Noname manuscript No.
(will be inserted by the editor)

Measuring Protocol Strength with Security Goals

Paul D. Rowe · Joshua D. Guttman · Moses D. Liskov

the date of receipt and acceptance should be inserted later

Abstract Flaws in published standards for security protocols are found regularly, often after systems implementing those standards have been deployed. Because of deployment constraints and disagreements among stakeholders, different fixes may be proposed and debated. In this process, security improvements must be balanced with issues of functionality and compatibility.

This paper provides a family of rigorous metrics for protocol security improvements. These metrics are sets of first order formulas in a goal language $\mathcal{GL}(\Pi)$ associated with a protocol Π. The semantics of $\mathcal{GL}(\Pi)$ is compatible with many ways to analyze protocols, and some metrics in this family are supported by many protocol analysis tools. Other metrics are supported by our Cryptographic Protocol Shapes Analyzer CPSA.

This family of metrics refines several "hierarchies" of security goals in the literature. Our metrics are applicable even when, to mitigate a flaw, participants must enforce policies that constrain protocol execution. We recommend that protocols submitted to standards groups characterize their goals using formulas in $\mathcal{GL}(\Pi)$, and that discussions comparing alternative protocol refinements measure their security in these terms.

1 Introduction

Security standards often contain flaws, and therefore evolve over time as people correct them. Often, these flaws are discovered after deployment, which puts pressure on the choice of mitigation. The constraints of operational deployments and the needs of the various stakeholders are crucial, but so is understanding the significance of the attack.

The MITRE Corporation.
E-mail: {prowe,guttman,mliskov}@mitre.org.

How are we to choose among the alternatives proposed to repair a flaw? A security flaw is a failure of the protocol to meet a goal—often one not well understood until after the flaw becomes apparent—and a revised understanding of the goals of the protocol is necessary to ensure that the mitigation is secure. Obviously, satisfying a goal that was not previously met is essential to any mitigation. However, two alternatives may eliminate the same insecure scenario while differing in the security they provide.

In this paper, we describe a formal language for expressing protocol security goals. We use sets of formulas in this language to compare protocols. A protocol Π_2 is then at least as secure as another protocol Π_1 with respect to a set of goal formulas G if, for each formula $\Gamma \in G$, if Π_1 achieves the goal Γ, then so does Π_2. We will explore various sets of formulas G below. We will argue that any set G containing formulas of a reasonable syntactic form should be regarded as a "measurement" for the security of the protocols.

In the case of a singleton $G = \{\Gamma\}$, we can use a variety of protocol analysis tools such as Maude-NPA [18], ProVerif [8], Scyther [14], and our Cryptographic Protocol Shapes Analyzer CPSA [32] to ascertain whether Π_2 is at least as secure as Π_1 with respect to G. It suffices to show that Π_2 achieves the goal Γ, or else that some attack on Π_1 provides a counterexample showing that Π_1 does not achieve Γ. In the same spirit, we can use the same tools to measure protocols relative to two-element sets $G = \{\Gamma_1, \Gamma_2\}$ or other finite sets. We simply evaluate the protocols for each goal separately and combine the results.

CPSA also allows protocol comparisons using larger sets G, especially sets of implications $\Phi \Longrightarrow \Psi$ that all share the same hypothesis Φ.

Measuring security. To compare alternative fixes, we need to measure the security of the alternatives. We view their security as their power to exclude failures. This suggests the basic tenet:

Tenet: A system S_2 is *at least as secure as* S_1, written $S_1 \preceq S_2$, if and only if any attack that is successful against S_2 is also successful against S_1.

Of course, it is difficult to instantiate this tenet in general. It requires a robust and realistic model of the adversary's capabilities, and it requires a clear understanding of what constitutes an attack. We will think of attacks as possible goals that fail, rather than as meaning the specific bad executions that illustrate how they can fail.

Indeed, S_1 and S_2 have to be sufficiently similar to make sense of the "same" attack succeeding on each system. Nevertheless, we believe this tenet can be used to clarify why some security metrics provide more insight than others. Let M be some measurement technique that yields values in a domain ordered by $\leq$. An ideal property to strive for in measuring security is the following.

$$M(S_1) \leq M(S_2) \text{ iff } S_1 \preceq S_2 \quad (1)$$

The crux here is a general philosophy of measurement, according to which the outcome of measurement under the $\leq$ ordering should be a perfect substitute for the $\preceq$ ordering applied directly to the systems. Luce and Suppes [27] refer to this as the Axiom of Order for theories of measurement. In practice, we may have to weaken property (1) in several ways. For instance, we might only achieve the property if we relativize $\preceq$ to a restricted class A of attacks and adversaries yielding a restricted security ordering $\preceq^A$. But it serves to explain why certain metrics do not measure security well. Let's consider a simple example.

The number of lines of code tends to be correlated with the number of bugs in software, and hence to the number of security vulnerabilities. If we let M be a method that counted the number of lines of code and output the negative of that number, then $M(S_1) \leq M(S_2)$ would mean that S_1 has at least as many lines of code as S_2. But it certainly does not follow that S_2 is more secure than S_1. In this instance $M(S_1) \leq M(S_2)$ is only loosely correlated with $S_1 \preceq S_2$, and so the metric is not very reliable.

The lines-of-code metric also has a property that is common to many security metrics proposed in the literature: it takes values in the real numbers. The urge to have *quantitative* metrics is understandable, especially when striving to achieve the rigor of measurement in the physical sciences. Analogies to length, weight, temperature, etc. make quantitative metrics almost irresistible. But our basic tenet shows why this approach cannot work in general. If the $\preceq$ security ordering naturally corresponds to sets of attacks ordered by inclusion, only very rarely will systems be totally ordered by $\preceq$. In many cases S_1 will admit attacks that S_2 does not allow and vice versa, meaning $\preceq$ tends to be only a partial order. Any metric that tries to totally order systems is very likely not to achieve property (1).

This paper represents an instantiation of our general philosophy of measuring security. We recognize that each time there is an attack on a protocol that means there is some (possibly unstated) security goal the protocol does not achieve. Definition 3 in Section 3.3 is a reformulation of our basic tenet.

By *security goal*, we will mean logical formula of a particular form. Roughly speaking, a security goal is an implication $\Phi \Longrightarrow \Psi$ where Φ is a conjunction of atomic formulas, and Ψ is built from atomic formulas by conjunction, disjunction, and existential quantification. (See Def. 1 in Section 3.1). When G is a set of formulas of this form, we will write $\Pi_1 \preceq^G \Pi_2$ to mean that, for every formula $\Gamma \in G$, if Π_1 achieves Γ, then so does Π_2. When G is a singleton, several protocol analysis tools can assist in determining whether $\Pi_1 \preceq^G \Pi_2$.

Enrich-by-need. Enrich-by-need analysis—as in the Cryptographic Protocol Shapes Analyzer (CPSA) [32] or in Scyther [14]—allows us to resolve $\preceq^G$ for some important non-singleton sets G.

Each enrich-by-need analysis process starts from a *scenario* containing some protocol behavior, such as one or more participant's local runs, and also some assumptions about freshness and uncompromised keys. The analysis returns a set of result scenarios that show all of the minimal, essentially different ways that the starting scenario could happen. When the starting scenario is undesirable (e.g. a confidentiality failure), we would like this to be the empty set. When the starting scenario is the behavior of one principal, then the analysis finds what the protocol guarantees from that participant's "point of view." These results indicate what authentication guarantees the protocol provides to that party.

For each run of the analyzer, there is a security goal formalizing the result of the analysis. Its hypothesis Φ describes the content of the starting scenario, and its conclusion is a disjunction. Each disjunct describes the content of one minimal, essentially different execution containing the starting point. Thus, this is a strongest security goal with hypothesis Φ.

Specifically, this goal is strongest in the *implication* ordering, where $\Gamma_2 \Longrightarrow \Gamma_1$ means that Γ_1 is below Γ_2 in this ordering. Enrich-by-need analysis computes maximal elements for certain subclasses G of security goals. Enrich-by-need can act as our measurement technique,

M, satisfying property (1) relative to the security ordering $\preceq^G$. In Section 6.3 we will identify key subclasses G of goals, defined in equation (9), for which each protocol achieves a unique strongest goal (Theorem 2). The maximal elements identified by the enrich-by-need method allow us to measure and compare protocol security relative to $\preceq^G$ (Corollary 1).

Application to standards. A concrete formal language of security goals makes security claims explicit and verifiable. One reason flaws and weaknesses are repeatedly discovered in published standards is that the standards include vague statements, for example about two parties achieving "authentication," without explaining what that means in practice. Basin et al. [5] have illustrated the problems that can arise when standards do not contain any claims about what properties they achieve or under which adversary models. It is thus important for the standardization process to create concrete artifacts that serve as evidence of the security of a protocol design.

Our logical language of security goals supports the goals of [5]. We envision a process where those proposing a protocol are required to include explicit and formal claims about what security properties it achieves. These claims should be explicit enough to allow formal, independent verification that the protocol achieves those security properties. The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) have started to lay the technical groundwork for such a process by publishing a standard (ISO/IEC 29128 [19]) for certifying the design of cryptographic protocols. While this is a good start, it still leaves room for ambiguity based on the specific formalism chosen. Our language is designed to be independent of both the underlying formalism used to represent the protocol and the tool(s) used to verify the claim. This allows a proposal to be verified relative to the same claim by independent experts using different tools.

Despite such efforts to avoid protocol weaknesses early in the standardization process, flaws will still undoubtedly be discovered after publication and deployment. Our development here of a hierarchy of security goals $\preceq^G$ against which to measure security is useful for this purpose. By expressing the flaw as some (previously unstated) goal that is not achieved, standards committees can more easily identify and compare related goals. Having an objective measure of the relative strength of security goals will allow the committee members to separate the security properties from other factors that may affect the feasibility of certain mitigations. The use of enrich-by-need analysis is particularly appropriate in this case, because the analysis can compare the strength of two proposed mitigations relative to an *infinite* set of goals that share some hypothesis Φ. Since the analysis results in the strongest security goal with Φ as a hypothesis, it does not require as much ingenuity on the part of the committee members to identify a particular strengthened security goal. This is an advantage, especially since flaws are frequently already the result of the failure of imagination in the human designers to identify useful security goals.

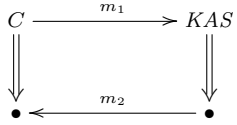
Our contributions. We make three main contributions.

1. We introduce the metrics of security $\preceq^G$ for cryptographic protocols. Each metric is parameterized by some set of desired security goals G.
2. We identify the sets G for which the enrich-by-need analysis method can act as a measurement tool M to compare the security of two or more protocols relative to $\preceq^G$.
3. We ground our theory in protocols that have undergone standardization. We compare our sets G with security goals discussed in the literature. Our methods can guide standards bodies' deliberations on proposed designs and protocol improvements.

Structure of this paper. In order to help the reader gain intuition for when and why it is useful to measure and compare the security of protocols, we begin in Section 2 by considering the Kerberos public key extension PKINIT. The initial version contained a flaw allowing a man-in-the-middle attack. Two alternatives emerged to fix this. We build up some intuition about how to formally express the security goals not met by the flawed version. Section 3 develops our logical language of security goals and explains how sets of goals can serve as security metrics.

Having explained our central ideas, we discuss related work in Section 4. We explain, in depth, the connection between our logical security goals related by implication and well-studied authentication hierarchies. We show how our formalism encompasses and extends that previous work. In Section 5, we show that our logical language helps us to understand how policy considerations that sit outside a protocol proper can contribute to the protocol's security. In Section 6 we explain the enrich-by-need method with examples, and prove that it allows us to compare protocols with regard to some infinite sets of goals. Concluding thoughts are in Section 7.

This paper is an extended version of our [23].



$m_1 = [t_C, n_2]_{\mathrm{sk}(C)}, C, T, n_1$
$m_2 = \{\![k, n_2]_{\mathrm{sk}(S)}\}_{\mathrm{pk}(C)},\ C, TGT,\ \{\!|AK, n_1, t_S, T|\!\}_k$

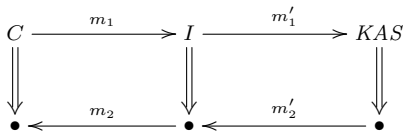
Fig. 1 PKINIT version 25, where $TGT = \{\!|AK, C, t_S|\!\}_{k_T}$

2 Example: Kerberos PKINIT

PKINIT [39] is an extension to Kerberos [30] that allows a client to authenticate to the Kerberos authentication server (KAS) and obtain a ticket-granting ticket using public-key cryptography. This is intended to avoid the management burden of establishing and maintaining user passwords, which the standard Kerberos exchange requires.

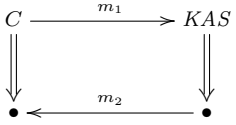
Cervesato et al. [11] found a flaw in PKINIT version 25, which was already widely deployed. The flaw was eventually fixed in version 27. Figure 1 shows the expected message flow between the client and the KAS in v. 25. The client provides a KAS with its identity C, the identity T of the server it would like to access, and a nonce n_1. It also includes a signature over a timestamp t_C and a second nonce n_2 using the client's private key $\mathrm{sk}(C)$. The KAS with identity S replies by creating a fresh session key k, signing it using its key $\mathrm{sk}(S)$ together with the nonce n_2 and encrypting the signature using the client's public key $\mathrm{pk}(C)$. It uses the session key k to protect another session key AK to be used between the client and the subsequent server T, together with the nonce n_1 and an expiration time t_S for the ticket. The ticket TGT is an opaque blob from the client's perspective because it is an encryption using a key shared between S and T. It contains AK, the client's identity C and the expiration time t_S of the ticket.

The flaw. In Cervesato et al.'s attack [11] (Fig. 2), an adversary I has obtained a private key to talk with the KAS S. I uses it to forward any client C's initial request, passing it off as a request from I. I simply replaces C's identity with I's own, re-signing the timestamp and nonce n_2. When the S responds, I re-encrypts the response for C, this time replacing the identity I with C. In the process, the adversary learns the session key k, and thus can also learn the subsequent session key AK. This allows the attacker to read any subsequent communication between the client and the next server T. Moreover, the adversary may impersonate the ticket granting server T to C, and *vice versa*, because



$m_1 = [t_C, n_2]_{\mathrm{sk}(C)},\ \mathbf{C},\ T, n_1$
$m_1' = [t_C, n_2]_{\mathrm{sk}(\mathbf{I})},\ \mathbf{I},\ T, n_1$
$m_2 = \{\![k, n_2]_{\mathrm{sk}(S)}\}_{\mathrm{pk}(\mathbf{C})},\ \mathbf{C},\ TGT, \{\!|AK, n_1, t_S, T|\!\}_k$
$m_2' = \{\![k, n_2]_{\mathrm{sk}(S)}\}_{\mathrm{pk}(\mathbf{I})},\ \mathbf{I},\ TGT, \{\!|AK, n_1, t_S, T|\!\}_k$

Fig. 2 Attack on flawed PKINIT, where $TGT = \{\!|AK, I, t_S|\!\}_{k_T}$



$m_1 = [t_C, n_2]_{\mathrm{sk}(C)}, C, T, n_1$
$m_2 = \{\![k, \mathbf{F(C, n_2)}]_{\mathrm{sk}(S)}\}_{\mathrm{pk}(C)},\ C, TGT,\ \{\!|AK, n_1, t_S, T|\!\}_k$

Fig. 3 Generic fix for PKINIT

the participants rely on the protocol to ensure that they are the only entities with knowledge of AK.

The attack arises from a lack of cryptographic binding between the session key k, and the client's identity C [11]. After C's two-message exchange, she knows the KAS produced the keying material k recently, because of its binding with n_2. However, the KAS S may not have intended k for C.

The protocol revision process. Since this absence of C's identity is the root cause of the attack, the natural fix is to include C as an additional field in the signed portion of the second message. Indeed this is the first suggestion in [11].

The authors of the PKINIT standard offered a different suggestion. For operational feasibility—namely, to preserve the previous message format—more than security, the PKINIT authors suggested replacing n_2 with a message authentication code over the entirety of the first message, keying the MAC with k. Since the client's identity is contained in the first message, this proposal also creates the necessary cryptographic binding between k and C, as well as with n_2.

Cervesato et al. used a manual proof method to verify a generic scheme for mitigating the attack (Fig. 3), ensuring that the two proposals were instances of the scheme. This allowed them to avoid the time-consuming process of writing proofs for any other proposals that might also fit this scheme. They verified that the attack fails if n_2 is replaced with any expression $F(C, n_2, \ldots)$ that is *injective* on C and n_2; that is, $F(C, n_2, \ldots) = F(C', n_2', \ldots)$ implies $C = C'$ and $n_2 = n_2'$.

We obtain the first proposal by instantiating F as concatenation: $F(C, n_2) = C, n_2$. The second proposal instantiates F as the MAC of the client's request:

$$F(C, n_2, \ldots) = H_k([t_C, n_2]_{\mathrm{sk}(C)}, C, T, n_1).$$

Since the MAC provides second preimage resistance, the injectivity requirement will hold, with overwhelming probability, in any execution the adversary can engineer.

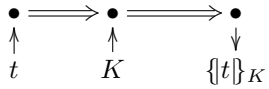
Security goals. The PKINIT parable illustrates recurring themes in developing and maintaining protocol standards. An attack often shows us that we care about previously unstated and unrecognized security goals as observed by Basin et al. [5]. PKINIT achieves some level of authentication, but it fails a more stringent type of authentication. In Lowe's terms [26], it achieves *recent aliveness* both for the client and for the *KAS* S, because each party signs time-dependent data. However, PKINIT does not achieve weak agreement, since C does not know that S was engaged in the protocol *with* C. The attack helps us to express the goal that the flawed protocol does not meet.

But an attack itself does not uniquely identify a security goal. We learned that it is important for the client to be guaranteed that it agrees with S on the client's identity, but what about other values such as the expiration time of the ticket? Operational difficulties might arise if the client is unaware of this expiration time, but are there any security consequences? Indeed a key contribution of [11] is to state carefully what security goal the repair provides.

This goal can be achieved in different ways. Different stakeholders may prefer different mitigations because of issues of efficiency, ease of deployment, or robustness to future modification. In PKINIT, the researchers opted for a change that was minimally invasive to their formal representation, thereby highlighting the root cause of the problem. The protocol designers had more operational context to constrain the types of solutions they deemed feasible.

While a pair of choices might both manage to satisfy some stated security goal, one of them may actually satisfy strictly stronger goals than another. We propose a goal language (Section 3) to express when a protocol mitigation is at least as good as a competitor—or strictly better than it—from the security point of view.

Strand spaces. We will develop our ideas using the strand space terminology [37, 20]. A *strand* is a sequence of transmission and reception events, each of which we will call a *node*. We use strands to represent the behavior of a single principal in a single local protocol session. We call these *regular strands*. We also use strands to represent the basic abilities of the adversary. For instance, a strand representing the adversary's ability to encrypt contains two reception nodes, in which the plaintext t and the encryption key K are received, followed by a transmission node in which the encryption is sent:



By convention, we draw strands with double-arrows connecting the successive nodes $\bullet \Rightarrow \bullet$, and single arrows indicating the message flow $\bullet \rightarrow \bullet$.

In this framework an execution is a kind of directed graph with these two kinds of edges. These graphs are *bundles*, meaning all finite directed acyclic graphs where (i) the nodes at the two ends of a message transmission arrow are labeled with the same message; (ii) each reception node has exactly one incoming message; and (iii) when a node on a strand is included in the graph, then so are all its predecessors on that strand. However, a bundle does not have to run all of the strands "to completion," and it may therefore contain only an initial segment of the nodes of that strand.

A *protocol* Π is a finite set of strands, called the *roles* of Π, together with possibly some auxiliary assumptions about fresh and non-compromised values. The messages sent and received on these strands contain *parameters* such as $C, S, AK, n_1, n_2, \ldots$ in PKINIT. The *regular strands of* Π consist of all strands obtained from the roles of Π by applying substitutions to these parameters. A Π-bundle is a bundle where every strand is either an adversary strand or a regular strand of the protocol Π.

Fig. 2 becomes a PKINIT-bundle when we expand the single central strand labeled I into a collection of adversary strands. We consider Fig. 2 an informal shorthand for the resulting bundle.

Formalizing the authentication goal. The attack of [11] undermines what the client C should know when he has completed a local run of PKINIT. The client knows less about what the *KAS* S has done than expected.

The actions of the regular (non-adversary) principals are *message transmission nodes* and *message reception nodes*. We will formulate the client's expectation about the *KAS*'s behavior as a formula about the transmission and reception nodes of the principals.

The formula applies in a situation, depicted in (2), where there is a reception node n which completes a run

of the client role, which we will write $\texttt{ClientDone}(n)$.

$$\mathsf{sk}(S) \in \mathsf{non} \qquad \begin{array}{l} \bullet \longrightarrow \\ {\scriptstyle s_1}\Downarrow \\ n \longleftarrow \end{array} \tag{2}$$

$$s_1 \in \mathsf{Client}[C, S, \_, \_, \_, \_, \_, \_, \_]$$

This node n belongs to a run of the protocol in which the active principal has some identity C, and its intended peer is a Kerberos Authentication Server S. We will write this $\texttt{Self}(n, C) \wedge \texttt{Peer}(n, S)$. Of course, if S's signature key $\mathsf{sk}(S)$ is compromised, then C learns nothing from a run of the protocol. Thus, we will assume that it is uncompromised, written as $\texttt{Non}(\texttt{sk}(S))$. For the moment, we ignore the other parameters of the Client role, since we will not assume anything about them.

We regard these formulas as forming a hypothesis, when combined by conjunction. Thus, we would like to understand what must be true when this hypothesis holds:

$$\begin{aligned} &\texttt{ClientDone}(n) \wedge \texttt{Self}(n, C) \\ &\qquad \wedge \texttt{Peer}(n, S) \wedge \texttt{Non}(\texttt{sk}(S)) \end{aligned} \tag{3}$$

In the attack, there is a local run of the KAS role, and in fact the server has the intended identity S. The problem is that the server's intended peer is not the client C, but some compromised client I. Thus, the behavior in Fig. 2 is a counterexample to the goal:

$$\begin{aligned} \forall n, C, S\,.\,(3) \Longrightarrow \exists m\,.\,&\texttt{KASDone}(m) \\ \wedge\ \ &\texttt{Self}(m, S) \wedge \texttt{Peer}(m, C) \end{aligned} \tag{4}$$

where the whole of formula (3) is the hypothesis, although we have contracted it to save space. Here also we write $\texttt{KASDone}(m)$ to indicate that the transmission node m is the final node of the KAS's role. We again use `Self` to refer to the identity of the participant enacting this role, and `Peer` to refer to its intended partner.

Clearly this is a weak goal, since it says nothing about other parameters such as the nonces n_1, n_2, the session key k, or the server T that the ticket will be shared with. But Fig. 2 is also a counterexample to all its stronger goals.

A curious fact about this formalism is that it says nothing about the specific messages sent. It talks about the nodes, such as n, m, and asserts that they lie on a particular role at a specific position in the sequence of nodes in which the role engages. It talks about the parameters associated with the nodes. For instance, the `Peer` parameter of n is the same identity S whose signature key is assumed to be uncompromised. Moreover, this is the same as the `Self` parameter of some node m, and that m is a `KASDone` node.

However, it never assumes or asserts that the messages formed from the parameters have a particular layout or structure. This means that the same formula can describe executions in different protocols.

For instance, it is clear how to interpret formula (4) in the two revised versions of PKINIT. There is a self-explanatory convention that links protocols in PKINIT v. 25 to roles in its two candidate successors, and similarly for their parameters. When the protocols are more remotely related, Guttman's formal notion of translation applies [22]. Our convention allows us to use the identity translation a large range of cases.

Formalizing a non-disclosure goal. We can represent secrecy goals in a similar style, using a special auxiliary role which we can assume belongs to all protocols implicitly. This is the "listener role" that consists of a single reception node. It has a single parameter x, and the message received on this node is x. It represents the assumption that x is compromised, and observed unprotected. We write $\texttt{Lsn}(n)$ to express that n is the reception node lying on an instance of the listener role. We write $\texttt{Hear}(n, x)$ to express that the message heard on node n is x, i.e. to stipulate a value for the parameter of the role. Thus, consider the assumption that adds a listener to formula (3):

$$\begin{aligned} &\texttt{ClientDone}(n) \wedge \texttt{Self}(n, C) \wedge \texttt{SessKey}(n, k) \\ &\qquad \wedge \texttt{Peer}(n, S) \wedge \texttt{Non}(\texttt{sk}(S)) \\ &\qquad \wedge \texttt{Lsn}(m) \wedge \texttt{Hear}(m, k) \end{aligned} \tag{5}$$

Here we are also using the predicate $\texttt{SessKey}(n, k)$ to refer to the session key parameter of the client's final reception node. The formula asserts that the same value k is also heard unprotected on the listener node m. This formula represents a situation visualized in (6).

$$\mathsf{sk}(S) \in \mathsf{non} \qquad \begin{array}{lll} C \longrightarrow & & \\ {\scriptstyle s_1}\Downarrow & & \\ n \longleftarrow & \xrightarrow{k} & m \end{array} \tag{6}$$

$$s_1 \in \mathsf{Client}[C, S, \_, \_, \_, \_, \_, k, \_]$$

For the protocol to ensure secrecy for k in this situation means that this situation should never be able to arise. Here we interpret secrecy failures as meaning the full disclosure of a secret. This is coarser than the standard cryptographic definition, which refers to any ability of the adversary to distinguish the secret from a random value [6]. Formalizing secrecy as full disclosure, the security goal would be:

$$\forall n, m, C, S, k\,.\,(5) \Longrightarrow \texttt{false} \tag{7}$$

Unfortunately, the adversary can extract k from m_2' in the run shown in Fig. 2. Thus, Fig. 2 illustrates why this goal fails: The adversary has the power to transmit k so that it will be heard on a listener node.

Thus, both non-disclosure and authentication goals are expressible using these ideas.

3 Protocol Goals to Measure Security

As we have just illustrated, we express protocol goals as formulas in first order logic. For each protocol Π, there is a goal language $\mathcal{GL}(\Pi)$; however, these languages are designed so that for related protocols, the languages can be similar or often identical. This helps when comparing the goals achieved by related protocols.

3.1 The Goal Languages

The goal language is designed to have the minimum possible expressiveness while remaining useful. It contains no arithmetic; it contains no inductively defined data-types such as terms; and it has no ability to describe the syntax of messages.

This is an important and useful feature of the goal language: goals are much easier to view as logical objects that exist independent of a particular protocol when they do not reference the syntax or structure of messages. The ability to view goals independently from individual protocols is essential when trying to compare different protocols in terms of the goals they achieve. This reduced expressiveness has proved useful in prior work. Formulas in $\mathcal{GL}(\Pi)$ are preserved under a class of "security preserving" transformations between protocols [22]. Also, for an interesting restricted class of protocols, Dougherty and Guttman showed the set of security goals they achieve is decidable [16]. Mödersheim et al. [2] show how to adapt another analysis approach to this class of goals.

Predicates in the goal language. We will describe the goal language $\mathcal{GL}(\Pi)$ associated with a given protocol Π. Although there is a connection between $\mathcal{GL}(\Pi)$ and Π, our intention is to be able to describe logical sentences that apply both to Π and to variants of Π.

For each node in a role of Π, the goal language $\mathcal{GL}(\Pi)$ has an associated *role position predicate*. The two predicates $\texttt{ClientDone}(n)$ and $\texttt{KASDone}(m)$ used above are examples. Each role position predicate is a one-place predicate that says what kind of node its argument n, m refers to.

On each node, there are parameters. The *parameter predicates* are two place predicates. Each parameter predicate associates a node with one of the values that has been selected when that node occurs. For instance, $\texttt{Self}(n, c)$ asserts that the *self* parameter of n is c. This allows us to assert agreement between different strands.

Functions:	$\texttt{pk}(a)$	$\texttt{sk}(a)$	$\texttt{inv}(k)$
	$\texttt{ltk}(a, b)$		
Relations:	$\texttt{Preceq}(m, n)$	$\texttt{Coll}(m, n)$	$=$
	$\texttt{Unq}(v)$	$\texttt{UnqAt}(n, v)$	$\texttt{Non}(v)$

Table 1 Protocol-independent vocabulary of languages $\mathcal{GL}(\Pi)$

$\texttt{Peer}(m, c)$ asserts that m appears to be partnered with c, which is the same principal who is in fact the *self* parameter of n.

The role position predicates and parameter predicates vary from protocol to protocol, depending on how many nodes the protocol has, and how many parameters. However, when we regard two protocols as variants of each other, we expect a certain amount of overlap between the associated names. For instance, the $\texttt{ClientDone}(n)$ predicate in the example should have a meaning (i.e. a node that satisfies it) in any variant of the PKINIT protocol. Furthermore, we expect that the satisfying nodes represent informally corresponding activity in the two variants.

Beside role position predicates and parameter predicates, $\mathcal{GL}(\Pi)$ has the protocol-independent vocabulary in Table 1. It helps to express the structural properties of bundles. $\texttt{Preceq}(m, n)$ asserts that either m and n represent the same node, or else m occurs before n; $\texttt{Coll}(m, n)$ says that they lie on the same strand. $m = n$ is satisfied when m and n are equal. $\texttt{Non}(v)$ and $\texttt{Unq}(v)$ express non-compromise and freshness (unique origination), and $\texttt{UnqAt}(n, v)$ identifies the node at which v is assumed fresh. $\texttt{pk}(a)$ and $\texttt{sk}(a)$ relate a principal a to its keys, $\texttt{ltk}(a, b)$ represents the long-term shared key of two principals a, b, and $\texttt{inv}(k)$ is the inverse of a key.

We have only presented here an informal idea of how to interpret formulas in this language. For a full, formal description of the semantics of the satisfaction relation $\models$ see [22].

The examples of the last section illustrate how we can use the vocabulary of $\mathcal{GL}(\Pi)$ to express a variety of security goals. These include authentication goals and—since we assume that any protocol Π contains the listener role—non-disclosure goals as well.

Goals. The formulas that we used in our examples have a special form. They are implications. Their hypotheses are conjunctions of atomic formulas of $\mathcal{GL}(\Pi)$. The conclusions took two superficially different forms. Formula (4) has an existential formula as its conclusion. It asserts that an additional event exists, satisfying a particular role position predicate, and with some parameters matching those in the hypothesis. Formula (7) has the conclusion $\texttt{false}$. A conclusion could also be a disjunction, where the protocol allows the behavior assumed in the hypothesis to be explained in a number

of different ways. For instance, the *KAS* server may have executed either the role shown in Fig. 1, or else a different role in which it first retrieves a public-key certificate for C, and then replies with the message shown in Fig. 1. Since we may regard `false` as the degenerate disjunction with zero disjuncts, and the conclusion of formula (4) as a degenerate disjunction with one disjunct, we regard them all as having n-ary disjunctions as conclusions. Since the goals are intended to hold in all cases, we regard any variables free in the whole formula as implicitly universally quantified. Thus, we stipulate:

Definition 1 A *security goal* (or sometimes simply a *goal*) is a closed formula $\Gamma \in \mathcal{GL}(\Pi)$ of the form

$$\forall \overline{x} \,.\, (\Phi \implies \bigvee_{1 \leq j \leq i} \exists \overline{y}_j \,.\, \Psi_j) \tag{8}$$

where Φ and Ψ_j are conjunctions of atomic formulas. We write

$\mathsf{hyp}(\Gamma) = \Phi$, and
$\mathsf{conc}(\Gamma) = \bigvee_{1 \leq j \leq i} \exists \overline{y}_j \,.\, \Psi_j$.

We assume that the existentially bound variables in $\overline{y}_j$ are distinct from all variables in $\mathsf{hyp}(\Gamma)$; this is no loss of generality, since we can rename bound variables.

Non-disclosure goals here are the special case in which the disjunction is empty since the upper index is 0.

We propose to express the security services that protocols provide by a set of formulas of the form (8). These so-called *geometric sequents* are natural to express security goals. Each enumerates some finite number of facts that serve as the hypothesis. Then, the claim in the conclusion is that one of zero or more alternatives holds, where each of these is a finite number of facts that should also be found to hold. Thus, the claim is localized, and independent of the totality of behavior in the world of users of the protocol. This is quite appropriate for security properties.

We do not formalize here indistinguishability properties, which are properties of pairs of runs, or probabilistic properties, which focus on the distributions governing runs.

3.2 Measuring a Protocol with Goal Formulas

Security goals admit a natural partial order based on implication: $\Gamma_1 \leq \Gamma_2$ iff $\Gamma_2 \Rightarrow \Gamma_1$. Stronger goals are higher in the partial order. We can use this partial order as a measure of the security of a protocol in the following way.

Definition 2 A protocol Π *achieves* security goal Γ iff for every bundle $\mathcal{B}$ of Π, $\mathcal{B} \models \Gamma$.

This provides our formal basis for deciding whether a protocol is "good enough" for a given purpose. An immediate consequence of this definition is that achieving goals is downward closed in the partial order. More formally:

Lemma 1 *If Π achieves Γ' and $\Gamma \leq \Gamma'$, then Π also achieves Γ.*

Thus, if Π achieves some goal Γ, it also achieves any other goal that is a consequence of Γ.

Def. 2 is intuitively clear. However, it quantifies over all bundles $\mathcal{B}$. This makes it look like a daunting definition to verify in practice. Although this problem is known to be undecidable in general [17], there is a semi-decision procedure for finding counterexamples.

Theorem 1 *There is a semi-algorithm to find a bundle $\mathcal{B}$ of Π such that $\mathcal{B} \not\models \Gamma$, if any exists.*

Proof Let $\mathsf{conc}(\Gamma)$ be $\bigvee_{1 \leq j \leq i} \exists \overline{y}_j \,.\, \Psi_j$. Since Π-bundles are finite structures, we can enumerate them. For each $\mathcal{B}$, again because it is a finite structure, there are at most finitely many variable assignments η such that $\mathcal{B} \models_\eta \mathsf{hyp}(\Gamma)$.

For each such η and each j such that $1 \leq j \leq i$, there are only finitely many ways to extend η to assign values to the variables $\overline{y}_j$ in $\mathcal{B}$. Thus, we can determine whether η satisfies $\exists \overline{y}_j \,.\, \Psi_j$. □

Of course, the method detailed in the proof above is naively inefficient. Numerous tools exist that implement much more efficient algorithms to achieve the same result. Throughout this paper we use our tool CPSA to implement this check.

We would like to apply these ideas to PKINIT v. 25 and the two proposed mitigations from Section 2. Let us use PKINIT_1 to denote the fix in which $F(C, n_2) = (C, n_2)$, and use PKINIT_2 to denote the other fix in which $F(C, n_2) = H_k([t_C, n_2]_{\mathrm{sk}(C)}, C, T, n_1)$. In the previous section we identified formula (4) as the security goal that is not achieved in PKINIT v. 25. Call this formula Γ. Cervesato et al. prove by hand that both PKINIT_1 and PKINIT_2 achieve Γ, so it should be no surprise that applying CPSA to those versions confirms the result. Thus both of these fixes are good enough for the newly described goal Γ.

3.3 Comparing Protocols

We want to use security goals not only to measure a protocol against a given goal, but also to measure the relative strength of two or more protocols Π_i against each other. We will be able to do this, but only when

the role position predicates and parameter predicates have a well-defined semantics for each Π_i. This ensures that if Γ is a goal then we can ask whether $\mathcal{B} \models \Gamma$ for any bundle $\mathcal{B}$ of any of the protocols Π_i.

There is a certain amount of freedom in choosing the names of these predicates, and the results of the comparisons will depend on the names chosen. We may arrive at strange conclusions if, for example, in PKINIT_1, $\texttt{ClientDone}(\cdot)$ is satisfied by the last node on a client strand, but in PKINIT_2 it is satisfied by the last node of a server strand. There is a natural way to identify the nodes and parameters of one protocol with those of its variants when there are protocol transformations [22] between them. For example, all the variants of PKINIT have corresponding nodes and parameters. The only thing that changes is the exact structure of the messages. [22] formalizes what it means for the semantics of a language to respect these identifications. For the remainder of this section we assume our security goals are expressed in a goal language where the predicate names are uniformly chosen for each of the protocols Π_i, respecting the natural identifications of nodes and parameters.

The partial order $\leq$ on goals naturally induces a partial order $\trianglelefteq$ on protocols in the following way.

Definition 3 A protocol Π_2 is *at least as strong as* a protocol Π_1, written $\Pi_1 \trianglelefteq \Pi_2$, iff for every goal Γ, if Π_1 achieves Γ then so does Π_2.

For the set of all goals that the protocol Π achieves, we write:

$$\mathsf{Ach}(\Pi) = \{\Gamma \mid \Pi \text{ achieves } \Gamma\}.$$

The position of each Π in the partial order $\trianglelefteq$ is determined by the set $\mathsf{Ach}(\Pi)$ of goals it achieves. Then $\Pi_1 \trianglelefteq \Pi_2$ iff $\mathsf{Ach}(\Pi_1) \subseteq \mathsf{Ach}(\Pi_2)$. So $\trianglelefteq$ mirrors the partial order on sets of goals ordered by inclusion. Thus, the $\trianglelefteq$ order justifies us in regarding Ach as a measure of protocol strength, as in our Tenet for measurement (cf. Eq. 1). By Lemma 1, the sets $\mathsf{Ach}(\Pi)$ are closed under logical implication.

Definition 3 is a very strong condition because it accounts for *every* goal. Indeed, the condition may be too strong for practical use. For one thing, it is not clear how one would verify that $\Pi_1 \trianglelefteq \Pi_2$. That would require a structured way to consider every goal, or at least an efficient way of calculating $\mathsf{Ach}(\Pi)$. Also, protocols are typically designed with a small, finite number of goals in mind. It may be sufficient to understand the relative behavior of Π_1 and Π_2 on a finite set of design goals G. This suggests relativizing the partial order $\trianglelefteq$ to a set G of goals of interest.

Definition 4 A protocol Π_2 is *at least as strong as* protocol Π_1 *under* G, written $\Pi_1 \trianglelefteq^G \Pi_2$, iff for every goal $\Gamma \in G$, if Π_1 achieves Γ then so does Π_2.

When $\Pi_1 \trianglelefteq^G \Pi_2$ and $\Pi_2 \trianglelefteq^G \Pi_1$ we write $\Pi_1 \bowtie^G \Pi_2$.
When $\Pi_1 \trianglelefteq^G \Pi_2$ and $\Pi_2 \not\trianglelefteq^G \Pi_1$ we write $\Pi_1 \triangleleft^G \Pi_2$.

We want to be clever in choosing sets of goals G so that (1) it is straightforward to verify $\Pi_1 \trianglelefteq^G \Pi_2$, and (2) $\trianglelefteq^G$ usefully distinguishes as many protocols as possible. The first condition pushes us to consider smaller sets, while the second condition pushes us to consider larger sets. In fact:

Lemma 2 *$\Pi_1 \bowtie^G \Pi_2$ and $G' \subseteq G$ implies $\Pi_1 \bowtie^{G'} \Pi_2$.*

3.4 Comparing Protocols by Finite Sets

The smallest useful set is a singleton $G = \{\Gamma\}$. In this case the induced partial order $\trianglelefteq^{\{\Gamma\}}$ is actually a two-point total order on protocols. Each protocol either achieves Γ or not. If we let $\Gamma_{(4)}$ be the PKINIT authentication goal in formula (4), then

$$\text{PKINIT} \triangleleft^{\{\Gamma_{(4)}\}} \text{PKINIT}_1 \bowtie^{\{\Gamma_{(4)}\}} \text{PKINIT}_2,$$

because PKINIT does not achieve the goal but both fixes do. Similarly, for the non-disclosure goal $\Gamma_{(7)}$ in formula (7),

$$\text{PKINIT} \triangleleft^{\{\Gamma_{(7)}\}} \text{PKINIT}_1 \bowtie^{\{\Gamma_{(7)}\}} \text{PKINIT}_2.$$

Thus, the two fixes agree on $\{\Gamma_{(4)}, \Gamma_{(7)}\}$, i.e.

$$\text{PKINIT}_1 \bowtie^{\{\Gamma_{(4)}, \Gamma_{(7)}\}} \text{PKINIT}_2.$$

Larger finite sets G may induce more interesting partial orders. When G represents the set of desired goals, it is the job of the protocol designer to define a protocol that achieves all the goals in G so that $G \subseteq \mathsf{Ach}(\Pi)$. As standards committees propose fixes to protocols, however, cases may arise in which two proposals Π_1 and Π_2 are incomparable via $\trianglelefteq^G$. Such cases provide an opportunity for standards committees to discuss the relative importance of goals in G. It may be worthwhile to sacrifice some goals in order to achieve others. Recognizing that two protocols are incomparable in the partial order defined by the goals allows the trade-offs to be much clearer.

4 Related Work

There are several approaches to measuring the security of systems. One approach focuses on establishing a set of standardized names for common concepts, languages for expressing and sharing information about

those concepts, and system administrator tools to automatically collect measurements of a system with results expressible in those languages. Works exemplified by Martin [28], Sun et al. [36], and Liu et al. [25] demonstrate ways to leverage common enumerations such as CVE [12] and CWE [13] in automatically generating security metrics for the analysis of operational systems. Our approach differs in several crucial aspects. By focusing on the relatively narrow domain of cryptographic protocols, we can more easily develop a general and formal language of events in which to express security properties. We are thus able to provide a clear semantics for formulas in our language that corresponds to the effects of attacks without enumerating those attacks explicitly.

4.1 Protocol Security Hierarchies

More closely related to our work are approaches to analyzing security protocols relative to specific definitions of protocol security. There have been many attempts in the literature to define protocol security [6, 38, 34, 26, 9, 10, 1] and there has been no consensus on a formal definition of what is meant by that term. This is for good reason: Security comes in various forms and strengths. Some varieties may be suitable for one purpose, but insufficient for another.

This point was clearly made by Lowe in [26] when he defined a hierarchy of authentication specifications ranging from aliveness to injective agreement on a set of values. Since then, Cremers and Mauw [14] have amended and extended this hierarchy to capture synchronization properties (reminiscent of properties defined in Woo and Lam's [38] and Roscoe's [34]) and whether or not some party's peer is running the expected role. In [4], Basin and Cremers identify a hierarchy of adversary models and derive a hierarchy of protocols according to the strongest adversary model under which the protocols are secure. Their adversary models come mostly from Canetti and Krawczyk's [9, 10]. Security in this case is with respect to a small, fixed set of well-defined secrecy and authentication goals.

We view the present work as a step in the direction of unifying, simplifying, and extending the related work on hierarchies of specifications, adversary models, and protocols. Our security goal language is designed to express properties in a manner that is independent of the underlying formalisms or tools used to verify them. The structure of goals as first order formulas makes the relative strength of security goals clear and immediate. The language is expressive enough to capture the natural notions of authentication and secrecy used by others. Judicious use of the atomic predicates $\texttt{Non}(v)$, $\texttt{Unq}(v)$, and $\texttt{Preceq}(m, n)$ can also express subtle limits on the adversary's ability to compromise both long-term and short-term data. These are the dimensions along which Basin and Cremers vary their adversary models in [4]. Thus we can also incorporate a variety of adversary models into the specifications themselves, which Basin and Cremers identify as an alternative approach to theirs.

Our aim in the rest of this section is to elaborate the detailed connection with the related work on protocol security hierarchies. By recreating the authentication hierarchy as defined by Lowe [26] and extended by Cremers and Mauw [14], we hope the reader will gain a stronger intuition for how to uniformly express a wide variety of important security goals drawn from the literature.

4.2 Recreating a Hierarchy of Authentication Goals

Lowe begins his investigation by defining four types of authentication, *weak aliveness, weak agreement, non-injective agreement,* and *injective agreement.*[1] They are all stated from the perspective of an agent A playing the initiator role of a protocol trying to authenticate another agent B playing the responder role. They each assert the existence of some protocol event given that some behavior has occurred. There is a clear ordering of strength among them because they demand increasingly more agreement between A and B. Lowe notes that the restriction to two party protocols and to authentication of a responder by an initiator is incidental. The definitions naturally generalize to reversing the order of authentication and to multi-party protocols.

We now express each of these properties in our goal language. Although our language is independent of the particular protocol being considered, it does require role position predicates for each node and parameter predicates for the values at those nodes. Thus the actual formalization of these authentication goals will depend to some degree on the protocol. In order to remain as general as possible, we assume a protocol which has an initiator role and a responder role. The role position predicates $\texttt{IStart}(\cdot), \texttt{RStart}(\cdot)$ will be satisfied by the first nodes of the initiator and responder roles respectively. Similarly $\texttt{IDone}(\cdot)$ and $\texttt{RDone}(\cdot)$ will be satisfied by their final nodes. We assume that, at the start of each role, parameters for its own identity and that of its peer are well defined. We use the parameter predicates $\texttt{Self}(\cdot, \cdot)$ and $\texttt{Peer}(\cdot, \cdot)$ to represent these parameters.

[1] We use the terminology of Cremers and Mauw's [14] instead of [26] because it makes finer distinctions that are useful for our purposes.

All other parameters p are represented by predicates $\texttt{Param}_p(\cdot,\cdot)$.

Most protocol goals are trivially broken when the participants use compromised long-term keys or credentials. Exactly which keys must be kept secure to achieve certain goals will depend on the protocol. These assumptions about uncompromised keys can be expressed in our logic by naming the keys with $\texttt{Param}_k(n, v_k)$ and asserting they are unavailable to the adversary with $\texttt{Non}(v_k)$. We may use the $\texttt{Self}(n, A)$ or $\texttt{Peer}(n, A)$ parameter predicate with $\texttt{sk}$ in $\texttt{Non}(\texttt{sk}(A))$. We use the notation $\texttt{GoodKeys}(n, \overline{k})$ to represent a conjunction of such formulas, expressing that the relevant keys (represented by the parameters $\overline{k}$) are not compromised. Our goals are thus somewhat parametric in which keys are covered in this formula.

The stronger authentication goals can naturally be expressed by modifying the weaker ones. To avoid typesetting large formulas that are difficult to read, we will define new formulas in terms of the parts of previous ones. Each goal has the form shown in Definition 1. We use the convention that for goal Γ_i the conjunction on the left of the implication is denoted by Φ_i. Each of the disjuncts on the right of the implication is denoted by Ψ_i^j for j between 1 and the number of disjuncts. Ψ_i^j includes the existential quantifier. Thus, authentication goal Γ_i is expressible as $\Phi_i \Rightarrow \bigvee_{1 \le j \le n} \Psi_i^j$ where any remaining free variables are implicitly universally quantified at the start of the formula. We build up larger formulas by adding conjuncts to these parts and capturing new variables under new existential quantifiers when needed.

Expressing the goals. We start by expressing the goals in Lowe's hierarchy in our formalism. For the following discussion, we direct the reader to Table 2 which summarizes the results. Weak aliveness, Γ_1, is the perhaps the weakest meaningful form of entity authentication. A protocol that satisfies weak aliveness guarantees that the intended peer started the protocol at some time in the past. It does not guarantee that the peer agrees on the initiator's identity, nor does it guarantee that the the peer is acting in the right role. The next property Γ_2, weak agreement, specifies that the peer must also agree on the initiator's identity. This is done by adding the relevant parameter predicates to the hypothesis Φ_1 and to each of the disjuncts Ψ_1^i.

Non-injective agreement requires the peer to be acting in the correct role (i.e. as a responder) and it requires that the two parties agree on some subset V of parameters used in their roles. We express non-injective agreement as Γ_3 which we obtain from Γ_2 in two steps. First we remove the disjunct Ψ_2^2 from the conclusion that allows the peer to act as an initiator. Then, for each $p \in V$, we conjoin the corresponding parameter predicate $\texttt{Param}_p(\cdot,\cdot)$ to both the hypothesis and conclusion, ensuring the variables in the second argument are the same in both places.

The injective-agreement property is designed to ensure that a protocol is resistant to replay attacks in which an adversary may record messages from a previous successful session and use them at a later time. This can be avoided if the protocol ensures that each set of values the initiator commits to is unique to the current session. This *injective session property* is formalized as Γ_*. Injective agreement, Γ_4, is then the conjunction of Γ_3 with Γ_*. It is possible to express injective agreement as a single goal, Γ_4', but we believe it is more informative to demonstrate the logical independence of agreement and injectivity.

Lowe points out in [26] that properties Γ_1 through Γ_4 do not capture the notion of recentness. In many protocols the mechanisms to ensure the peer has been recently active are similar to those used to ensure the injective session property. Namely, random challenge nonces are used to ensure both that each session has some unique input and that the peer's activities have occurred after the creation of the nonce. However, recentness and injectivity are logically independent. One is about the relative ordering of events, while the other is about the uniqueness of sessions. For each of the properties defined so far, Lowe defines another version that ensures recentness.

Recentness requires the existence of an event that is known to have occurred not too long ago. This could be a previous event performed by the initiator, or it could be an external event such as a clock tick. We remain agnostic about which event is used as a time reference. We simply assume we know what event in the protocol is sufficient for this purpose and we say that such a node satisfies the role position predicate $\texttt{TimeRef}(\cdot)$. For each Γ_i we obtain a version that requires recentness by modifying each of its disjuncts Ψ_i^j, adding the two conjuncts $\texttt{TimeRef}(m') \wedge \texttt{Preceq}(m', m)$. For example, we modify Γ_1, weak aliveness, into Γ_5, recent weak aliveness. Modifying Γ_2, Γ_3 and Γ_4 analogously, yields respectively

Γ_6, recent weak agreement;
Γ_7, recent non-injective agreement; and
Γ_8, recent injective agreement.

We omit these from Table 2 since they are completely analogous to Γ_5.

In [14], Cremers and Mauw augment this hierarchy in two ways, which we formalize in Table 3. First, they choose to modify Γ_1 not by ensuring agreement on both identities, but by first ensuring that the peer is acting

Weak aliveness.

$$\left(\begin{array}{l}\texttt{IDone}(n) \wedge \texttt{Peer}(n,r) \wedge \\ \texttt{GoodKeys}(n,\overline{k})\end{array}\right) \implies \left(\begin{array}{l}(\exists m\,.\,\texttt{RStart}(m) \wedge \texttt{Self}(m,r)) \vee \\ (\exists m\,.\,\texttt{IStart}(m) \wedge \texttt{Self}(m,r))\end{array}\right) \qquad (\Gamma_1)$$

Weak agreement.

$$\Phi_1 \wedge \texttt{Self}(n,i) \implies (\Psi_1^1 \wedge \texttt{Peer}(m,i)) \vee (\Psi_1^2 \wedge \texttt{Peer}(m,i)) \qquad (\Gamma_2)$$

Weak agreement: Variant.

$$\Phi_1 \implies \left(\begin{array}{l}(\exists i\,.\,\Psi_1^1 \wedge \texttt{Self}(n,i) \wedge \texttt{Peer}(m,i)) \vee \\ (\exists i\,.\,\Psi_1^2 \wedge \texttt{Self}(n,i) \wedge \texttt{Peer}(m,i))\end{array}\right) \qquad (\Gamma_2')$$

Non-injective agreement.

$$\Phi_2 \wedge \textstyle\bigwedge_{p \in V} \texttt{Param}_p(n, v_p) \implies \Psi_2^1 \wedge \bigwedge_{p \in V} \texttt{Param}_p(m, v_p) \qquad (\Gamma_3)$$

Injective session.

$$\left(\begin{array}{l}\texttt{IDone}(n_1) \wedge \bigwedge_{p \in P(init)} \texttt{Param}_p(n_1, v_p) \wedge \\ \texttt{IDone}(n_2) \wedge \bigwedge_{p \in P(init)} \texttt{Param}_p(n_2, v_p)\end{array}\right) \implies n_1 = n_2 \qquad (\Gamma_*)$$

Injective agreement.

$$\Gamma_3 \quad \wedge \quad \Gamma_* \qquad (\Gamma_4)$$

Injective agreement: Variant.

$$\left(\begin{array}{l}\Phi_3 \wedge \texttt{IDone}(n') \wedge \\ \bigwedge_{p \in P(init)} \texttt{Param}_p(n', v_p)\end{array}\right) \implies \Psi_3^1 \wedge n = n' \qquad (\Gamma_4')$$

Recent weak aliveness.

$$\Phi_1 \implies \left(\begin{array}{l}(\exists m'\,.\,\Psi_1^1 \wedge \texttt{TimeRef}(m') \wedge \texttt{Preceq}(m', m)) \vee \\ (\exists m'\,.\,\Psi_1^2 \wedge \texttt{TimeRef}(m') \wedge \texttt{Preceq}(m', m))\end{array}\right) \qquad (\Gamma_5)$$

Table 2 Authentication formulas in Lowe's hierarchy

Weak aliveness in role

$$\Phi_1 \implies \Psi_1^1 \qquad (\Gamma_9)$$

Non-injective synchronization.

$$\Phi_3 \implies \exists \overline{m}\,.\,\Psi_3 \wedge \texttt{NodeOrder}(\overline{m}, n) \qquad (\Gamma_{11})$$

Injective synchronization.

$$\Gamma_{11} \quad \wedge \quad \Gamma_* \qquad (\Gamma_{12})$$

Table 3 Additional Cremers and Mauw authentication formulas

in the right role regardless of whether or not the peer correctly knows the initiator's identity. The result is weak aliveness in a role, Γ_9, obtained by omitting Ψ_1^2 from Γ_1. They also modify Γ_9 to obtain

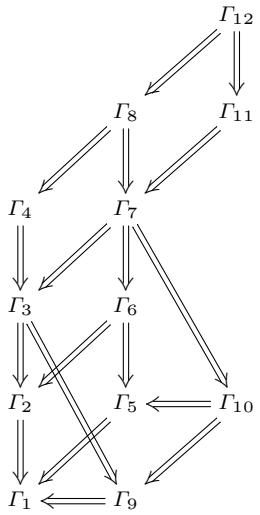
Γ_{10}, yielding recent weak aliveness in a role, by adding $\texttt{TimeRef}(m') \wedge \texttt{Preceq}(m', m)$ to the conclusion.

Cremers and Mauw also introduce the notion of *synchronization* which has both a non-injective and injective variety. Non-injective synchronization is to be similar to the Bellare-Rogaway notion of matching conversations [6] and Roscoe's intensional security [34]. It completely describes the intended protocol execution given completed run of the initiator. It requires that every transmission by the initiator precedes a matching reception by the peer and vice versa, where each corresponding pair of events agrees on the value of the message.

$\mathcal{GL}(\Pi)$ does not talk directly about this full message value. Instead, it talks about the parameters one-by-one. This limited expressiveness is actually a design target of our language: a security goal that depends too heavily on the exact structure of messages cannot be preserved under small syntactic changes. Since our language is designed for cross-protocol comparisons we have traded off expressibility for simultaneous applicability of goals to several protocols. In many cases, equalities between protocol messages can be enforced by stating that corresponding parameters of sender and receiver are equal. However, when they destructure the messages in different ways, this may not suffice. In a protocol such as PKINIT in which one party transmits a ciphertext it has encrypted, which another receives as an opaque blob, we cannot explicitly state that this component is unchanged.

Nonetheless, in many cases the specification of all the parameters will uniquely determine the message structure, and hence agreement on all parameters will entail agreement on the messages. In such cases we can formalize non-injective synchronization by modifying Γ_3 in the following way. We express the existence of each node we expect by the corresponding role position predicate. We express the desired ordering of these events using the $\texttt{Preceq}(\cdot,\cdot)$ and $\texttt{Coll}(\cdot,\cdot)$ predicates. This conjunction of predicates may be abbreviated by $\texttt{NodeOrder}(\overline{m}, n)$, where n is the only node variable in the hypothesis of Γ_3. We then conjoin this to the conclusion Ψ_3, existentially quantifying over all new node variables $\overline{m}$, resulting in non-injective synchronization, Γ_{11}. We do not add any more role parameter predicates or equalities because we assume the set V of parameters to agree on for Γ_3 is already the entire set of parameters $P(init)$ and is enough to enforce agreement on the value of each message of the protocol.

To express injective synchronization we simply conjoin the injective session property to non-injective synchronization resulting in Γ_{12}.



Γ_1 :	Weak Aliveness	Γ_7 :	Rec. non-inj. Agrmt.
Γ_2 :	Weak Agrmt.	Γ_8 :	Rec. inj. Agrmt.
Γ_3 :	Non-inj. Agreement	Γ_9 :	Weak Aliveness in Role
Γ_4 :	Inj. Agreement	Γ_{10} :	Rec. Aliveness in Role
Γ_5 :	Rec. Weak Aliveness	Γ_{11} :	Non-inj. Synch
Γ_6 :	Rec. Weak Agrmt.	Γ_{12} :	Inj. Synch

Fig. 4 Combined hierarchy

Ordering the goals by strength. Having expressed the various goals in these hierarchies in our logical goal language, we now demonstrate their relative strength by implication. As we saw above, many goals are obtained from others by a handful of reusable modifications. Most of these modifications involve either adding conjuncts to the conclusion of an implication or removing disjuncts. Such modifications strengthen the implication by strengthening its conclusion. The rest of this section discusses this set of reusable modifications to goals and demonstrates that the resulting goals are stronger. The results are summarized in Fig. 4 where the implications on opposite sides of each of the parallelograms hold for analogous reasons.

We first work our way up the diagram vertically. In order to see that Γ_2 is stronger than Γ_1 we first note that we have obtained Γ_2, in part, by adding conjuncts to the conclusion. If this were the only modification, it would immediately follow that Γ_2 was stronger than Γ_1. However, we have also seemingly strengthened the hypothesis by adding the conjunct $\texttt{Self}(n, i)$, thereby potentially weakening the implication. In fact, however, it does not actually weaken the goal. We know that $\Phi_1 \Leftrightarrow \Phi_1 \wedge \exists i.\ \texttt{Self}(n, i)$ because we have assumed that the self parameter is always well defined at the first (and hence also last) node of the roles. Any skeleton satisfying $\texttt{IDone}(n)$ has a well-defined value for the self parameter at that node, and hence also satisfies $\texttt{Self}(n, i)$. Because of this we could have expressed Γ_2 as the alternate form Γ_2' in Table 2, making it syntactically evident that it is stronger than Γ_1. Γ_6 is stronger than Γ_5 for the same reasons.

To see that Γ_3 is a stronger requirement than Γ_2 we argue similarly. The consequent is strengthened by removing one of the disjuncts and adding a conjunct to the remainder. Again, although it looks like a strengthening of the antecedent, the set V is chosen to ensure that for any $p \in V$, $\texttt{IDone}(n) \Rightarrow \exists v_p.\ \texttt{Param}_p(n, v_p)$. In order for non-injective agreement to be satisfiable, it may be necessary to replace $\texttt{RStart}(m)$ with a later node such as $\texttt{RDone}(m)$. The issue is that the responder may not have learned or committed to all the relevant values in V at its first node. Such a modification only serves to further strengthen the goal because later nodes always imply the existence of earlier nodes of the same strand. Similarly, we conclude that Γ_7 is stronger than Γ_6.

Injective agreement Γ_4 is clearly stronger than non-injective agreement Γ_3 since Γ_4 simply requires an additional goal to be met. For the same reason, we can conclude that $\Gamma_8 \Rightarrow \Gamma_7$ and that $\Gamma_{12} \Rightarrow \Gamma_{11}$.

Γ_9 strengthens Γ_1 since it results from omitting one of the disjuncts of Γ_1's conclusion. For the same reason $\Gamma_{10} \Rightarrow \Gamma_5$.

Γ_5 is obtained from Γ_1 by adding the requirement for recency. This is done by adding conjuncts to the conclusion, thereby strengthening the goal. The same rea-

soning justifies all the parallel implications that simply add recency to a goal.

To see that $\Gamma_3 \Rightarrow \Gamma_9$, we recognize that both goals require the peer to be in the expected role, but Γ_3 requires more agreement among the parameters. That is, Γ_3 can be obtained from Γ_9 by adding parameter predicates to both the hypothesis and the conclusion. As we argued above, the addition of these predicates to the hypothesis does not strictly strengthen the hypothesis. Thus the additional parameter predicates conjoined to the conclusion can only strengthen the goal. Γ_{10} similarly strengthens Γ_5.

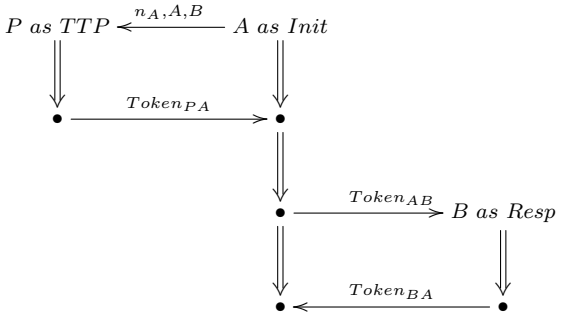
Finally, we argue that Γ_{11} is stronger than Γ_7. The conjunction $\texttt{NodeOrder}(\overline{m}, n)$ is typically stronger than the conjunction $\texttt{TimeRef}(m') \wedge \texttt{Preceq}(m', m)$, because $\texttt{TimeRef}(m')$ is usually one of the role position predicates included in $\texttt{NodeOrder}(\overline{m}, n)$, and $\texttt{Preceq}(m', m)$ is one of the required orderings. Thus $\texttt{NodeOrder}(\overline{m}, n)$ asserts the existence of more nodes and more orderings among them. This means that synchronization is a stronger requirement than recency. Thus we see that $\Gamma_{11} \Rightarrow \Gamma_7$, and similarly, $\Gamma_{12} \Rightarrow \Gamma_8$.

5 Protocols and Policies

One common source of resistance to the use of formal verification tools is their inability to capture the wide variety of realistic assumptions about the environment in which a protocol operates. When a potential attack is presented to a standards committee, it is common for the committee to argue that the attack violates some assumption guaranteed externally by some policy. In Section 5.1 we use an example protocol from an ISO standard [24] to show how we can use our goal language to capture not only properties that a protocol must enforce, but also limitations on the adversary that are enforceable through policy decisions. In Section 5.2 we discuss how a well-known flaw in the Transport Layer Security (TLS) protocol, discovered in 2009 [33], illuminates the tension that may arise when applications rely on a protocol to provide enough information to make local policy decisions. We show how our goal language might help structure standards bodies' understanding of the root cause of such a flaw and appropriate ways to mitigate it.

5.1 Policy-Based Constraints

The hierarchy is rather extensive, and yet it is still insufficient to express many goals or properties that arise for protocols developed as standards. In [3], Basin et al. use Scyther to analyze the ISO/IEC 9798 standard for



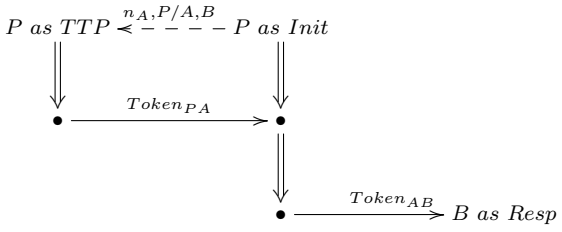
$Token_{PA}$=$\{|n_A, k_{AB}, B|\}_{k_{AP}}, \{|n_P, k_{AB}, A|\}_{k_{BP}}$
$Token_{AB}$=$\{|n_P, k_{AB}, A|\}_{k_{BP}}, \{|m_A, B|\}_{k_{AB}}$
$Token_{BA}$=$\{|n_B, A|\}_{k_{AB}}$

Fig. 5 ISO/IEC 9798-2, Mechanism 5

authentication protocols. They measure the protocols against goals in the above hierarchy and discover numerous cases in which the protocols do not achieve the goals. They then propose repairs to the protocols and demonstrate that the repaired protocols each achieve at least recent non-injective agreement. Interestingly, their alterations to the protocol messages were not sufficient to repair all the protocols. The repairs to two mechanisms from Part 2 of the standard [24] only achieves the goal under the extra external assumption that some policy is enforced. We dedicate the remainder of this section to demonstrating how we can apply our goal language to one of these protocols to express the adequacy of the protocol repairs under such policy-based constraints.

Figure 5 depicts the expected execution the protocol identified as Mechanism 5 in [24]. Two parties, A and B, want to authenticate each other using an online trusted third party (TTP) P. We assume that P shares symmetric keys k_{AP} and k_{BP} with A and B respectively. A, acting as initiator, begins by sending a nonce n_A together with B's identity to P. P, acting in the role of TTP, creates a fresh, symmetric session key k_{AB} to be used between A and B. The key is encrypted separately for A and B together with the random values n_A and n_P and the identities of the intended peers. A passes along P's encryption for B together with another encryption using the session key k_{AB} of a random value m_A and B's identity. The responder B completes the protocol by using k_{AB} to encrypt a random value n_B and A's identity.

The protocol as described does not achieve non-injective agreement because many of the encryptions have similar structure. This allows an adversary to combine messages from two sessions in which A and B are both acting as initiator to convince A that B is acting



$Token_{PA}$=$\{|n_A, k_{AB}, B|\}_{k_{AP}}, \{|n_P, k_{AB}, A|\}_{k_{BP}}$
$Token_{AB}$=$\{|n_P, k_{AB}, A|\}_{k_{BP}}, \{|m_A, B|\}_{k_{AB}}$
$Token_{BA}$=$\{|n_B, A|\}_{k_{AB}}$

Fig. 6 Attack on ISO/IEC 9798-2, Mechanism 5

as responder. Basin et al. suggest including unique tags in each of the encryptions so that such role confusions are no longer possible. Although this change eliminates the role-mixup attacks, the resulting protocol still does not achieve non-injective agreement (in fact, it does not even achieve weak aliveness).

Figure 6 displays a remaining attack on the protocol. In this instance P is executing both the initiator and the TTP roles simultaneously. As initiator, P sends the nonce and the two identities P and B. The adversary swaps P's identity with A's identity before delivering the message to P in the TTP role (represented with P/A over the dotted line). As TTP, P believes the request to be coming from A wanting to talk with B, so he prepares encryptions for both A and B using the long-term shared symmetric keys k_{AP} and k_{BP}. The adversary can redirect this message back to P in the initiator role. P is willing to accept this message because it believes it to be coming from A acting as TTP and it is encrypted with the key k_{AP} that he shares with A. When P acting as initiator sends the final message to B, B believes that A was active at some point which is not true. This violates weak aliveness of A from the responder's point of view.

The root cause of this problem is that the second message does not contain enough information about who the TTP thinks is playing which role, assuming that a principal can play both the TTP role and the initiator role. This suggests two natural ways to address the attack. The first is to alter the message structure to add more detailed information about which principals appear to be playing which roles. The second is to eliminate the ambiguity by forbidding any principals that acts as TTP to also act as either initiator or responder. While the first option may be desirable because it allows for more flexible deployment scenarios, it may be difficult for a standards committee to accept such a drastic change to the message structure when numerous implementations exist. Furthermore, the first option requires the protocol implementors to create and distribute a patch, while the second option empowers any enterprise that is an end user of the protocol to protect themselves with a policy change.

Our goal language makes it straightforward to evaluate whether a particular proposal to amend the protocol will achieve the desired authentication goal as we demonstrated for the case of PKINIT. But how can we use our techniques to evaluate whether a change in local policy will allow the protocol to achieve its goal? An analysis of the protocol will always tell us that it does not even achieve weak aliveness. We must capture what the protocol does achieve and ensure that it entails a goal that says either the protocol violates some external policy or else it achieves non-injective agreement.

The external policy can be viewed as an extra assumption we would like to make on our analysis. While goals typically represent assumptions as conjuncts in the antecedent, our language restricts us to using positive statements such as the fact that some event has occurred, or some value is freshly chosen. This policy is really a negative statement: A principal will not play both the TTP role and another role. We can express such requirements by negating the policy to make a positive statement and place that in the disjunctions of the conclusion. So if we let $\Phi'_3 \Rightarrow \Psi'_3$ represent the goal of non-injective agreement with the initiator from the responder's point of view, then we can use the following goal:

$$\Phi'_3 \quad \Longrightarrow \quad \Psi'_3 \vee \Psi_a \vee \Psi_b \qquad (\Gamma_{13})$$

in which we let

$$\Psi_a = \exists n_1, n_2, v\,.\ \mathtt{TTPStart}(n_1) \wedge \mathtt{Self}(n_1, v) \wedge \mathtt{InitStart}(n_2) \wedge \mathtt{Self}(n_2, v)$$
$$\Psi_b = \exists n_1, n_2, v\,.\ \mathtt{TTPStart}(n_1) \wedge \mathtt{Self}(n_1, v) \wedge \mathtt{RespStart}(n_2) \wedge \mathtt{Self}(n_2, v).$$

The disjunction $\Psi_a \vee \Psi_b$ captures a violation of the policy that restricts any principal that plays the TTP role from playing either the initiator role (Ψ_a) or the responder role (Ψ_b). Using CPSA we have verified that the protocol achieves goal Γ_{13} thereby demonstrating that such a policy, if properly enforced, can ensure the protocol achieves non-injective agreement.

Notice that Γ_{13} fits only tangentially into the hierarchy of Fig. 4. It is weaker than non-injective agreement because we achieved it by weakening the conclusion, adding extra disjuncts. However, it is not stronger than any of the other goals in the hierarchy. This example demonstrates the inadequacy of such hierarchies in addressing the real goals and constraints that arise

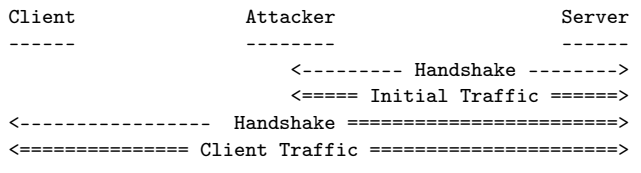


Fig. 7 TLS renegotiation attack

in the standardization of security protocols. We believe that combining our goal language with formal verification tools goes a long way to offering the flexibility and extensibility necessary to naturally express and verify goals and policy-based constraints that arise during the development and maintenance of protocol standards.

5.2 TLS Renegotiation

The policy in Section 5.1 is encoded in our security goal Γ_{13}. One aspect of that example that made it particularly clean is that the policy is a global one that can be enforced during the registration of participants. That is not always the case. Some policy decisions may dynamically rely on information provided by the protocol itself, for example when an application is relying on the underlying protocol for certain guarantees. When a surprising behavior—potentially a flaw—is discovered, we may need to change what information is available at the interface between the protocol and the application. The application can then use this information to correctly enforce a policy. We start with an example.

Transport Layer Security (TLS) [15] is a globally deployed protocol designed to add confidentiality, authentication and data integrity between communicating applications. It is secure, scalable, and robust enough to protect e-commerce transactions over HTTP. Despite its success, it has been forced to evolve over time, in part due to the discovery of various flaws in the design logic.

One such flaw, discovered in 2009 by Marsh Ray (see [33]), concerns renegotiating TLS parameters. It works on the boundary between the TLS layer and the application layer it supports. Fig. 7 is a high-level picture of the attack borrowed from [33]. The attacker first creates a *unilaterally* authenticated session with the server in the first handshake. Thus, the server authenticates itself to the attacker, but not vice versa. The attacker and server then exchange initial traffic protected by this TLS session. Later, a renegotiation occurs, possibly when the application at the server requires *mutual* authentication for some action. The attacker then allows the client to complete a handshake with the server, adding and removing TLS protections. The client's handshake occurs in the clear (depicted by <---> in Fig. 7), while the server's handshake is protected by the current TLS session. The attacker has no access to this newly negotiated session, but the server may retroactively attribute data sent in the previous session to the authenticated client. The server may then perform a sensitive action in response to a request sent by the attacker, but based on the credentials subsequently provided by the client.

In a typical illustration, the server is a pizza shop. The attacker orders a pizza for delivery to his address; then the client authenticates and orders a pizza for his own address. Because this is (now) a bilaterally authenticated connection, the server charges the orders made in this process to the client's account.

Which level is to blame for this attack?

- Does TLS fail to achieve a security goal that it should achieve?
- Or should the application take responsibility? It accepts some data out of a stream that is not bilaterally authenticated, and lumps it with the future data which will be bilaterally authenticated.
- Or is there shared responsibility? Perhaps TLS has the responsibility of providing clearer indications to the application when a change in the TLS properties takes place, and the application has the responsibility of heeding these indications.

TLS was subsequently updated with a renegotiation extension [33]. Renegotiation now creates a cryptographic binding between the new session and the existing session. If a server completes a mutually authenticated renegotiation with a client, then the earlier session was also negotiated with the same client. However, the authors of [33] also note:

> While this extension mitigates the man-in-the-middle attack described in the overview, it does not resolve all possible problems an application may face if it is unaware of renegotiation.

As Bhargavan et al. [7]'s recent attacks showed, the practically important issue was not in fact resolved by this. Perhaps future versions of TLS will simplify the situation by eliminating renegotiation.

However, the main point stands: for applications to use a security protocol such as TLS effectively, and take partial responsibility for achieving reasonable policies, some signals and commands must be shared between TLS and the application. Formal verification—coupled with our goal language—fits naturally here. With a little effort the goal language can be updated to address the multilayer nature of flaws such as this.

5.3 Enforcing Policies at Protocol Interfaces

The job of TLS, acting in either direction, is to take a stream of data from an application, delivering as much as possible of this stream to the peer application. When the sender is authenticated to the receiver, TLS guarantees that the portion delivered is an initial segment of what the authenticated sender transmitted. When the mode offers confidentiality, no other principal should learn about the content (beyond its length).

Naturally, these goals are subject to the usual assumptions, such as that the certificate authorities are trustworthy, that the private keys are uncompromised, and that randomness was freshly chosen.

When renegotiation occurs, this affects what the application should rely on. If a handshake authenticates a client identity C, then that guarantee should apply to the data starting when the cipher spec changes. It continues to apply until another cipher spec change or the end of the connection. Authentication guarantees for the "client traffic" should definitely not apply to the "initial traffic" of Fig. 7, which lies on the other side of a cipher spec change.

We may regard the TLS record layer as engaging in two types of events, namely the network-facing events and the application-facing events. Consider the outbound stream of data from the application to its peer. The network-facing events are transmission events for messages on the network to the peer. The application-facing events *accept* a stretch of data from the application, to be transmitted to the peer via the network. These stretches of data may be of various lengths, so that the record layer has to break them into many TLS records before network transmission.

For the inbound stream of data from the peer, the network-facing events are reception events for messages on the network from the peer. The application-facing events *deliver* a stretch of data to the application. As the record layer assembles these stretches of data from many TLS records on the network, they may be of various lengths.

Thus, in representing the TLS record layer, we use four kinds of node:

$\texttt{NetReceive}(n)$: n is a network-facing event receiving a record;

$\texttt{NetSend}(n)$: n is a network-facing event transmitting a record;

$\texttt{AppAccept}(n)$: n is an application-facing event accepting data from the application to go to the peer;

$\texttt{AppDeliver}(n)$: n is an application-facing event delivering data from the peer to the application.

Several parameter predicates may apply to these nodes. If $\texttt{NetReceive}(n)$ or $\texttt{NetSend}(n)$, then n certainly has a MAC key and an encryption key, as well as sequence number and payload data. We will not pause to formalize these, nor to define the roles that these events lie on. They are most easily formalized as protocol roles that interact with mutable state [21,31].

When $\texttt{AppAccept}(n)$ or $\texttt{AppDeliver}(n)$, then three parameter predicates that apply to n are:

$\texttt{AppData}(n, d)$: n is accepting or delivering the application data d;

$\texttt{AppPeer}(n, p)$: the authenticated peer at the time n occurs is p;

$\texttt{AppSelf}(n, p)$: the authenticated identity of the active principal at the time n occurs is p.

The predicates $\texttt{AppPeer}(n, p)$ and $\texttt{AppSelf}(n, p)$ may not hold of any value p if the endpoint is unauthenticated when n occurs.

Using these notions about the protocol behavior, the application can express its requirements about the security services that the protocol provides. We are interested in a situation where two segments of data are delivered to the pizza server, and it replies with a segment completing the transaction. We assume that the application defines a method to concatenate data segments d_1, d_2, which we write $d_1 \,\hat{}\, d_2$; and that it has a way to assert that data d expresses a well-formed transaction $\texttt{Transact}(d)$. Then the application is interested in the situation when:

$$\begin{aligned} &\texttt{AppAccept}(n_1) \wedge \texttt{AppAccept}(n_2) \\ \wedge\ &\texttt{AppDeliver}(n_3) \wedge \\ \bigwedge_{1 \le i \le 3}\ &\texttt{AppData}(n_i, d_i) \wedge \texttt{Transact}(d_1 \,\hat{}\, d_2 \,\hat{}\, d_3). \end{aligned}$$

In this situation, we certainly want it to be the case that there is a single client c which is the peer of all three events:

$$\begin{aligned} \exists c\,.\ &\texttt{AppPeer}(n_1, c) \wedge \texttt{AppPeer}(n_2, c) \\ &\wedge \texttt{AppPeer}(n_3, c). \end{aligned}$$

The authentication properties of the protocol itself may also imply further conclusions, such as c actually having transmitted $d_1 \,\hat{}\, d_2$.

Thus, the goal language we have provided can be enriched to express events at the interface between the protocol and its application. This notation—augmented with some application level vocabulary—allow the application author to express the meaningful conclusions that should hold whenever the application obtains its security services from the protocol.

6 Comparing Protocols by Hypotheses

Up to now, we have limited ourselves to considering only finite sets of goals G for the comparison of pro-

tocols. Using finite sets is helpful because $\mathsf{Ach}(\Pi) \cap G$ can be enumerated. The limitation of considering only finite sets of goals is that the distinguishing power of the induced partial order $\trianglelefteq^G$ is limited by the size of G (cf. Lemma 2). We may lack the imagination to ensure G includes goals that would help distinguish protocols in a useful way. A more flexible approach might be to fix a set of assumptions and try to compare protocols according to the strength of the conclusions they allow. For example, Π_2 should be considered stronger than Π_1 if Π_2 allows an initiator to conclude more information about their peer's session than Π_1 allows. This insight leads us to consider the following set of goals that all share a common hypothesis:

$$H(\Phi) = \{\Gamma \mid \mathsf{hyp}(\Gamma) = \Phi\}. \tag{9}$$

Thus, for $\Gamma, \Gamma' \in H(\Phi)$, $\Gamma \leq \Gamma'$ iff $\mathsf{conc}(\Gamma') \Rightarrow \mathsf{conc}(\Gamma)$. $H(\Phi)$ is typically not a finite set, so we can no longer calculate $\mathsf{Ach}(\Pi) \cap H(\Phi)$ by enumeration.

However, using our tool CPSA [32], we can find—for a large, natural class of assumptions Φ—a single strongest formula in $\mathsf{Ach}(\Pi) \cap H(\Phi)$. We will call this formula the *shape analysis sentence* $\mathsf{SAS}_\Phi(\Pi)$. It serves to summarize the whole set $\mathsf{Ach}(\Pi) \cap H(\Phi)$. The key idea here is *enrich-by-need protocol analysis*. As we will see, enrich-by-need analysis outputs such a maximally strong formula.

Thus the sets $H(\Phi)$ are a natural companion to enrich-by-need protocol analysis. This raises the question of whether the analysis methods used by other tools have similar connections to other classes of goals.

6.1 Enrich-by-Need Protocol Analysis

Our approach to protocol analysis is based on what we call the "point-of-view principle." Most of the security goals we care about in protocol design and analysis concern the point of view of a particular participant C. C *knows* that it has sent and received certain messages in a particular order. C may be willing to *assume* that certain keys are uncompromised, which for us means that they will be used only in accordance with the protocol in question. And C may also be willing to *assume* that certain randomly chosen values will not also be independently chosen by another participant, whether a regular (compliant) participant including C itself on another occasion, or an adversary.

These facts and assumptions may be formalized in a hypothesis. The hypothesis in Eqn. 3 is an example. It summarizes the situation in which a client has executed a full local run; it declares the variables C, S to stand for two of the parameters of the client run, and it adds the assumption that the signature key $\mathtt{sk}(S)$ of the peer is uncompromised.

The protocol analysis question is, given these facts and assumptions, what follows about what may happen on the network? The answers to this question are of two main kinds. Positive conclusions assert that some regular participant Q has taken protocol actions. These are authentication goals. They assert, subject to the assumptions, that C's message transmissions and receptions authenticate Q as having taken corresponding actions. Negative conclusions are generally non-disclosure assertions. They say that a value cannot be found available on the network in a particular form; often, that a key k cannot be observed unprotected by encryption on the network.

Skeletons and cohorts. The enrich-by-need process starts with a representation of the hypothesis. We will refer to these representations of behavior and assumptions as *skeletons* $\mathbb{A}$. A skeleton consists of some behavior of Π, i.e. some *regular strands* in the strand space terminology, together with the stated assumptions.

A *skeleton* $\mathbb{A}$ for Π is a structure that provides partial information about a set of Π-bundles. It consists of a finite set of regular strands of Π, or initial segments of them; some assumptions about which keys should be assumed uncompromised and which should be assumed to have been freshly chosen; and a partial ordering on the nodes of Π. A bundle $\mathcal{B}$ is an *instance* of a skeleton $\mathbb{A}$ if there is a structure-preserving map ("homomorphism") from the strands of $\mathbb{A}$ into the regular strands of $\mathcal{B}$ such that $\mathcal{B}$ satisfies the freshness and non-compromise assumptions of $\mathbb{A}$, and its arrows enrich the partial ordering of $\mathbb{A}$ (see [20] for copious detail).

CPSA [32] is a software tool that carries out protocol analysis in strand spaces. Given any skeleton $\mathbb{A}_0$ as a starting point, CPSA provides information about the set of all bundles $\mathcal{B}$ such that there is a homomorphism from $\mathbb{A}_0$ into $\mathcal{B}$. CPSA does this by computing a set of skeletons $\{\mathbb{B}_i\}_{i \in I}$ with two properties. First, each of the skeletons $\mathbb{B}_i$ is a *realized skeleton*. By this we mean that there is a bundle $\mathcal{B}_i$ such that $\mathbb{B}_i$ results from $\mathcal{B}_i$ when we "forget" the specific adversary strands in $\mathcal{B}_i$. Second, *every* bundle $\mathcal{B}$ containing an image of $\mathbb{A}_0$ also contains an image of one of the $\mathbb{B}_i$. We summarize these properties by saying that $\{\mathbb{B}_i\}_{i \in I}$ *characterizes* the skeletons compatible with the starting point $\mathbb{A}_0$.

We start the enrich-by-need analysis with a skeleton $\mathbb{A}_0$. At any point in the enrich-by-need process, we have a set $\mathcal{S}$ of skeletons to work with. Initially, $\mathcal{S} = \{\mathbb{A}_0\}$.

At each step, we select one of these skeletons $\mathbb{A} \in \mathcal{S}$, and ask if the behavior of the participants recorded in it is possible. When a participant receives a message, then the adversary should be able to generate that message,

using messages that have been sent earlier, without violating the assumptions. In this case, we regard that reception as "explained," since we know how the adversary can arrange to deliver the expected message. We say that that particular reception is *realized*. When every reception in a skeleton $\mathbb{A}$ is realized, we call $\mathbb{A}$ itself realized. It then represents—together with behavior that the adversary can supply—a possible complete execution, i.e. a bundle.

We collect the realized skeletons in a set $\mathcal{R}$.

If the skeleton $\mathbb{A} \in \mathcal{S}$ we select is not realized, then we use a small number of rules to generate an enrichment step. An enrichment step takes one unrealized reception and considers how to add some or all of the information that the adversary would need to generate its message. It returns a *cohort* of skeletons: This is a finite set $\{\mathbb{A}_1, \ldots, \mathbb{A}_i\}$ of skeletons forming a disjunctive representation of all the ways the regular participants can supply the necessary information to the adversary. We update $\mathcal{S}$ by removing $\mathbb{A}$ and adding the cohort members: $\mathcal{S}' = (\mathcal{S} \setminus \{\mathbb{A}\}) \cup \{\mathbb{A}_1, \ldots, \mathbb{A}_i\}$.

As a special case, a cohort may be the empty set, i.e. $i = 0$, and in this case $\mathbb{A}$ is discarded and nothing replaces it. This occurs when there are no possible behaviors of the regular participants that would explain the required reception. Then the skeleton $\mathbb{A}$ cannot contribute any executions (realized skeletons).

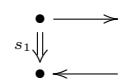
This process may not terminate, and in fact the underlying class of problems is undecidable [17]. However, when it does terminate, it yields a finite set $\mathcal{R}$ of realized skeletons with a crucial property: For a class of security goals, if they have no counterexample in the set $\mathcal{R}$, then the protocol really achieves that goal [22]. Moreover, we can inspect the members of $\mathcal{R}$ and determine whether any of them is a counterexample.

We call the members of $\mathcal{R}$ *shapes*, and they represent the *minimal, essentially different* executions consistent with the starting point.

Enrich-by-need analysis originates with Meadows's NPA [29]. Dawn Song's Athena [35] applied the idea to strand spaces [37]. Two systems in use currently that use the enrich-by-need idea in a form close to what we describe here are Scyther [14] and our CPSA [32]. See [20, 22] for a comprehensive discussion, and for more information about our terminology here.

Example: Initiator's authentication guarantee in PKINIT. Suppose that the client C has executed a strand of the client role in the fixed PKINIT_1, i.e., where we instantiate $F(C, n_2) = (C, n_2)$. Suppose also that we are willing to assume that the authentication server S has an uncompromised signature key $\text{sk}(S)$. We annotate this assumption as $\text{sk}(S) \in \mathsf{non}$, meaning that $\text{sk}(S)$ is non-compromised.

$\text{sk}(S) \in \mathsf{non}$ (10)

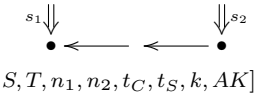


$s_1 \in \mathsf{Client}[C, S, T, n_1, n_2, t_C, t_S, k, AK]$

This is our starting point $\mathbb{A}_0$, depicted in (10), which corresponds to the information expressed in formula (3) from Section 2. C receives a message that contains the digital signature $[k, (C, n_2)]_{\text{sk}(S)}$, and we know that the adversary cannot produce this because $\text{sk}(S)$ is uncompromised. Thus, this second node of the local run is unrealized.

To explain this reception, we look at the protocol to see what ways a regular participant might create a message of the form $[k, (C, n_2)]_{\text{sk}(S)}$. In fact, there is only one. Namely, the second step of the KAS role does so. Knowing the KAS sends this signature means it will agree on the parameters used: S, k, C, n_2. However, we do not yet know anything about the other parameters used in S's strand. They could be different values $t'_C, T', n'_1, TGT', AK', t'_S$. Thus, we obtain a cohort containing a single skeleton $\mathbb{A}_1$, depicted in (11), that includes an additional KAS strand with the specified parameters.

$\text{sk}(S) \in \mathsf{non}$ (11)



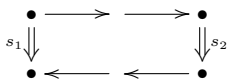
$s_1 \in \mathsf{Client}[C, S, T, n_1, n_2, t_C, t_S, k, AK]$
$s_2 \in \mathsf{KAS}[C, S, T', n'_1, n_2, t'_C, t'_S, k, AK']$

This skeleton is now already realized, because, with this weak assumption, the adversary may be able to use C's private decryption key to obtain k and modify the authenticator $\{\!|AK, n_1, t_S, T|\!\}_k$ as desired. The adversary might also be able to guess k, e.g. if S uses a badly skewed random number generator. Similarly, the components that are not cryptographically protected are under the power of the adversary.

We can we now re-start the analysis with two additional assumptions to eliminate these objections. First, we add $\text{sk}(C)$ to non. Second, we assume that S randomly generates k, and we write $k \in \mathsf{unique}$, meaning that k is chosen at a unique position. We are uninterested in the negligible probability of a collision between k and a value chosen by another principal, even one chosen by the adversary. This situation is depicted in (12).

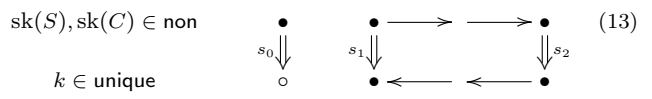
$\text{sk}(S), \text{sk}(C) \in \mathsf{non}$ (12)

$k \in \mathsf{unique}$



$s_1 \in \mathsf{Client}[C, S, T, n_1, n_2, t_C, t_S, k, AK]$
$s_2 \in \mathsf{KAS}[C, S, T', n'_1, n_2, t'_C, t'_S, k, AK']$

This skeleton is not realized, because with the assumption on $\mathrm{sk}(C)$, the adversary cannot create the signed unit $[t'_C, n_2]_{\mathrm{sk}(C)}$; it must come from a compliant principal. Examining the protocol, this can only be the first node of a client strand with matching parameters C, n_2, t'_C, i.e. a local run s_0:



(13)

$$s_0 \in \mathsf{Client}[C, S'', T'', n''_1, n_2, t'_C, t''_S, k'', AK'']$$
$$s_1 \in \mathsf{Client}[C, S, T, n_1, n_2, t_C, t_S, k, AK]$$
$$s_2 \in \mathsf{KAS}[C, S, T', n'_1, n_2, t'_C, t'_S, k, AK']$$

Curiously, two possibilities now remain. The strand s_0 could be identical with s_1, in which case the doubly-primed parameters equal the unprimed ones. Or alternatively, s_0 might be another client strand that has by chance selected the same n_2; we have not assumed $n_2 \in \mathsf{unique}$. In this case, the doubly-primed variables are not constrained. If we further add the assumption that $n_2 \in \mathsf{unique}$, then $s_1 = s_0$ follows.

In all of these cases, C and S do not have to agree on TGT, since this item is encrypted with a key shared between S and T, and C cannot decrypt it or check any properties about what he receives.

We have illustrated several points about enrich-by-need protocol analysis. Authentication follows by successive inferences about regular behavior, based on the message components the adversary cannot build. When two inferences are possible, the method branches, potentially resulting in several outputs. Various levels of authentication may be achieved; they depend on which parameters principals agree on, and which parameters may vary. Secrecy properties hold when no execution is compatible with the secret's disclosure.

More formally, there is a notion of homomorphism between skeletons [20]. Suppose that we start with the "point of view" expressed in the skeleton $\mathbb{A}_0$, and CPSA terminates, providing us with the shapes $\mathbb{C}_1, \ldots, \mathbb{C}_i$, which are the minimal, essentially different executions compatible with $\mathbb{A}_0$. Then for each $\mathbb{C}_j$, there is a homomorphism H_j from $\mathbb{A}_0$ to $\mathbb{C}_j$. Moreover, every homomorphism $K\colon \mathbb{A}_0 \to \mathbb{D}$ from $\mathbb{A}_0$ to a realized skeleton $\mathbb{D}$ agrees with at least one of the H_j. Specifically, we can regard K as the result of adding more information after one of the H_j. We mean that we can always find some $J\colon \mathbb{C}_j \to \mathbb{D}$ where K is the composition $K = J \circ H_j$.

6.2 Shape Analysis Formulas

Given a skeleton, we can summarize all of the information in it in the form of a conjunction of atomic formulas. We call this formula the *characteristic formula* for the skeleton, and write $\mathsf{cf}(\mathbb{A})$. Thus, a CPSA run with starting point $\mathbb{A}_0$ is essentially exploring the security consequences of $\mathsf{cf}(\mathbb{A}_0)$.

When CPSA reports that $\mathbb{A}_0$ leads to the shapes $\mathbb{C}_1, \ldots, \mathbb{C}_i$, it is telling us that any formula that is true in all of these skeletons, and is preserved by homomorphisms, is true in all realized skeletons $\mathbb{D}$ accessible from $\mathbb{A}_0$. The set of formulas preserved by homomorphism are called *positive existential*, and are those formulas built from atomic formulas, $\wedge, \vee$, and $\exists$. By contrast, formulas using negation $\neg\phi$, implication $\phi \implies \psi$, or universal quantification $\forall y\,.\,\phi$ are not always preserved by homomorphisms.

Thus, the disjunction of the characteristic formulas of the shapes $\mathbb{C}_1, \ldots, \mathbb{C}_i$ tell us just what security goals $\mathbb{A}_0$ leads to. However, we can be somewhat more precise. The skeleton $\mathbb{C}_j$ may have nodes that are not in the image of $\mathbb{A}_0$, and it may involve parameters that were not relevant in $\mathbb{A}_0$. Thus, $\mathbb{A}_0$ will not determine exactly which values these new items take in $\mathbb{C}_j$, e.g. which session key is chosen on some local run not present in $\mathbb{A}_0$. Thus, these new values should be existentially quantified. Effectively, these are all the variables that do not appear in $\mathsf{cf}(\mathbb{A}_0)$. Thus, for each $\mathbb{C}_j$, let $\overline{y}_j$ list all the variables in $\mathsf{cf}(\mathbb{C}_j)$ that are not in $\mathsf{cf}(\mathbb{A}_0)$. Let $\overline{x}$ list all the variables in $\mathsf{cf}(\mathbb{A}_0)$. Then this run of CPSA has validated the following formula that has the required form of a security goal (Definition 1):

$$\forall \overline{x}\,.\,(\mathsf{cf}(\mathbb{A}_0) \implies \bigvee_{1 \le j \le i} \exists \overline{y}_j\,.\,\mathsf{cf}(\mathbb{C}_j)) \tag{14}$$

The conclusion $\bigvee_{1 \le j \le i} \exists \overline{y}_j\,.\,\mathsf{cf}(\mathbb{C}_j)$ is the strongest formula that is true in all of the $\mathbb{C}_j$.

We call the formula (14) the *shape analysis sentence* for this run of CPSA. Since the empty disjunction is the constantly false sentence, in the special case where $i = 0$, (14) is $\forall \overline{x}\,.\,\mathsf{cf}(\mathbb{A}_0) \implies \mathsf{false}$. That is to say, it is $\forall \overline{x}\,.\,\neg\mathsf{cf}(\mathbb{A}_0)$.

We will frequently have need to reference particular shape analysis sentences and their various parts. We introduce here a notation for ease of presentation. Suppose the characteristic formula $\mathsf{cf}(\mathbb{A}_0)$ in protocol Π is represented by the conjunction of atomic formulas Φ, and each resulting shape is formalized by $\exists \overline{y}_j\,.\,\mathsf{cf}(\mathbb{C}_j)$. The shape analysis sentence (14) is called $\mathsf{SAS}_\Phi(\Pi)$. For the conclusion of $\mathsf{SAS}_\Phi(\Pi)$, i.e. the conclusion of the implication, we write $\mathsf{SASConc}_\Phi(\Pi)$.

6.3 Enrich-by-Need Compares Protocols by Hypotheses

Fortunately, the shape analysis sentences $\mathsf{SAS}_\Phi(\Pi)$ are always in $H(\Phi)$, and the induced partial order $\trianglelefteq^{H(\Phi)}$ has a nice property relative to $\mathsf{SAS}_\Phi(\Pi)$.

Theorem 2 *Π achieves Γ iff $\mathsf{SASConc}_{\mathsf{hyp}(\Gamma)}(\Pi) \Rightarrow \mathsf{conc}(\Gamma)$.*

Proof For the forward implication, we rely on the fact that $\mathsf{SASConc}_{\mathsf{hyp}(\Gamma)}(\Pi)$ is the strongest conclusion allowed from $\mathsf{hyp}(\Gamma)$ in Π. That is, whenever $\mathsf{hyp}(\Gamma) \Rightarrow \chi$ then $\mathsf{SASConc}_{\mathsf{hyp}(\Gamma)}(\Pi) \Rightarrow \chi$. So let $\chi = \mathsf{conc}(\Gamma)$.

For the reverse implication, the assumption means that $\Gamma \leq \mathsf{SAS}_{\mathsf{hyp}(\Gamma)}(\Pi)$ in the strength ordering. Also, Π achieves $\mathsf{SAS}_{\mathsf{hyp}(\Gamma)}(\Pi)$ by definition, so by Lemma 1, Π achieves Γ. □

This theorem says that we can completely characterize $\mathsf{Ach}(\Pi) \cap H(\Phi)$ because it has a single maximum element, namely $\mathsf{SAS}_\Phi(\Pi)$. Thus, we get the following corollary.

Corollary 1 *$\Pi_1 \trianglelefteq^{H(\Phi)} \Pi_2$ iff Π_2 satisfies $\mathsf{SAS}_\Phi(\Pi_1)$.*

This means we can use CPSA to compare protocols according to $\trianglelefteq^{H(\Phi)}$. We simply run the tool once to generate $\mathsf{SAS}_\Phi(\Pi_1)$ and then check whether Π_2 satisfies $\mathsf{SAS}_\Phi(\Pi_1)$ using any semi-decision procedure for finding counterexamples.

We can also define larger sets by considering a set P of hypotheses and taking the union $G = \bigcup_{\Phi \in P} H(\Phi)$. When P is finite we can run the tool once for each member of P. We thus verify that $\Pi_1 \trianglelefteq^G \Pi_2$, by checking that Π_2 satisfies $\mathsf{SAS}_\Phi(\Pi_1)$ for each $\Phi \in P$.

We can again use the example of the variants of PKINIT to demonstrate the induced ordering. We can let Γ be the desired security goal and run CPSA on both fixes PKINIT_1 and PKINIT_2 starting from inputs representing $\mathsf{hyp}(\Gamma)$. The result is that

$$\mathsf{SASConc}_{\mathsf{hyp}(\Gamma)}(\text{PKINIT}_1) = \mathsf{SASConc}_{\mathsf{hyp}(\Gamma)}(\text{PKINIT}_2).$$

Thus we find that

$$\text{PKINIT} \trianglelefteq^{H(\Phi)} \text{PKINIT}_1 \bowtie^{H(\Phi)} \text{PKINIT}_2.$$

This is the same ordering as for $\trianglelefteq^{\{\Gamma\}}$. In this case, increasing the size of the set of goals we consider does not help us further distinguish the protocols under consideration. It does, however, tell us that the two fixes are equally good for a larger class of goals. This assurance would be useful for standards committees. It would allow to them to know that both fixes will behave the same under adjustments to the strength of the conclusion of Γ.

6.4 Comparing Protocols by Conclusions

Using the set $H(\Phi)$ uses the idea that fixing a hypothesis, we can compare goals based on the strength of their conclusions. We can also reverse that idea by fixing a conclusion and comparing goals based on the strength of their hypotheses. The intuition is that for two protocols that both achieve a given conclusion, we might prefer the one that requires fewer assumptions. For example, in order for an initiator to authenticate a responder, one protocol may require that the private keys of *both* parties be kept secret, while another protocol may only require that the responder's private key be kept secret. This motivates the use of the following set of goals that share a common conclusion.

$$C(\chi) = \{\Gamma \mid \mathsf{conc}(\Gamma) = \chi\}$$

Thus, for $\Gamma, \Gamma' \in C(\chi)$, $\Gamma \leq \Gamma'$ if and only if $\mathsf{hyp}(\Gamma) \Rightarrow \mathsf{hyp}(\Gamma')$. In order to determine an ordering $\Pi_1 \trianglelefteq^{C(\chi)} \Pi_2$, we are again faced with the task of trying to compute $\mathsf{Ach}(\Pi_i) \cap C(\chi)$. Unfortunately, there is no simple analog to Theorem 2. $\mathsf{Ach}(\Pi) \cap C(\chi)$ may not have a single maximal element.

Discovering maximal elements of $\mathsf{Ach}(\Pi) \cap H(\Phi)$ relies on a sort of deductive inference in which conclusions are inferred from the hypothesis Φ. The enrich-by-need analysis method naturally supports this type of "forward" inference systematically determining the strongest conclusions allowed from the hypothesis.

Discovering maximal elements of $\mathsf{Ach}(\Pi) \cap C(\chi)$ is a type of *abductive reasoning* in which hypotheses are inferred from the conclusion χ. The enrich-by-need method is not as well suited to support this type of "backward" inference. A better method would be one that systematically determines weaker hypotheses that are still sufficient to imply χ. This would be a sort of "impoverish-as-possible" analysis method. We are unaware of any tools that are specifically designed to support this type of reasoning. Such a tool might enable an analog to Theorem 2 and thus provide a simple way to verify whether $\Pi_1 \trianglelefteq^{C(\chi)} \Pi_2$. This suggests a potential area for future research.

In the absence of an impoverish-as-possible analysis tool, we believe that $\trianglelefteq^{C(\chi)}$ is still a useful notion that can help clarify relationships among protocol variants. For one thing, if one finds a sufficiently strong assumption Φ to ensure χ, one can run CPSA repeatedly, omitting conjuncts of Φ until χ is no longer satisfied. Tool support apart, $\trianglelefteq^{C(\chi)}$ also serves to organize the concepts in a way that enables clearer discussions among standards committee members regarding relative trade-offs. Also, if we are able to use other means to come up with a goal $\Gamma \in C(\chi)$ that Π_1 achieves but

Π_2 does not, then we can use Theorem 1 to demonstrate $\Pi_1 \not\preceq^{C(\chi)} \Pi_2$.

7 Conclusion

In this paper we have presented a family of security metrics $\preceq^G$ for cryptographic protocols, parameterized by sets of goals G expressed in a logical language. The natural partial ordering on formulas induced by logical implication yields a partial order on protocols that reflects which attacks are possible for a Dolev-Yao style adversary to achieve. This ensures that protocol Π_1 is at least as strong as Π_2 exactly when Π_1 achieves any security goal achieved by Π_2. We showed how these security metrics can capture and expand well-studied security hierarchies from the literature [26, 14].

The language is designed to be tool-independent, so that any of the wide variety of protocol analysis tools available might be used to verify that a protocol meets a given goal. Although we rely on a well-understood semantics of our language in the strand space formalism, there should be no fundamental barriers to using other formalisms as a semantic base. We believe this is a step toward enabling a diverse set of analysis tools to "speak the same language," allowing analysts to use different tools based on their relative strengths. This vision is mostly limited by the types of properties we have chosen to consider and the adversary model we use to verify them. We currently only consider trace properties. Thus, our language is not well-suited for indistinguishability properties (useful for privacy goals, among others) or probabilistic properties relying on the more complex details of a computational adversary model. Expanding our ideas to accommodate such goals (and the tools that verify them) is an area of future work.

We also identified the strengths of a particular style of protocol analysis called enrich-by-need. This analysis style is embodied by such tools as our own CPSA as well as Scyther. With such tools we can identify the unique strongest goal achieved by a protocol Π in a particular class of goals denoted $H(\Phi)$ that all share a common hypothesis Φ. We suspect other styles of protocol analysis might be similarly characterized. An interesting area for future investigation would be to develop an "impoverish-as-possible" style of analysis which would identify maximal elements achieved by Π in the set $C(\chi)$ of goals that share a conclusion χ.

We believe this logical formulation of security goals supported by formal verification can help the process of standardizing cryptographic protocols. Standards bodies could adopt the practice of requiring submissions to include explicit claims of goals they achieve—written in a formal goal language such as ours—together with evidence that those goals have been formally verified with an analysis tool. Our motivating example of PKINIT suggests that the process of explicitly including formal security claims (even without evidence) would go a long way toward improving the quality of published standards as suggested by Basin et al. [5]. It would force the standards bodies to specifically delineate what properties a protocol is designed to achieve, for instance formulas (4) and (7) in the case of PKINIT. The result would be standards that contain concrete artifacts that could be independently verified by others, thereby increasing the public's confidence in the claims.

Our logical goal language could be useful even in the absence of a formal process for its use. The precision of the language makes it easier to understand the security consequences of subtle changes to a protocol's design. This can help guide committee members by providing explicit statements and evidence for the positive and negative consequences of design changes, thereby making it easier to make a case for or against a particular design choice. We expect that, used in this way, our goal language could make the standardization process more transparent, with results that are more robust to future attacks.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages (POPL '01)*, pages 104–115, January 2001.
2. Omar Almousa, Sebastian A. Mödersheim, Paolo Modesti, and Luca Viganò. Typing and compositionality for security protocols: A generalization to the geometric fragment. In *ESORICS*, LNCS. Springer, September 2015.
3. David A. Basin, Cas Cremers, and Simon Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 21(6):817–846, 2013.
4. David A. Basin and Cas J. F. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *ESORICS*, pages 340–356, 2010.
5. David A. Basin, Cas J. F. Cremers, Kunihiko Miyazaki, Sasa Radomirovic, and Dai Watanabe. Improving the security of cryptographic protocol standards. *IEEE Security & Privacy*, 13(3):24–31, 2015.
6. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Crypto*, pages 232–249, 1993.
7. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *IEEE Symposium on Security and Privacy*, 2014.
8. Bruno Blanchet. An efficient protocol verifier based on Prolog rules. In *14th Computer Security Foundations Workshop*, pages 82–96. IEEE CS Press, June 2001.
9. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Eurocrypt*, LNCS, pages 453–474. Springer, 2001.

10. Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *Eurocrypt*, LNCS, pages 337–351. Springer Verlag, 2002.
11. Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key Kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.
12. The MITRE Corporation. The common vulnerabilities and exposures (CVE) initiative. `http://cve.mitre.org`.
13. The MITRE Corporation. The common weakness enumeration (CWE). `http://cwe.mitre.org`.
14. Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Springer, 2012.
15. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.
16. Daniel J. Dougherty and Joshua D. Guttman. Decidability for lightweight Diffie-Hellman protocols. In *IEEE Symposium on Computer Security Foundations*, 2014.
17. Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004. Initial version appeared in *Workshop on Formal Methods and Security Protocols*, 1999.
18. Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. *Foundations of Security Analysis and Design V*, pages 1–50, 2009.
19. International Organization for Standardization. ISO/IEC 29128: Information technology—security techniques—verification of cryptographic protocols, 2011.
20. Joshua D. Guttman. Shapes: Surveying crypto protocol runs. In Veronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, Cryptology and Information Security Series. IOS Press, 2011.
21. Joshua D. Guttman. State and progress in strand spaces: Proving fair exchange. *Journal of Automated Reasoning*, 48(2):159–195, 2012.
22. Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.
23. Joshua D Guttman, Moses D Liskov, and Paul D Rowe. Security goals and evolving standards. In *Security Standardisation Research*, pages 93–110. Springer, 2014.
24. ISO/IEC IS 9798-2, "Entity authentication mechanisms – Part 2: Entity authentication using symmetric encipherment algorithms", 1993.
25. Changwei Liu, Anoop Singhal, and Duminda Wijesekera. A model towards using evidence from security events for network attack analysis. In *WOSIS 2014 - Proceedings of the 11th International Workshop on Security in Information Systems, Lisbon, Portugal, 27 April, 2014*, pages 83–95, 2014.
26. Gavin Lowe. A hierarchy of authentication specification. In *CSFW*, pages 31–44, 1997.
27. R. D. Luce and P. Suppes. Measurement, theory of. *Encyclopedia Britannica*, 15th Edition(11):739–745, 1974.
28. Robert A. Martin. Making security measurable and manageable. In *MILCOM 2008*, November 2008.
29. C. Meadows. The NRL protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.
30. C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806.
31. John D. Ramsdell, Daniel J. Dougherty, Joshua D. Guttman, and Paul D. Rowe. A hybrid analysis for security protocols with state. In *Integrated Formal Methods*, pages 272–287, 2014.
32. John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. `http://hackage.haskell.org/package/cpsa`.
33. E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010.
34. A. W. Roscoe. Intensional specifications of security protocols. In *IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.
35. Dawn Xiaodong Song. Athena: A new efficient automated checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE CS Press, June 1999.
36. Kun Sun, Sushil Jajodia, Jason Li, Yi Cheng, Wei Tang, and Anoop Singhal. Automatic security analysis using security metrics. In *MILCOM 2011*, November 2011.
37. F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
38. Thomas Y. C. Woo and Simon S. Lam. Verifying authentication protocols: Methodology and example. In *Proc. Int. Conference on Network Protocols*, October 1993.
39. L. Zhu and B. Tung. Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). RFC 4556 (Proposed Standard), June 2006. Updated by RFC 6112.