guarantees one-way authentication with *non-injective* agreement to an agent *A* acting as the initiator if *A* completes a run of the protocol apparently with *B*, then there was an agent *B* acting as responder *not necessarily with A*, and *A* and *B* agreed on the data for this run.

Formula 6 shows one-way non-injective agreement.

$$\varphi_{ow-ni-agree} \triangleq \forall\, I\, R\, n\, i.\, Commit(I, R, n)@i \rightarrow (\exists\, I_2\, j.\, Running(R, I_2, n)@j) \tag{6}$$

As will be shown, one-way non-injective agreement is stronger than aliveness, but weaker than non-injective agreement. Unlike weak agreement, it does not guarantee that the responder knew the initiator identity during the protocol run.

The entity authentication property used by Cremers et al. during formal verification of TLS 1.3 [10] matches our definition of one-way non-injective agreement. Therefore, their results do not show that replay attacks are prevented. This does not mean that replay attacks against TLS are possible, but rather that the stronger property one-way injective agreement as defined below would be needed to formally prove absence of replay attacks.

Note that the main contributions of this paper are not proposing the properties themselves (as variations have been discussed in the literature previously), but rather extending the authentication hierarchy with well-defined generic properties with strictness proofs between the properties.

b) Definition: one-way injective agreement: A protocol guarantees one-way authentication with *injective* agreement to an agent *A* acting as the initiator if *A* completes a run of the protocol apparently with *B*, then there was an agent *B* acting as responder *not necessarily with A*, and *A* and *B* agreed on the data for this run, *and for each run of A there was a unique run by B*.

Formula 7 shows one-way injective agreement.

$$\begin{aligned} &\varphi_{ow-i-agree} \triangleq \\ &\forall I\, R\, n\, i.\, Commit(I, R, n)@i \\ &\quad \rightarrow (\exists I_2\, j.\, Running(R, I_2, n)@j \\ &\qquad \wedge \neg(\exists I_3\, R_2\, i_2.\, Commit(I_3, R_2, n)@i_2 \\ &\qquad\quad \wedge\ i_2 \neq i)) \end{aligned} \tag{7}$$

One-way injective agreement ensures to the initiator that the responder run was unique and agreed on the data. This is enough to show that a session was unique between the responder and the initiator, however the responder's view of the participating agents may not match what the initiator believes.

IV. Strength and Strictness

We prove that if we extend the hierarchy with the above one-way properties, then each property is distinct and the sets of protocols that fulfil the properties will follow a strict partial order according to Theorem 1. Property sets defined in this work are shown in blue.

Theorem 1.
$$\begin{array}{ccc} S_{i-agree} & \subset & S_{ow-i-agree} \\ \subset & & \subset \\ S_{ni-agree} & \subset & S_{ow-ni-agree} \\ \subset & & \subset \\ S_{w-agree} & \subset & S_{alive} \end{array}$$

The proof strategy for Theorem 1 is given in Fig. 2. The dotted lines represent relations that are already taken to be strict as per Assumption 1. The proof begins by showing in step 1 that the weaker properties hold from the stronger properties. Step 1 of the proof follows directly from the definitions of the properties, and only shows implication between protocol properties. Next, we show that properties are distinctly different along the diagonal in step 2[2]. Finally, in step 3 we prove strictness for the relations for which implication was shown in step 1. Step 3 depends on the previous steps, along with Assumption 1. This proves Theorem 1, and additionally shows that properties are distinct along the diagonal.

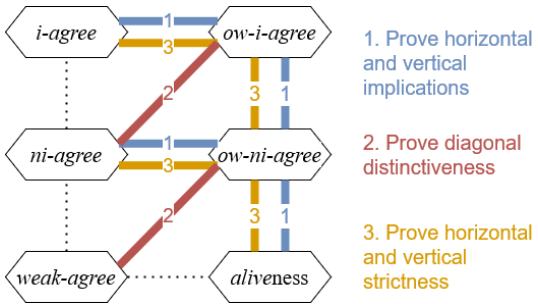

Fig. 2. Proof Strategy

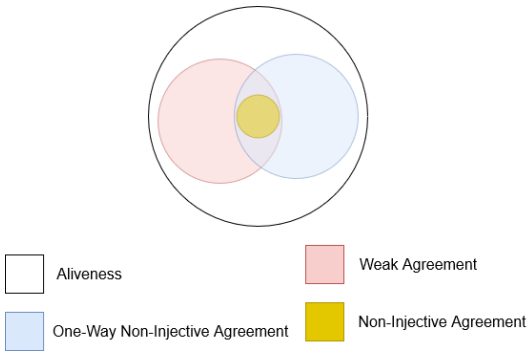

Fig. 3. Euler Diagram of Property Sets

Steps 2 and 3 are needed in order to show that the definitions for one-way authentication properties are distinctly different from any existing authentication property when considering them as protocol properties. There are several cases to consider. Assume that we know implication holds, and we are tasked with proving strictness for the sub-problem shown in Formula 8 (i.e. the lower four properties in the hierarchy).

$$\begin{array}{ccc} S_{ni-agree} & \subseteq & S_{ow-ni-agree} \\ \subset & & \subseteq \\ S_{w-agree} & \subset & S_{alive} \end{array} \tag{8}$$

It is possible that the proposed property $S_{ow-ni-agree}$ is equal to $S_{ni-agree}$, that $S_{ow-ni-agree}$ is equal to S_{alive}, or that $S_{ow-ni-agree}$ is equal to $S_{w-agree}$. It is also possible that one of $S_{ow-ni-agree}$ and $S_{w-agree}$ is a subset of the other. Together, steps 2 and 3 show that none of these cases apply; all subsets are strict, and $S_{ow-ni-agree}$ and $S_{w-agree}$ each contain elements not in the other set. That is, we conclude the set relation shown in Fig. 3.

[2]Distinctly different in this case refers to the sets being incomparable with respect to the subset relation.

Step 2 specifically proves that $S_{ow-ni-agree}$ and $S_{w-agree}$ are distinct and each contain elements not in the other set. Step 3 specifically proves that from Assumption 1 it follows that all other relations must be strict. Using the same method we can then prove the top part of the hierarchy, showing that all relations are strict.

A. Step 1: Proving Horizontal and Vertical Implications

We begin with the first part of the proof by showing where implication holds between the proposed and existing properties in the hierarchy. Clearly, the original properties non-injective and injective agreement are at least as strong as strong as their corresponding one-way properties (proof in Appendix).

Lemma 1. $\varphi_{ni-agree} \Rightarrow \varphi_{ow-ni-agree}$

Lemma 2. $\varphi_{i-agree} \Rightarrow \varphi_{ow-i-agree}$

We can also conclude that one-way injective agreement implies one-way non-injective agreement directly from their formal definition.

Lemma 3. $\varphi_{ow-i-agree} \Rightarrow \varphi_{ow-ni-agree}$

Proof. Trivial. Formula (7) ⇒ (6) □

Finally, one-way non-injective agreement implies aliveness (proof in Appendix).

Lemma 4. $\varphi_{ow-ni-agree} \Rightarrow \varphi_{alive}$

Based on Lemmas 1 – 4 and Assumption 1 it is possible to extend the hierarchy in the sense that logical implications hold as illustrated in Table I.

B. Part 2: Proving Diagonal Distinctiveness

For part 2 we prove diagonal distinctiveness by first creating four protocols that each provide a different level of authentication. We prove in each case that the authentication property holds for all traces of the protocol. Next, we show pairwise differences between the authentication properties by showing traces of each protocol that cause violations of other authentication properties. This proves that the two properties are distinctly different.

We define four protocols, one for each property connected with a diagonal red line in Fig. 2. The protocols are defined with an initiator (labelled *I*) and a responder (labelled *R*), and are shown in Fig. 4. We use the notation $\{x, y, ...\}_{skA}$ to represent a signature over the variables $x, y, ...$ using secret key A. We assume a revealing signature scheme for all figures. We require that the initiator is in possession of the responder's public key and is able to verify signatures made using the responder's private key. In these protocols, each signature using the responder private key is produced as part of the *Running* action.

We then analyse the protocols in a symbolic model, where we assume perfect cryptographic primitives. Specifically, we assume an asymmetric signature scheme, where a signature of a message can only be produced by an agent with knowledge of the secret key. We also assume a Dolev-Yao adversary that can intercept all messages, modify their contents, remove messages from the channel and send its own messages. We further assume that the key of the responder cannot get leaked, however the adversary may possess its own keys. Finally, we assume all agents (including the adversary) are able to generate unique fresh values.

Clearly, in this model there is no way for the adversary to produce any signatures using the responder key. Then any signature required by the initiator must have been produced by the responder, and therefore the responder must agree on at least the data items contained in the signature. From this alone it easy to check that Protocol P_a must fulfil one-way non-injective agreement, Protocol P_b must fulfil weak agreement, and Protocol P_d must fulfil non-injective agreement. Protocol P_c requires a little more motivation. For this protocol, the value *n* is produced by *I*, meaning *Commit* will only occur using values for n that *I* generated and believed to be unique, and so there will only be one *Commit* for each nonce.

In order to make our proofs rigorous, we have used the Tamarin prover [3] in order to verify that our properties hold for each protocol[3]. The Tamarin prover is based upon the same formal model used in this paper.

Lemma 5. $P_a \in S_{ow-ni-agree}$

Lemma 6. $P_b \in S_{w-agree}$

Lemma 7. $P_c \in S_{ow-i-agree}$

Lemma 8. $P_d \in S_{ni-agree}$

Having established that the four protocols each fulfil one of the four properties that are diagonally connected in Fig. 2, we now proceed to show that each protocol violates the diagonally adjacent property.

Fig. 5 shows four traces. Each trace belongs to the corresponding protocol in Fig. 4. The traces represent runs of the protocols where a network adversary has intercepted and modified messages. Trace T_a is a valid trace for Protocol P_a, Trace T_b is a valid trace for Protocol P_b, and so on. Note that in Trace T_d, initiator *I* performs two runs of the protocol, while responder *R* only performs one. The adversary is able to send the final message by repeating a previous message that it has already seen, completing the protocol for *I* without the involvement of *R*.

Lemma 9. $P_a \in S_{ow-ni-agree} \backslash S_{w-agree}$

Proof. Consider the trace T_a shown in Fig. 5 (a). In the trace, each agent follows the rules described by Protocol P_a. The initiator *I* sends and receives matching messages, and the responder *R* receives and then sends a matching message. The steps performed by the adversary *A* falls within the adversary capabilities. The trace is therefore a valid trace for Protocol P_a. This protocol fulfils one-way non-injective agreement according to lemma 5. The trace corresponds to a sequence of actions $T_a = [Running(R, A, n), Commit(I, R, n)]$. This trace is incompatible with Formula 2 for weak agreement, since the

[3]Available at https://gitlab.liu.se/ida-rtslab/public-code/2023_ow-auth

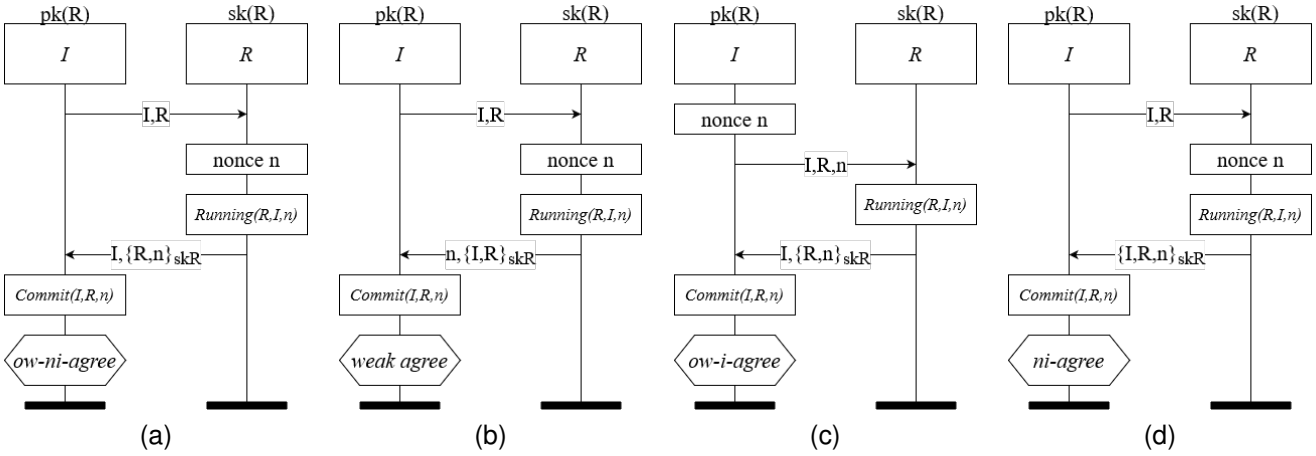

Fig. 4. Distinguishing protocols. Fig. (a) shows a protocol fulfilling one-way non-injective agreement. Fig. (b) shows a protocol fulfilling weak agreement (c) shows a protocol fulfilling one-way injective agreement. Fig. (d) shows a protocol fulfilling non-injective agreement.

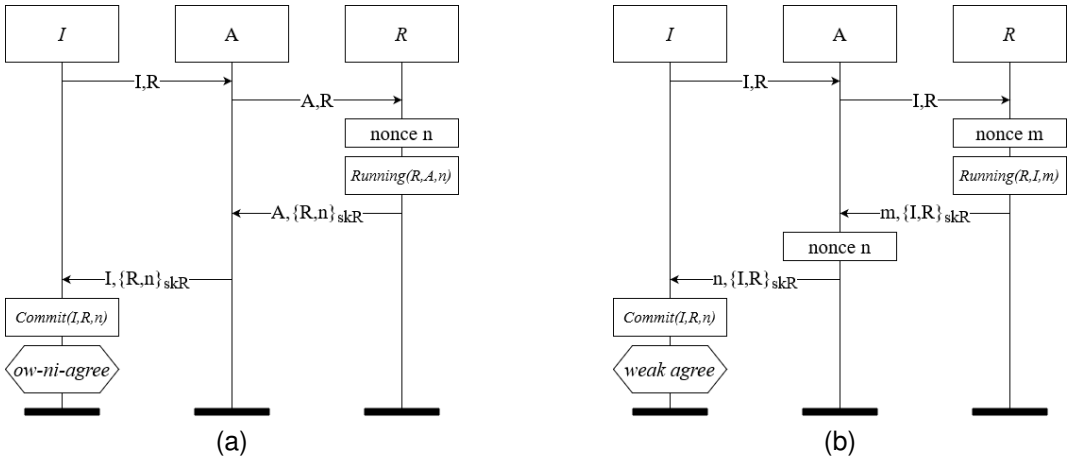

Fig. 5. Distinguishing traces. Fig. (a) shows a trace fulfilling one-way non-injective agreement, but not weak agreement. Fig. (b) shows a trace fulfilling weak agreement, but not one-way non-injective agreement.

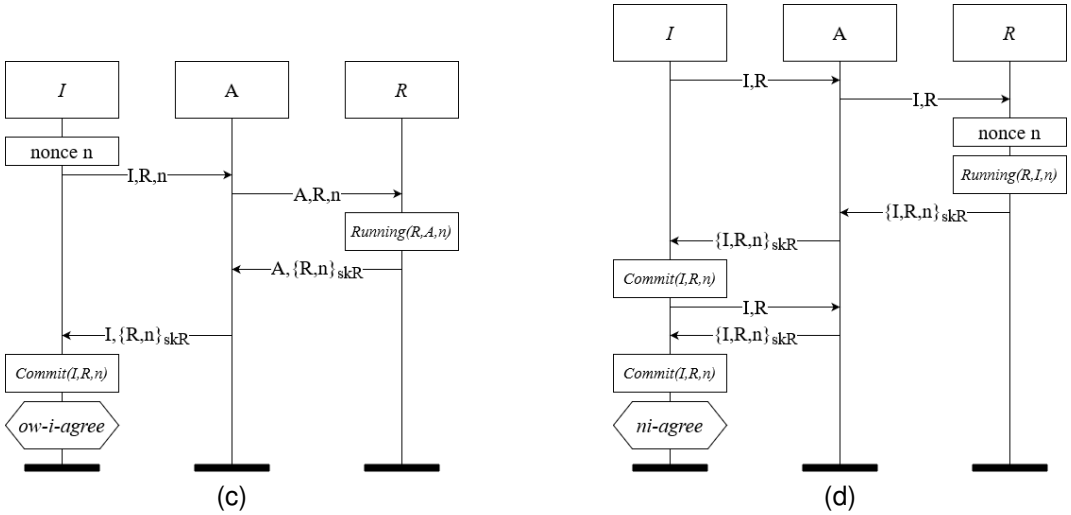

Fig. 5. Distinguishing traces. Fig. (c) shows a trace fulfilling one-way injective agreement, but not non-injective agreement. Fig. (d) shows a trace fulfilling non-injective agreement, but not one-way injective agreement.

initiators do not match between the *Running* and *Commit* actions. In practice this means the responder *R* believed it performed the protocol with the adversary *A* instead of the initiator *I*.

$T_a \in P_a \wedge T_a \nvDash \varphi_{w-agree}$
$\Rightarrow P_a \in S_{ow-ni-agree} \wedge P_a \notin S_{w-agree}$
$\Rightarrow P_a \in S_{ow-ni-agree} \backslash S_{w-agree}$

□

We state similar lemmas for the three remaining traces. The proofs are included in the Appendix, but follow essentially the same procedure. Trace T_d is a special case, since the initiator performs two runs. After the second run, one-way injective agreement does not hold since the *Commit* was not unique. Both runs still adhere to the specification set by Protocol P_d, which fulfils non-injective agreement.

Lemma 10. $P_b \in S_{w-agree} \backslash S_{ow-ni-agree}$

Lemma 11. $P_c \in S_{ow-i-agree} \backslash S_{ni-agree}$

Lemma 12. $P_d \in S_{ni-agree} \backslash S_{ow-i-agree}$

Lemmas 9 – 12 show that some of the properties are distinct from each other, namely that $S_{ow-ni-agree}$ and $S_{w-agree}$ are distinct, and that $S_{ow-i-agree}$ and $S_{ni-agree}$ are distinct.

Lemmas 5 – 8 and 9 – 12 together show that the set of protocols that satisfy the diagonally connected properties in Fig. 2 are distinctly different (i.e., neither is a subset of the other). For each property there is a protocol that belongs to the set of protocols that satisfy that property, but which does not satisfy the diagonally adjacent property.

C. Part 3: Proving Horizontal and Vertical Strictness

We now proceed to show that from Assumption 1 we can conclude that *all* properties are distinct. In particular, we prove inequality for the relations in yellow in Fig. 2. We only show the proof for two of the relations, the remaining two are included in the appendix.

Lemma 13. $S_{ow-ni-agree} \subset S_{alive}$

Proof. 1) From Lemma 4: $S_{ow-ni-agree} \subseteq S_{alive}$
2) From Lemma 10: $P_b \in S_{w-agree} \backslash S_{ow-ni-agree}$
We prove inequality by showing that $S_{alive} \backslash S_{ow-ni-agree}$ is nonempty:
$S_{w-agree} \subset S_{alive}$ (Assumption 1)
$\Rightarrow S_{w-agree} \backslash S_{ow-ni-agree} \subseteq S_{alive} \backslash S_{ow-ni-agree}$
$\overset{2)}{\Longrightarrow} P_b \in S_{alive} \backslash S_{ow-ni-agree}$
$\Rightarrow S_{alive} \neq S_{ow-ni-agree}$
$\overset{1)}{\Longrightarrow} S_{ow-ni-agree} \subset S_{alive}$ □

Lemma 14. $S_{ni-agree} \subset S_{ow-ni-agree}$

Proof. 1) From Lemma 1: $S_{ni-agree} \subseteq S_{ow-ni-agree}$
2) From Lemma 9: $P_a \in S_{ow-ni-agree} \backslash S_{w-agree}$
We prove inequality by showing that $S_{ow-ni-agree} \backslash S_{ni-agree}$ is nonempty:
$S_{ni-agree} \subset S_{w-agree}$ (Assumption 1)
$\Rightarrow S_{ow-ni-agree} \backslash S_{w-agree} \subseteq S_{ow-ni-agree} \backslash S_{ni-agree}$
$\overset{2)}{\Longrightarrow} P_a \in S_{ow-ni-agree} \backslash S_{ni-agree}$
$\Rightarrow S_{ow-ni-agree} \neq S_{ni-agree}$
$\overset{1)}{\Longrightarrow} S_{ni-agree} \subset S_{ow-ni-agree}$ □

Lemma 15. $S_{i-agree} \subset S_{ow-i-agree}$

Lemma 16. $S_{ow-i-agree} \subset S_{ow-ni-agree}$

Now we have proven all relations in Theorem 1. The proof is summarised through the list of lemmas below.

Proof of Theorem 1.
From Assumption 1: $S_{i-agree} \subset S_{ni-agree} \subset S_{w-agree} \subset S_{alive}$
From Lemma 15: $S_{i-agree} \subset S_{ow-i-agree}$
From Lemma 14: $S_{ni-agree} \subset S_{ow-ni-agree}$
From Lemma 16: $S_{ow-i-agree} \subset S_{ow-ni-agree}$
From Lemma 13: $S_{ow-ni-agree} \subset S_{alive}$ □

We have also shown that our definitions are distinctly different from existing authentication properties. We state this as Theorem 2 and Theorem 3.

Theorem 2. $S_{ow-i-agree} \neq S_{ni-agree}$

Proof. From Lemma 11: $P_c \in S_{ow-i-agree} \backslash S_{ni-agree}$
$\Rightarrow S_{ow-i-agree} \neq S_{ni-agree}$ □

Theorem 3. $S_{ow-ni-agree} \neq S_{w-agree}$

Proof. From Lemma 9: $P_a \in S_{ow-ni-agree} \backslash S_{w-agree}$
$\Rightarrow S_{ow-ni-agree} \neq S_{w-agree}$ □

V. Preplay Attacks and Injectivity

A preplay attack occurs when an adversary is able to predict and perform a part of the protocol before it is initiated, and then replay messages in order to successfully complete the protocol at a later stage. An injective protocol guarantees that no replay attacks can occur, but not necessarily that preplay attacks are impossible . Preplay attacks are attacks where the adversary can break security by preparing a protocol run in advance. Consider the case that the initiator is using a counter instead of a nonce in order to identify each protocol run. The initiator can be sure that the responder has performed the protocol at least as many times as itself by requiring a message signed by the responder with the same counter value that was used to initiate the protocol. However, since the counter values are predictable, an adversary could perform preplay attacks by initiating a run with the responder using a counter value that will be used in the future (injective agreement does not require the initial message to be signed). In order to protect against preplay attacks, we may want to describe a property that is at least as strong as injective agreement but that additionally ensures that the responder run occurred recently. To this end we define a property named *recent injective agreement*. We then show that for a reasonable definition of recent injective agreement, there is an alternative method to provide preplay attack protection that is far simpler to reason about. Namely, usage of fresh data automatically ensures that a run must have been performed recently. Reasoning about recentness in

this way was proposed originally by Lowe [2]. Our work contributes by providing a formal definition for recentness, along with a formal proof that agreement on fresh data ensures recentness. Below we define recent injective agreement.

a) Definition: recent injective agreement: A protocol guarantees recent injective agreement to an agent *A* acting as the initiator if *A* completes a run of the protocol apparently with *B*, then there was an agent *B* acting as responder, and *A* and *B* agreed on the data for this run, and each such run of A corresponds to a unique run of B, and the run by *B* happened at some point after the protocol was initiated by *A*.

We formalise the definition by introducing a *Start* action, which is performed by the initiator upon initiating the protocol. Our property requires that *Running* happened before *Commit* but after *Start*.

$$\begin{aligned} &\varphi_{r-i-agree} \triangleq \\ &\forall\, I\, R\, n\, i.\, Commit(I,R,n)@i \\ &\quad \rightarrow (\exists\, j\, k.\, Running(R,I,n)@j \\ &\qquad \wedge\ Start(I,R,n)@k \\ &\qquad \wedge\ k<j \wedge j<i \\ &\qquad \wedge \neg(\exists\, I_2\, R_2\, i_2.\, Commit(I_2,R_2,n)@i_2 \\ &\qquad\quad \wedge\ i_2 \neq i)) \end{aligned} \tag{9}$$

Recent injective agreement gives us a guarantee regarding the timing of the *Running* action. *Running* must have occurred while the initiator was performing the protocol, since it occurred at a timepoint between the start and end of the protocol.

An alternative way to specify a property in order to protect against preplay attacks named *synchronisation* was given by Cremers et al. [8]. Injective synchronisation is a stronger property than recent injective agreement, since it requires an ordering for all messages in the protocol, instead of only the three specified here. Cremers et al. provide a sufficient condition for a non-injectively synchronising protocol to also be injective, and named it the LOOP property. The LOOP property describes that given a role and a claim for that role (in our formalism a *Commit* action), if for every other role which has an event that precedes the claim, there exists some other event for the same role (e.g. *Running*) which happened after an event in the claim role (e.g. *Start*), but before the claim itself. The LOOP property is a sufficient condition to fulfil injective synchronisation for a protocol fulfilling non-injective synchronisation.

The LOOP property and our definition of recency both involve a casual chain of actions that alternate roles. As noted by Cremers et al. it is possible to achieve such a causal chain using a nonce, although it is not always sufficient. Since we assume the protocol is already injective, it is sufficient in our case, and a nonce can be used to provide recency. We take inspiration from the LOOP property and provide a condition for showing that a protocol fulfilling injective agreement also fulfils recent injective agreement.

b) Definition: Unique Start Property: A protocol fulfils the unique start property if *Start* is always performed before each *Commit* in the protocol steps performed by the initiator, and each matching *Start* and *Commit* uses the same nonce, and each *Start* uses a unique nonce that does not appear in any previous action.

The unique start property is a sufficient requirement in order for any protocol fulfilling injective agreement to also fulfil recent injective agreement. We provide a proof below.

Proof. Assume any trace *T* where injective agreement holds, and the unique start property holds. Then for every *Commit* action there exists a *Running* action which happened earlier, using the same nonce as the given *Commit*. From the unique start property we can conclude that at some point before each *Commit*, there was a *Start* action using the same nonce. The unique start property also requires that *Start* must have been the first action to use this particular nonce, and so must have happened before both *Running* and *Commit*. We can then conclude that trace *T* fulfils recent injective agreement. Since *T* was an arbitrary trace, any protocol *P* for which every trace $T \in traces(P)$ fulfils injective agreement and the unique start property, all traces must also fulfil recent injective agreement. □

The unique start property is very useful since it is already provided by a very common mechanism, namely using unique and unpredictable nonces. We only need to assume that generated nonces are sufficiently unique and sufficiently unpredictable in order to provide a unique identity for each protocol run. Once it has been proven that injective agreement holds, we can rule out preplay attacks[4] based on proper usage of nonces. We have not included recent injective agreement in the hierarchy because it is so similar in practice to injective agreement when using nonces. In principle, we could extend the hierarchy also with recency, and additionally capture a corresponding one-way property.

These results depend on the restrictions we have applied on what protocols we consider. Importantly, if *Commit* and *Start* are not performed by the initiator, so that *Start* is not the first action of the protocol, then we can no longer guarantee that the initial message was sent recently. An adversary that can inject messages may be able to initiate the protocol earlier than intended. This could be considered a form of preplay attack. The property recent injective agreement still gives a guarantee that the continuation of the protocol was recent, since we ensure that a *Running* action occurred after *Start*.

Our results generalise to protocols with more than one data item, since we have put no restriction on the contents of n. The variable n may consist of a list of values, and as long as a single item in n is unique and unpredictable then our assumptions still hold.

VI. Remote Attestation Case Study

We exemplify how specifications using one-way authentication requirements naturally arise for some protocols by using remote attestation as an example. Remote attestation is a

[4]Not all types of preplay attacks can be ruled out, but we can at least rule out an attack where *Running* occurred before the protocol was started.

process in which some target platform proves or attests its correct and trustworthy operation to some remote party [15][16]. Remote attestation schemes have been proposed for the purpose of malware detection for low-end devices [17], trust in enterprise networks [18], and for secure provisioning of workloads to cloud services [19][20].

Remote attestation requires a mechanism for providing a proof of correct state on the target. Trustworthy integrity measurements can be performed by trusted firmware or dedicated hardware modules, forming a hardware root of trust. We refer to this part of the platform as the *attestation agent*. It is typically desirable for any root of trust to be minimal, so that it is easy to reason about the security of the trusted computing base. Therefore, we expect the attestation agent to have very limited capabilities.

Remote attestation protocols involve two main roles, the *verifier* and the *prover* [16]. The verifier is the role interested in the state of the prover. The prover is the role providing evidence of the trustworthiness of some *target environment* through the use of the attestation agent. The verifier, target and attestation agent all participate in the protocol. Remote attestation protocols are interesting since they are naturally asymmetric. The verifier is interested in the state of the target, but it is the attestation agent which is required to provide trustworthy evidence of the target. In addition, while it is crucial that the verifier can trust the attestation, and therefore the attestation agent, the attestation agent may not have knowledge about the identity of the verifier.

In the next section we describe an attestation scenario and an attacker model. In the section after we list some attacks that a remote attestation protocol will need to protect against, and then in a following section we describe the security properties we should require in order to guarantee absence of the attacks listed.

A. Attestation Scenario and Attacker Model

We will describe a remote attestation scenario where some low-end medical equipment is continuously being monitored to ensure correct operation. In order to ensure that the equipment is operating correctly, a remote attestation protocol is implemented as part of the trusted computing base built into the equipment. In order for the equipment to prove its trustworthiness, remote attestation is performed. The equipment will be acting as the prover in this scenario. The remote attestation scenario follows the protocol sketch shown in Fig. 6.

First the verifier sends a challenge to the prover, asking it to prove it is in a correct state. The prover uses its attestation agent in order to collect proof of its correct operation. This is sent back to the verifier, and the verifier then decides whether to trust or not to trust the prover based on the information provided by the attestation agent.

We describe the role capabilities and objectives in more detail. We first focus on the verifier role. The verifier is trying to perform the protocol with a particular target T, and ensure that T is operating correctly. The verifier has failed its objective if it is possible for an adversary to compromise

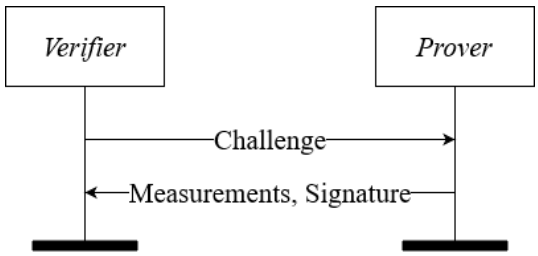

Fig. 6. Example Remote Attestation Protocol

TABLE II
REMOTE ATTESTATION PROTOCOL GOALS

Role	Objective
Verifier	Ensure that target is secure
Target	Provide attestation on request
Attestation Agent	Provide measurements of target state
Adversary	Compromise target without detection

T at any point without the verifier ever learning about the compromise. We assume that the verifier knows all devices it manages, and specifically that it knows the public attestation key provisioned to the attestation agent of each prover. The objective of the target and the attestation agent is simply to provide a proof of correct operation. Finally, the adversary is attempting to compromise the target while avoiding detection by the verifier. The goals for each role are summarised in Table II.

B. Specifying Security Properties

Remote attestation faces a number of challenges. The central requirement, common to all use-cases of remote attestation, is that a proof sent by the prover should accurately reflect the state of the target. Sailer et al. [21] described some of the potential attacks that are important for a remote attestation protocol to protect against. They required protection against replay attacks, data tampering and masquerading attacks. Preplay attacks are also a potential danger to remote attestation protocols. A preplay attack in the context of remote attestation could involve the adversary storing an honest attestation response before compromising a target. Once the verifier requests a new attestation, the old attestation is sent instead. A mechanism must be in place to ensure the old data is not accepted. The attacks are summarised in Table III together with an explanation of what the attacks mean in the context of remote attestation, as well as the corresponding authentication property that mitigates the attack.

Masquerading attacks are possible when the verifier is not able to correctly identify the attesting agent. If the verifier thinks it is verifying some device A, when in reality attestation was performed by device B, there is a complete lack of authentication. With aliveness we at least ensure that device A is capable of performing remote attestation, and has done so at some point. While aliveness is not enough to prevent past messages from being replayed, or that measurement data was authentic, we will cover such cases as part of

TABLE III
ATTACKS THAT COULD COMPROMISE THE SECURITY OF REMOTE ATTESTATION. THE RIGHTMOST COLUMN SHOWS A PROPERTY THAT WILL HELP IN MITIGATING THE ATTACK.

Attack	**Explanation**	**Auth. Property**
Masquerading Attack	The adversary is able to masquerade as another device, by utilising another device's response in order to attest itself to the verifier.	Aliveness
Measurement Tampering	The adversary is able to tamper with the attestation data, in order to remove any discriminating evidence of device compromise.	One-Way Non-injective Agreement
Replay Attack	The adversary is able to reuse previous attestation data or messages in order to attest itself.	One-Way Injective Agreement
Preplay Attack	The adversary is able to prepare an attestation that can be used later once the system has been compromised	One-Way Recent Injective Agreement

the protection for the other attacks. Preventing measurement tampering requires that we include agreement on data into our authentication specification. This is provided by the non-injective agreement properties, and so we should require either non-injective agreement or one-way non-injective agreement. A replay attack can occur if we do not require unique data for every run. Injective agreement ensures that unique verifier runs always correspond to unique device runs. If a protocol is at least one-way injective, replay attacks are not possible. As shown in Section V, assuming our verifier always generates a new nonce at the start of the protocol in order to provide injectivity, then we also protect against preplay attacks where the attestation was performed before the initial request. One-way injective agreement is enough to cover agreement on responder identity, meaning the verifier knows which prover attested. In addition, there is agreement on measurement data, meaning it was the same as when it was sent. Finally, it ensures that the attestation was unique and not replayed.

C. Mutual vs. One-Way

As can be seen in Table III, one-way injective agreement is sufficient to prevent common remote attestation attacks. We can ask if it is beneficial to additionally consider (mutual) injective agreement. The guarantee provided by mutual authentication as opposed to one-way authentication, is that no other initiator (in this case the verifier) can intercept communication with the responder (the prover) in order to introduce itself as the initiator to the responder. This attack is prevented if the response from the responder conveys the identity of its believed initiator, since then the true initiator can see that there was no initiator mismatch.

In the scenario with the low-end medical equipment, no communication is meant to take place after the attestation. Once the prover has sent the response then the protocol is completed, and the prover is not waiting for any additional information from its believed initiator. If an additional verifier masquerades itself as the origin of the attestation challenge, then this makes no difference for neither the prover nor the original verifier. The verifier still receives the same information, and the prover can complete the protocol regardless. Since there seems to be no security benefits to additionally indicating the original verifier during attestation, we may consider omitting it during protocol design in order to save bandwidth and make attestation more efficient.

VII. RELATED WORK

The first part of this section goes through related work on specifying authentication properties. We then highlight some examples where one-way authentication properties have been defined or used in the literature, but without the same formal treatment, or in contexts different from ours. Finally, we describe related work to our case-study. Specifically, we mention some previous works where authentication properties for remote attestation have been discussed.

A. Hierarchies

As discussed in Section II, the original work by Lowe [2] forms the foundation of our work. In it, Lowe defines and formalises authentication properties in the process algebra CSP. We extend this work by identifying suitable properties for one-way authentication, and show how such properties relate to the existing hierarchy. In addition, we provide a formalisation of a claim made by Lowe, that relates recentness and agreement through the usage of fresh data.

Roscoe [9] proposed an alternative authentication specification in the form of an intensional specification for protocols. An intensional specification does not depend on what each protocol is trying to achieve, rather it treats the ordering of messages to be the defining trait of a correctly executed protocol. For the intensional specification to be fulfilled, each message must be sent and received in the correct order for each run. This property is stronger than the definition of injective agreement, since we additionally consider the ordering of individual messages as violation of the protocol security.

Work by Cremers et al. [8] formalised a definition for intensional authentication named *synchronisation*, and showed how non-injective and injective synchronisation could be seen as a strictly stronger property compared to non-injective and injective agreement respectively. Our methodology for proving strictness takes inspiration from their work. Their definition extends the hierarchy, providing stronger protocol requirements. We have taken the opposite approach by specifying properties that are strictly weaker than the definition of injective agreement. We argue both approaches are needed in order to provide sufficient granularity during verification, but also allow verifying the strongest known requirements in order to prove absence of any unintended behaviour. Their definition does not distinguish between data items in the form of agent names or nonces. This in practice means that the distinction between one-way authentication and mutual authentication is not specified in the security property for their model, but is rather a consequence of the protocol specification.

Cremers and Mauw [22] define an alternative authentication hierarchy that builds upon on the synchronisation specification. This hierarchy additionally considers multiple properties that each describe flavours of aliveness, in addition to the synchronisation properties presented in [8]. The aliveness properties are *weak aliveness*, *weak aliveness in the correct role*, *recent aliveness* and *recent aliveness in the correct role*. Due to differences in the formalisation of properties, it is not possible to directly translate the formalism to ours. In particular, the formalism used in our work does not allow for mismatched roles, since the actions *Running* and *Commit* are only considered from the perspective of a particular role. Thus their definition of weak aliveness is weaker than any specification described here, since it only states that an agent has performed some action, regardless of which step or in which role. In the formalism used by Cremers and Mauw, the aliveness properties and the agreement properties are not related: proving a property from one side of the hierarchy does not prove anything about any property from the other side of the hierarchy. This means that the relative strength of the properties is not apparent. In particular, recent aliveness with the correct roles is not stronger, nor weaker than non-injective agreement according to their analysis. The reason is subtle, since a protocol without any communication fulfils non-injective agreement but not recent aliveness in the correct role, according to their definitions. It is not immediately obvious how the variations of aliveness would be useful in the analysis of real-world protocols.

A definition of authentication called *correspondence* was given by Woo and Lam in 1993 [23]. The definition is similar to agreement in the sense that it requires that an event A should have been preceded by an event B. When using correspondence to define authentication, each event may contain different parameters. In a sense correspondence is a very general definition for agreement, however we must reason about the meaning of each specification individually. Authentication properties in the protocol verifier ProVerif [24] are specified as correspondence assertions.

Yet another definition of authentication is known as *entity authentication* [25][26]. Entity authentication describes informally that the protocol provides recent evidence that an entity really is the one claimed. In addition, strong entity authentication is defined so that the responder must provide evidence of its peer entity. Therefore, entity authentication describes a one-way authentication property, while strong entity authentication describes a mutual authentication property. It is not clear how entity authentication is to be formalised in a symbolic model.

Generic formal goals for authentication protocols were given by van Oorschot [27] in the context of authenticated key establishment. Six goals are specified, and they are stated as beliefs of agent A about some intended other agent B. Some goals are shown to be related to each other, but not all relations. Formulating goals specifically for key establishment was out of scope for our work.

B. One-Way Authentication Usage

Cremers et al. [10] formally verified draft 10 of TLS 1.3 using Tamarin. It was noted that for unilaterally authenticated mode it was not possible to guarantee that the server believed it was communicating with the client who actually initiated the protocol. In order to verify that the client could authenticate the server, a weakening of the usual agreement property had to be used. The property used corresponds to our definition of one-way non-injective agreement.

Another, later draft of TLS 1.3 was formally verified by Bhargavan et al. [11] using ProVerif. During formal verification an issue with RTT 0.5 was found, which allowed for a mismatch between the believed authentication of the client and the server. In particular, it is possible that the client believed it was authenticated to the server when it received early data, when in reality the server had not received the authenticating messages, and so believed the message was sent to an anonymous client. This effectively corresponds to a mismatch on initiator identity similar to the one-way properties discussed. Bhargavan et al. refer to this weakened property as weak authentication. Weak authentication was undesired in the case they analysed, since the intent was to use mutual TLS, and so the client expects that it is authenticated. If the intent is to perform unilateral TLS then the client is effectively anonymous, and so weak authentication is expected.

Some works have discussed one-way authentication [28][29][30], primarily in the context of key exchange protocols. Authenticated key exchange takes a different view on authentication. In such models, authentication and secrecy is usually combined, with authentication being defined as a successful exchange of a session key between two agents. As long as it can be guaranteed that the key can only be known to the specified agents, then usage of the session key is enough to prove authentication. While such definitions are useful in the context of authenticated key exchange, they do not cover pure authentication. Furthermore, having a separation between authentication and secrecy properties can be beneficial in order to more clearly specify requirements, and in order to allow for changes in security models more easily.

C. Remote Attestation Authentication

Formal verification of remote attestation can be divided into two categories. Ether the integrity proof mechanism is verified, or the properties of the protocol. Describing properties for formal verification of remote attestation protocols has received limited attention. Of the works on protocol verification, we have examples of verification for Direct Anonymous Attestation [31][6][7][32], Intel SGX attestation [33], and Intel TDX attestation [34]. During formal verification of Direct Anonymous Attestation, focus has been on verifying the anonymity of the proposed cryptographic schemes.

VIII. Conclusions

In conclusion, we argue that the strongest possible definition for authentication is not always appropriate, and that it is

sometimes necessary to specify a weaker property in order to accurately capture the guarantees provided by a protocol. One-way authentication requires strictly weaker authentication requirements, and has received little attention outside of authenticated key exchange. We show that one-way authentication is a useful concept that generalises to other types of protocols such as remote attestation, in addition to unilateral TLS which has been discussed previously in the literature. We also conclude that considerable effort is needed in order to formally prove the relationship between different authentication properties. Seemingly distinct trace properties may sometimes coincide due to adversary capabilities or protocol assumptions. In particular our modified injective agreement property coincides with injective agreement, as does recent injective agreement when we can assume the unique start property.

Acknowledgment

We thank the reviewers for their valuable feedback and suggestions, in particular for pointing out an inconsistency in how we handled the temporal condition for Lowe's original properties.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] M. Barbosa, G. Barthe, K. Bhargavan, *et al.*, "Sok: Computer-aided cryptography," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 777–795. DOI: 10.1109/SP40001.2021.00008.

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

[3] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, Springer Berlin Heidelberg, 2013, pp. 696–701, ISBN: 978-3-642-39799-8.

[4] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, 2018, pp. 1383–1396. DOI: 10.1145/3243734.3243846.

[5] C. Cremers and M. Dehnel-Wild, "Component-based formal analysis of 5g-aka: Channel assumptions and session confusion," 2019.

[6] J. Whitefield, L. Chen, R. Sasse, S. Schneider, H. Treharne, and S. Wesemeyer, "A symbolic analysis of ecc-based direct anonymous attestation," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 127–141.

[7] S. Wesemeyer, C. J. Newton, H. Treharne, L. Chen, R. Sasse, and J. Whitefield, "Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme," ser. ASIA CCS '20, Association for Computing Machinery, 2020, pp. 784–798. DOI: 10.1145/3320269.3372197.

[8] C. Cremers, S. Mauw, and E. de Vink, "Injective synchronisation: An extension of the authentication hierarchy," *Theoretical Computer Science*, vol. 367, no. 1, pp. 139–161, 2006, Automated Reasoning for Security Protocol Analysis, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2006.08.034.

[9] A. Roscoe, "Intensional specifications of security protocols," in *Proceedings 9th IEEE Computer Security Foundations Workshop*, 1996, pp. 28–38. DOI: 10.1109/CSFW.1996.503688.

[10] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 470–485. DOI: 10.1109/SP.2016.35.

[11] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the tls 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 483–502. DOI: 10.1109/SP.2017.26.

[12] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *2012 IEEE 25th Computer Security Foundations Symposium*, 2012, pp. 78–94. DOI: 10.1109/CSF.2012.25.

[13] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983. DOI: 10.1109/TIT.1983.1056650.

[14] E. Rescorla, *The transport layer security (tls) protocol version 1.3*, 2018. DOI: 10.17487/RFC8446.

[15] G. Coker, J. Guttman, P. Loscocco, *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 2011.

[16] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified Hardware/Software Co-Design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*, USENIX Association, 2019, pp. 1429–1446, ISBN: 978-1-939133-06-9.

[17] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6. DOI: 10.7873/DATE.2014.257.

[18] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn, "Attestation-based policy enforcement for remote access," ser. CCS '04, Washington DC, USA: Association for Computing Machinery, 2004, pp. 308–317. DOI: 10.1145/1030083.1030125.

[19] G. Chen, Y. Zhang, and T.-H. Lai, "Opera: Open remote attestation for intel's secure enclaves," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, Association for Computing Machinery, 2019, pp. 2317–2331. DOI: 10.1145/3319535.3354220.

[20] M. Russinovich, "Introducing azure confidential computing," *Microsoft Azure Blog*, 2017. [Online]. Available: https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/.

[21] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a tcg-based integrity measurement architecture.," in *USENIX Security symposium*, vol. 13, 2004, pp. 223–238.

[22] C. Cremers and S. Mauw, "Security properties," in *Operational Semantics and Verification of Security Protocols*. Springer Berlin Heidelberg, 2012, pp. 37–65. DOI: 10.1007/978-3-540-78636-8_4.

[23] T. Woo and S. Lam, "A semantic model for authentication protocols," in *Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, 1993, pp. 178–194. DOI: 10.1109/RISP.1993.287633.

[24] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[25] C. Boyd, A. Mathuria, and D. Stebila, "Introduction to authentication and key establishment," in *Protocols for Authentication and Key Establishment*. Springer Berlin Heidelberg, 2020, pp. 1–52. DOI: 10.1007/978-3-662-58146-9_1.

[26] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology (CRYPTO' 93)*, Springer Berlin Heidelberg, 1994, pp. 232–249, ISBN: 978-3-540-48329-8.

[27] P. van Oorschot, "Extending cryptographic logics of belief to key agreement protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93)*, Association for Computing Machinery, 1993, pp. 232–243. DOI: 10.1145/168588.168617.

[28] I. Goldberg, D. Stebila, and B. Ustaoglu, "Anonymity and one-way authentication in key exchange protocols," *Designs, Codes and Cryptography*, vol. 67, no. 2, pp. 245–269, 2013.

[29] Y. Dodis and D. Fiore, "Interactive encryption and message authentication," in *International Conference on Security and Cryptography for Networks*, Springer, 2014, pp. 494–513.

[30] Y. Dodis and D. Fiore, "Unilaterally-authenticated key exchange," in *International Conference on Financial Cryptography and Data Security*, Springer, 2017, pp. 542–560.

[31] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attes-

tation protocol," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, pp. 202–215. DOI: 10.1109/SP.2008.23.

[32] B. Smyth, M. D. Ryan, and L. Chen, "Formal analysis of privacy in direct anonymous attestation schemes," *Science of Computer Programming*, vol. 111, pp. 300–317, 2015, Special Issue on Automated Verification of Critical Systems (AVoCS 2013), ISSN: 0167-6423. DOI: 10.1016/j.scico.2015.04.004.

[33] M. U. Sardar, D. L. Quoc, and C. Fetzer, "Towards formalization of enhanced privacy id (epid)-based remote attestation in intel sgx," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, 2020, pp. 604–607. DOI: 10.1109/DSD51259.2020.00099.

[34] M. U. Sardar, S. Musaev, and C. Fetzer, "Demystifying attestation in intel trust domain extensions via formal verification," *IEEE Access*, vol. 9, pp. 83 067–83 079, 2021. DOI: 10 . 1109 / ACCESS . 2021 . 3087421.

APPENDIX

A. Proofs for lemmas 1, 2, and 4.

Lemma 1. $\varphi_{ni-agree} \Rightarrow \varphi_{ow-ni-agree}$

Proof. Definition of non-injective agreement from Formula 3:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, j.\, Running(R, I, n)@j)$

So for any commit there is some running event with matching arguments. Then, with $I_2 = I$:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, I_2\, j.\, Running(R, I_2, n)@j)$ □

Lemma 2. $\varphi_{i-agree} \Rightarrow \varphi_{ow-i-agree}$

Proof. Definition of injective agreement from Formula 4:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, j.\, Running(R, I, n)@j \wedge\ j < i$
$\wedge\neg(\exists\, I_2\, R_2\, i_2.\, Commit(I_2, R_2, n)@i_2 \wedge\ i_2 \neq i))$

Rename I_2 to I_3.
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, j.\, Running(R, I, n)@j \wedge\ j < i$
$\wedge\neg(\exists\, I_3\, R_2\, i_2.\, Commit(I_3, R_2, n)@i_2 \wedge\ i_2 \neq i))$

Then, with $I_2 = I$:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, I_2\, j.\, Running(R, I_2, n)@j \wedge\ j < i$
$\wedge\neg(\exists\, I_3\, R_2\, i_2.\, Commit(I_3, R_2, n)@i_2 \wedge\ i_2 \neq i))$

□

Lemma 4. $\varphi_{ow-ni-agree} \Rightarrow \varphi_{alive}$

Proof. Definition of one-way non-injective agreement from Formula 6:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, I_2\, j.\, Running(R, I_2, n)@j)$

With $n2 = n$:
$\forall\, I\, R\, n\, i.\, Commit(I, R, n)@i$
$\rightarrow\ (\exists\, I_2\, n_2\, j.\, Running(R, I_2, n_2)@j$

□

B. Proofs for lemmas 10 – 12

Lemma 10. $P_b \in S_{w-agree} \backslash S_{ow-ni-agree}$

Proof. Consider the trace T_b shown in Fig. 5 (b). In the trace, each agent follows the rules described by Protocol P_b. The initiator *I* sends and receives matching messages, and the responder *R* receives and then sends a matching message. The steps performed by the adversary *A* falls within the adversary capabilities. The trace is therefore a valid trace for Protocol P_b. This protocol fulfils weak agreement according to lemma 5. The trace corresponds to a sequence of actions $T_b = [Running(R, I, m), Commit(I, R, n)]$. This trace is incompatible with Formula 6 for one-way non-injective agreement, since the nonces do not match between the *Running* and *Commit* facts. In practice this means the responder *R* believed it performed the protocol using nonce *m* instead of the value *n* seen by the initiator. $T_b \in P_b \wedge T_b \nvdash \varphi_{ow-ni-agree}$
$\Rightarrow P_b \in S_{w-agree} \wedge P_b \notin S_{ow-ni-agree}$
$\Rightarrow P_b \in S_{w-agree} \backslash S_{ow-ni-agree}$ □

Lemma 11. $P_c \in S_{ow-i-agree} \backslash S_{ni-agree}$

Proof. Consider the trace T_c shown in Fig. 5 (c). In the trace, each agent follows the rules described by Protocol P_c. The initiator *I* sends and receives matching messages, and the responder *R* receives and then sends a matching message. The steps performed by the adversary *A* falls within the adversary capabilities. The trace is therefore a valid trace for Protocol P_c. This protocol fulfils one-way injective agreement according to lemma 7. The trace corresponds to a sequence of actions $T_c = [Running(R, A, n), Commit(I, R, n)]$. This trace is incompatible with Formula 3 for non-injective agreement, since the initiators do not match between the *Running* and *Commit* facts. In practice this means the responder *R* believed it performed the protocol with the adversary *A* instead of the Initiator *I*. $T_c \in P_c \wedge P_c \wedge T_c \nvdash \varphi_{ni-agree}$
$\rightarrow P_c \in S_{ow-i-agree} \wedge P_c \notin S_{ni-agree}$
$\rightarrow P_c \in S_{ow-i-agree} \backslash S_{ni-agree}$ □

Lemma 12. $P_d \in S_{ni-agree} \backslash S_{ow-i-agree}$

Proof. Consider the trace shown in Fig. 5 (d). In the trace, each agent follows the rules described by Protocol P_d. The initiator *I* performs two runs. In the first run, the initiator run perfectly matches the responder run. However, in the second run the adversary replays the previous messages sent by the responder. This is within the capabilities of the adversary. Therefore, the trace is a valid trace for P_d. This protocol fulfils non-injective agreement according to lemma 8. The trace corresponds to a sequence of actions $T_d = [Running(R, A, n), Commit(I, R, n), Commit(I, R, n)]$. This trace is incompatible with Formula 7 for one-way injective agreement, since the initiator was tricked to accept a replayed responder run. $T_d \in P_d \wedge T_d \nvdash \varphi_{ow-i-agree}$
$\Rightarrow P_d \in S_{ni-agree} \wedge P_d \notin S_{ow-i-agree}$
$\Rightarrow P_d \in S_{ni-agree} \backslash S_{ow-i-agree}$ □

C. Proofs for lemmas 15 and 16

Lemma 15. $S_{i-agree} \subset S_{ow-i-agree}$

Proof. 1) From Lemma 2: $S_{i-agree} \subseteq S_{ow-i-agree}$
2) From Lemma 11: $P_c \in S_{ow-i-agree} \backslash S_{ni-agree}$
We prove inequality by showing that $S_{ow-i-agree} \backslash S_{i-agree}$ is nonempty:
$S_{i-agree} \subset S_{ni-agree}$ (Assumption 1)
$\Rightarrow S_{ow-i-agree} \backslash S_{ni-agree} \subseteq S_{ow-i-agree} \backslash S_{i-agree}$
$\overset{2)}{\Longrightarrow} P_c \in S_{ow-i-agree} \backslash S_{i-agree}$
$\Rightarrow S_{ow-i-agree} \neq S_{i-agree}$
$\overset{1)}{\Longrightarrow} S_{i-agree} \subset S_{ow-i-agree}$ □

Lemma 16. $S_{ow-i-agree} \subset S_{ow-ni-agree}$

Proof. 1) From Lemma 3: 1) $S_{ow-i-agree} \subseteq S_{ow-ni-agree}$
2) From Lemma 12: 3) $P_d \in S_{ni-agree} \backslash S_{ow-i-agree}$
We prove inequality by showing that
$S_{ow-ni-agree} \backslash S_{ow-i-agree}$ is nonempty:

$S_{ni-agree} \subset S_{ow-ni-agree}$ (Lemma 14)
$\Rightarrow S_{ni-agree} \backslash S_{ow-i-agree} \subseteq S_{ow-ni-agree} \backslash S_{ow-i-agree}$
$\overset{2)}{\Longrightarrow} P_d \in S_{ow-ni-agree} \backslash S_{ow-i-agree}$
$\Rightarrow S_{ow-ni-agree} \neq S_{ow-i-agree}$
$\overset{1)}{\Longrightarrow} S_{ow-i-agree} \subset S_{ow-ni-agree}$ □