Security Goals and Evolving Standards

Joshua D. Guttman, Moses D. Liskov, and Paul D. Rowe
{guttman,mliskov,prowe}@mitre.org

The MITRE Corporation

Abstract. With security standards, as with software, we cannot expect to eliminate all security flaws prior to publication. Protocol standards are often updated because flaws are discovered after deployment. The constraints of the deployments, and variety of independent stakeholders, mean that different ways to mitigate a flaw may be proposed and debated.

In this paper, we propose a criterion for one mitigation to be at least as good as another from the point of view of security. This criterion is supported by rigorous protocol analysis tools. We also show that the same idea is applicable even when some approaches to mitigating the flaw require cooperation between the protocol and its application-level caller.

1 Introduction

Security standards, which often contain errors, evolve over time as people correct them. Often, their flaws may be discovered after considerable product deployment, which places great pressure on the choice of mitigation. The constraints of the operational deployments and the need to satisfy the various stakeholders involved is crucial, but so is a precise understanding of the attack and what it enables.

How are we to choose among various proposed alternatives for mitigating a flaw? A security flaw is a failure of the protocol to meet a goal—possibly 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, any two alternatives both of which eliminate a particular insecure scenario may not have equivalent security implications.

In this paper, we describe a formal language for expressing protocol security goals that supports "enrich-by-need" analysis tools such as the Cryptographic Protocol Shapes Analyzer (CPSA) [18]. We will use it to guide our description, even though there are other tools that are using other versions of enrich-by-need [5, 13].

Each "enrich-by-need" analysis process starts from a scenario containing some protocol behavior, 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 results indicate what authentication guarantees the protocol ensures.

For each run of the analyzer, there is a formula expressing the security goal that the analysis justifies. These security goals are independent of the particular protocol variant described.

Our contributions. We make two main contributions in this paper. First, we show how to compare security consequences by analyzing alternatives in a formal framework. The protocol analyses determine a partial order on protocols. When one protocol Π_1 is above another protocol Π_2 in this order, that means that Π_1 achieves at least as much security as Π_2 with regard to the starting scenario of the analysis.

Second, we show that our techniques are flexible enough to accommodate a variety of viewpoints on the protocol and its goals during the remediation discussion. Whether an attack represents a flaw in the protocol, or a flaw in the interface between protocol and application, or perhaps some mixture of the two, is often debatable. Our techniques may apply to a model of the protocol at any level of detail, and we describe how models that include interfaces relate to models that do not. A model of this sort may help to clarify what goals are important for a protocol to offer to the applications that use it.

Structure of this paper. In Section 2, we consider 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 describe how to evaluate them with CPSA and the security goal language it suggests, giving a clear result: From the security point of view, the choices are equally good. In Section 3, we explain the core ideas of CPSA, which motivates our choice of protocol goal languages in Section 4.

In Section 5, we turn to cross level issues, illustrated with the TLS vulnerabilities arising from renegotiation. One natural diagnosis of this flaw is that the higher-level application is not aware enough of how the API it uses engages in the TLS protocol. In Section 6, we show that our techniques support discussion at this level also. Higher-level goals can thus be described purely in terms of observable application behavior.

2 Example: Kerberos PKINIT

PKINIT [22] is an extension to Kerberos [17] 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 ease the management burden of establishing shared secrets (specifically passwords) and maintaining them, which the standard Kerberos exchange requires.

Cervesato et al. found a flaw in PKINIT version 25 [4], which was already widely deployed. The flaw was eventually fixed in version 27. Figure 1 shows

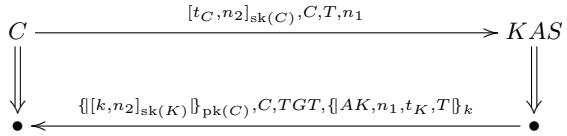


Fig. 1. PKINIT version 25, where $TGT = \{\!|AK, C, t_K|\!\}_{k_T}$

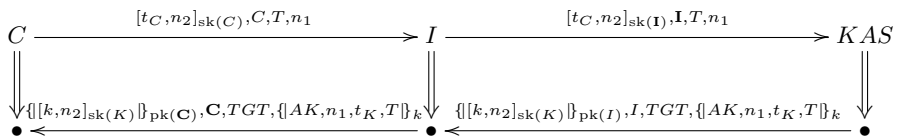


Fig. 2. Attack on flawed PKINIT, where $TGT = \{\!|AK, I, t_K|\!\}_{k_T}$

the expected message flow between the client and the KAS in v. 25. The client provides the 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 replies by creating a fresh session key k, signing it 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_K 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 K and T. It contains AK, the client's identity C and the expiration time t_K of the ticket.

In Cervesato et al.'s attack [4] (Fig. 2), an adversary I has obtained a private key to talk with the KAS. 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 KAS 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, because C believes the only other entity with knowledge of AK is T.

The attack arises from a lack of cryptographic binding between the session key k, and the client's identity C [4]. When C completes the two-message exchange, although she knows the KAS must have recently produced the keying material (due to the binding between k and n_2), it would be incorrect to conclude that the KAS intended the key to be used by C. Identifying this as the root cause of the attack suggests a natural fix, namely including the client's

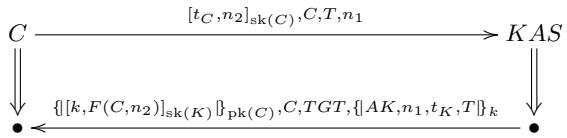


Fig. 3. Generic fix for PKINIT

identity C in the signed portion of the second message. Indeed this is the first suggestion in [4].

The authors of the PKINIT standard offered a different suggestion. For reasons of operational feasibility 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.

Cervesato et al., working with a manual proof method, opted to verify a generic scheme for mitigating the attack, 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 is prevented if n_2 is replaced with any expression $F(C, n_2)$ that is injective on those values (i.e. $F(C, n_2) = F(C', n_2')$ implies $C = C'$ and $n_2 = n_2'$).

We obtain the first proposal by instantiating F as the identity: $F(C, n_2) = (C, n_2)$. The second proposal results by instantiating F as the MAC of the client's request: $F(C, n_2) = H_k([t_C, n_2]_{\mathrm{sk}(C)}, C, T, n_1)$. Since the MAC provides second preimage resistance, the injectivity requirement holds with overwhelming probability (Fig. 3).

The PKINIT parable illustrates recurring themes in protocol standard development and maintenance. Frequently, attacks show us that we care about previously unstated, unrecognized security goals. PKINIT does achieve some level of authentication, but it fails a more stringent type of authentication. In Lowe's terms [12], PKINIT achieves recent aliveness for both the client and the KAS because each party signs time-dependent data. However, PKINIT fails weak agreement which requires each side to know the other party was engaged in the protocol *with them*. When we see the attack, it forces us to identify explicitly 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 the KAS 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 [4] is to state carefully what security goal the repair provides.

This goal can be achieved by different mitigations. Issues of efficiency, ease of deployment, or robustness to future protocol modifications may influence various stakeholders to prefer different mitigations. In our PKINIT example, 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 4) 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.

3 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 P. P *knows* that it has sent and received certain messages in a particular order. P 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 P 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 P itself on another occasion, or an adversary.

The protocol analysis question is, given these facts and assumptions, what follows about what may happen on the network? These conclusions are of two main kinds. Positive conclusions assert that some regular participant Q has taken protocol actions. These are authentication goals. They say that P's message transmissions and receptions authenticate Q as having taken some corresponding actions, subject to the assumptions. Negative conclusions are generally nondisclosure 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* $\mathbb{A}$. The skeleton $\mathbb{A}_0$ we start from includes some behavior of P, together with the stipulated assumptions. 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. 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, meaning a finite set $\{\mathbb{A}_1, \ldots, \mathbb{A}_i\}$ of skeletons which together supply this information to the adversary in all of the ways that the regular participants could supply it. We update $\mathcal{S}$ by removing $\mathbb{A}$ and adding the cohort members: $\mathcal{S}' = (\mathcal{S} \setminus \{\mathbb{A}\}) \cup \{\mathbb{A}_1, \ldots, \mathbb{A}_i\}$.

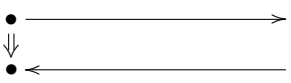
As a special case, a cohort may be the empty set, i.e. $i = 0$, and in this case $\mathbb{A}$ is discarded and nothing replaces it. This occurs when there are no possible behaviors of the regular participants that would explain the required reception. Then the skeleton $\mathbb{A}$ cannot contribute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

This process may not terminate, and in fact the underlying class of problems is undecidable [9]. 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 [11]. 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 protocol analysis originates with Meadows's NPA [13]. Dawn Song's Athena [20] applied the idea to strand spaces [21]. Two systems in use currently that use the enrich-by-need idea in a form close to what we describe here are Scyther [5] and CPSA [18]. See [10, 11] for a comprehensive discussion, and for more information about our terminology here.

In particular, we will use the term *regular strand* to mean a local run of a particular principal in a single compliant local session of a protocol. A regular strand (or often, we will just say strand) contains a sequence of transmission and reception actions. We will refer to any one of these actions as a *node*.

Example 1: Initiator's Authentication Guarantee in PKINIT. Suppose that the client C has executed a strand of the client role in the fixed PKINIT, where for now we will instantiate $F(C, n_2) = (C, n_2)$. Suppose also that we are willing to assume that the authentication server K has an uncompromised signature key $\mathrm{sk}(K)$. We annotate this assumption as $\mathrm{sk}(K) \in \mathsf{non}$, meaning that $\mathrm{sk}(K)$ is non-compromised.



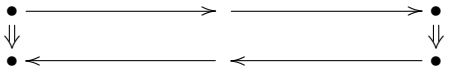
$$\mathsf{Client}[\mathsf{C}, \mathsf{K}, \mathsf{T}, \mathsf{n}_1, \mathsf{n}_2, \mathsf{t}_\mathsf{C}, \mathsf{t}_\mathsf{K}, \mathsf{k}, \mathsf{AK}] \quad \mathrm{sk}(K) \in \mathsf{non} \tag{1}$$

This is our starting point $\mathbb{A}_0$. C receives a message that contains the digital signature $[k, (C, n_2)]_{\mathrm{sk}(K)}$, and we know that the adversary cannot produce this

because $\mathrm{sk}(K)$ 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 $[k, (C, n_2)]_{\mathrm{sk}(K)}$. 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: K, k, C, n_2. However, we do not yet know anything about the other parameters used in K's strand. They could be different values $t'_C, T', n'_1, TGT', AK', t'_K$. Thus, we obtain a cohort containing a single skeleton $\mathbb{A}_1$ that includes an additional KAS strand with the specified parameters.

$$\mathrm{sk}(K) \in \mathsf{non} \tag{2}$$

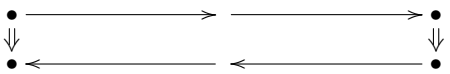


$\mathsf{Client}[C, K, T, n_1, n_2, t_C, t_K, k, AK]$ $\quad$ $\mathsf{KAS}[C, K, T', n'_1, n_2, t'_C, t'_K, k, AK']$

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 $\{|AK, n_1, t_K, T|\}_k$ as desired. The adversary might also be able to guess k, e.g. if K 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 $\mathrm{sk}(C)$ to non. Second, we assume that K 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.

$$\mathrm{sk}(K), \mathrm{sk}(C) \in \mathsf{non} \qquad\qquad k \in \mathsf{unique} \tag{3}$$



$\mathsf{Client}[C, K, T, n_1, n_2, t_C, t_K, k, AK]$ $\quad$ $\mathsf{KAS}[C, K, T', n'_1, n_2, t'_C, t'_K, k, AK']$

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 a client strand with matching parameters C, n_2, t'_C, i.e. a local run $\mathsf{Client}[C, K'', T'', n''_1, n_2, t'_C, t''_K, k'', AK'']$. Curiously, there are two possibilities now. This strand could be identical with the one already in the diagram, in which case the doubly-primed parameters are identical with the unprimed ones. Or alternatively, it might be another client strand that has also by chance selected the same n_2, since 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 the second client strand must coincide with the first.

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

This summary of enrich-by-need protocol analysis illustrates several important points. Authentication properties are built up by successive inferences of regular behavior, driven by some message component the adversary cannot build. When two inferences are possible the method branches, potentially resulting in a set of outputs. Various levels of authentication may be achieved according to which parameters principals agree on, and which parameters may vary. Secrecy properties are met when we can infer that no execution is compatible with the disclosure of the secret.

More formally, there is a notion of homomorphism between skeletons [10]. Given a starting point $\mathbb{A}_0$, with shapes $\mathbb{C}_1, \ldots, \mathbb{C}_i$, 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 $J \colon \mathbb{C}_j \to \mathbb{D}$ such that K is the composition $K = J \circ H_j$.

4 A Language of Protocol Goals

Shape analysis formulas. The pattern of enrich-by-need protocol analysis suggests how to express the security properties of protocols. These security properties are essentially implications, that say that if the circumstances described in the starting point $\mathbb{A}_0$ hold, then some further information must hold. 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 formula:

$$\forall \overline{x} \,.\, (\mathsf{cf}(\mathbb{A}_0) \implies \bigvee_{1 \le j \le i} \exists \overline{y_j} \,.\, \mathsf{cf}(\mathbb{C}_j)) \tag{4}$$

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 (4) the *shape analysis formula* for this run of CPSA. In the special case where $i = 0$, so that the conclusion of the implication is the empty disjunction, (4) is $\forall \overline{x} \,.\, \mathsf{cf}(\mathbb{A}_0) \implies \mathsf{false}$, or equivalently $\forall \overline{x} \,.\, \neg \mathsf{cf}(\mathbb{A}_0)$, since the empty disjunction is the constantly false formula.

The goal language $\mathcal{GL}(\Pi)$. So far, we have discussed characteristic formulas without concern for the vocabulary we use to build them. We choose a vocabulary that is motivated by the kinds of analysis CPSA does. In particular, it is adapted to expressing which instances of roles have occurred, and how far each has progressed. It also allows us to say what value each parameter takes; we have already seen that a prime category of flaw occurs when local runs that should agree on a parameter do not. The language also expresses the orderings among events, and assumptions on uncompromised keys and fresh values.

However, it is also designed to have the minimum possible expressiveness. 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. As a consequence, its formulas are preserved under a class of "security preserving" transformations between protocols [11]. Also, for interesting restricted classes of protocols, the set of security goals they achieve is decidable [8]. These properties require careful control over expressiveness.

In this language, we may summarize Eqn. (1) by the formula:

$$\mathtt{ClientDone}(n) \wedge \mathtt{Peer}(n, K) \wedge \mathtt{Non}(\mathtt{sk}(K)) \tag{5}$$

This asserts of a node n that it completes a client run, i.e. it is the second event on that local run. It also asserts that the peer parameter of this node n is a name K such that the signature key of K is non-compromised. The letters n, K here are free variables, and this formula is satisfied under an assignment of values to the free variables if those values have the properties we mentioned. A precise semantics is given in [11].

Observe that we don't have to say $\mathtt{ClientStart}(m)$, referring to the first node of the run. The presence of a second node ensures that the previous step occurred, and we don't need to say anything in particular about it.

Turning to Eqn. (2), the conjuncts of Eqn. (5) still hold. There is however also another strand, which is a complete KAS run. We also know that several of its parameters agree with those of C:

$$\begin{aligned} (5) \wedge \mathtt{Self}(n, c) \wedge\ & \mathtt{AuthNonce}(n, n_2) \wedge \\ \mathtt{KASDone}(m) \wedge \mathtt{Self}(m, K) \wedge\ & \mathtt{AuthNonce}(m, n_2) \wedge \\ \mathtt{Peer}(m, c) \wedge\ & \mathtt{Preceq}(m, n) \end{aligned} \tag{6}$$

The shape analysis formula that results has a single disjunct in the conclusion:

$$\forall n, K\,.\,(5) \Longrightarrow \exists m, c, n_2\,.\,(6) \tag{7}$$

More generally, suppose that we are given a protocol Π. It has a number of roles, and each of its roles has a number of nodes. For each of these nodes, the goal language $\mathcal{GL}(\Pi)$ has a *role position predicate*. The two predicates $\texttt{ClientDone}(n)$ and $\texttt{KASDone}(m)$ used above are examples. Each one 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 one 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. $\texttt{Peer}(m, c)$ asserts that m appears to be partnered with 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. The predicate names may be chosen as convenient. For instance, we may choose to use the same predicates for two different protocols, using this to emphasize structural similarities between them.

All protocols also have some shared common vocabulary (summarized in Table 1) that helps to express the structural properties of bundles. $\texttt{Preceq}(m, n)$ asserts that one node occurs before another; $\texttt{Coll}(m, n)$ says that they lie on the same strand. $\texttt{Non}(v)$ and $\texttt{Unq}(v)$ express non-compromise and freshness (unique origination). $\texttt{pk}(a)$ and $\texttt{sk}(a)$ relate a principal to its keys, $\texttt{ltk}(a, b)$ represents the long-term key of two principals, and $\texttt{inv}(k)$ is the inverse of a key.

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 the languages $\mathcal{GL}(\Pi)$

Using shape analysis formulas to evaluate alternatives. We now return to the intuitive notions of "good enough" and "as good as" and describe how these notions can be rigorously reflected through a combined understanding of shape analysis formulas and goal languages.

The notion of "good enough", naturally, is to be defined relative to a goal or set of goals. A protocol Π is good enough if all the required goals are satisfied in all executions. Each goal is of the form

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

where Φ and Ψ are conjunctions of atomic formulas of the goal language $\mathcal{GL}(\Pi)$. We can evaluate whether Π achieves the goal by running CPSA starting from a suitable skeleton, the "characteristic skeleton" of Φ. If some Ψ_j is satisfied in each of the resulting shapes, the goal is achieved.

"At least as good as." We relativize our definition of one protocol being at least as good as another to a particular hypothesis Φ. This hypothesis should be a formula of both $\mathcal{GL}(\Pi_1)$ and $\mathcal{GL}(\Pi_2)$. In that case, Π_2 is *at least as good as* Π_1 *relative to* the hypothesis Φ if, for every goal of the form $\forall \bar{x} \,.\, \Phi \Longrightarrow \bigvee \Psi_j$ with this hypothesis Φ, if Π_1 achieves this goal, then so does Π_2.

We can write $\Pi_1 \preceq_\Phi \Pi_2$ to mean that Π_2 is at least as good as Π_1 relative to Φ.

Two protocols are *equally good relative to* Φ if each is at least as good as the other relative to it.

We can establish $\Pi_1 \preceq_\Phi \Pi_2$ from shape analysis formulas. Φ determines a skeleton $\mathbb{A}_0$ in protocol Π_1 and a skeleton $\mathbb{B}_0$ in protocol Π_2. Suppose the set of shapes for $\mathbb{A}_0$ are $\mathbb{C}_1, \ldots, \mathbb{C}_n$, and the set of shapes for $\mathbb{B}_0$ are $\mathbb{D}_1, \ldots, \mathbb{D}_m$. If the disjunction of the characteristic formulas of the $\mathbb{D}_i$ entails the disjunction of the characteristic formulas of the $\mathbb{C}_j$, then $\Pi_1 \preceq_\Phi \Pi_2$:

$$\bigvee_{i \leq m} \exists \overline{y_i} \,.\, \mathsf{cf}(\mathbb{D}_i) \implies \bigvee_{j \leq n} \exists \overline{z_j} \,.\, \mathsf{cf}(\mathbb{C}_j). \tag{9}$$

If Π_1 achieves a goal $\forall \bar{x} \,.\, \Phi \Rightarrow \Psi$, then $\bigvee_{j \leq n} \exists \overline{z_j} \,.\, \mathsf{cf}(\mathbb{C}_j) \implies \Psi$. Hence, Π_2 achieves the goal also, because in protocol Π_2,

$$\begin{aligned} \forall \bar{x} \,.\, \Phi \Rightarrow & \bigvee_{1 \leq i \leq n} \exists \overline{y_i} \,.\, \mathsf{cf}(\mathbb{D}_i) \\ \Rightarrow & \bigvee_{1 \leq j \leq m} \exists \overline{z_j} \,.\, \mathsf{cf}(\mathbb{C}_j) \;\Rightarrow\; \Psi \end{aligned}$$

5 Example: TLS Renegotiation

Our method gives a good criterion for deciding that two protocols are equally good—or one is at least as good as the other—relative to a hypothesis Φ. But as described, this method applies if the fix consists of internal protocol modifications only. In the remainder of this paper, we would like to consider the possibility that the fix involves both modifying the protocol and also the information that it passes up to the application on behalf of which it is acting. Thus, part of the resolution is for the application to use this additional information correctly, so as to achieve its security goals. The underlying protocol is obligated to provide it with accurate information, and signaling when relevant events occur. We start with an example.

Transport Layer Security (TLS) [7] is a globally deployed protocol designed to add confidentiality, authentication and data integrity between two communicating applications. It is secure, scalable, and robust enough to protect e-commerce

transactions performed over HTTP. Despite the success of TLS 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, concerns renegotiating TLS parameters. It works on the boundary between the TLS layer and the application layer it supports. [19] contains a good description of the flaw; we give a brief summary.

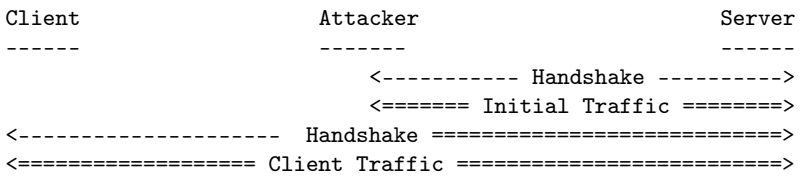


Fig. 4. TLS renegotiation attack

Fig. 4 (borrowed from [19]) is a high-level picture of the attack. 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. 4), 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. 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 should provide clearer indications to the application when a change in the TLS properties takes place, and then the application should heed these indications.

In fact TLS was updated with a renegotiation extension [19]. TLS renegotiation now cryptographically binds the new session to the existing session. If a server completes a mutually authenticated renegotiation with a client, then the current session was also negotiated with the same client. However, the authors of [19] 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. [2]'s recent attacks showed, the practically important issue was not in fact resolved by this.

However, for applications to take partial responsibility, some signals and commands must be shared between TLS and the application. Enrich-by-need protocol analysis—coupled with our goal language—fits in naturally here. With a little effort the goal language can be updated to address the multilayer nature of flaws such as this.

6 Goals for Protocol Interfaces

We now describe how to express protocol goals to make cross-level choices explicit.

The job of TLS, acting in either direction, is to take a stream of data from a transmitting application, and to deliver as much as possible of this stream to the receiving 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 (as opposed to the 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 a renegotiation occurs, this affects what the application should rely on. If a handshake authenticates a client identity C, then the authentication guarantee should apply to the data starting when the cipher spec changes. We will call the period starting from a cipher spec change, and lasting until the next one (if any) an *epoch*, and part of the work of a handshake is to agree on an epoch ID between the endpoints. Thus, any guarantee should apply throughout an epoch. Authentication guarantees for the Client Traffic should definitely not apply to the Initial Traffic of Fig. 4, which lies on the other side of an epoch boundary from the authenticated traffic.

The interface between the protocol and its application, then, is an essential part of expressing the guarantees that the application should rely on. We can enlarge our notion of "protocol" to include signals across the API as well as the message roles. This enlarged protocol can itself be used to define a goal language. Some goal formulas refer only to API events and their parameters, not to the nuts-and-bolts events of the core protocol itself. These formulas express API-level goals. They are of course true only if the lower level protocol behaves properly. However, their content speaks explicitly only about the events of interest to the upper level.

Such a language allows us to apply our notions of "good enough" and "at least as good as", showing that they are relevant to our enlarged, API-aware protocols.

Representing APIs. An API is a set of signals and commands that may occur in a certain pattern, between an application and the service that implements the API. In our case, the service directly controls actual protocol interactions. The API thus consists of the signals (from service to application) and commands (from application to service), and how the reception of those signals and commands line up with the implemented protocol behavior.

The communication between service and application is of a different nature than the communication that takes place in protocol execution: in particular, this communication is not observable or controllable by a typical network adversary.

We enlarge our notion of the protocol to one that includes the signals and commands as well as how they interact with protocol messages. This enlarged notion of the protocol allows us to describe a goal language. In that language, an *application goal* is a goal expressible in terms directly referring to application-level roles.

Enrich-by-need analysis for APIs. In order to evaluate "good enough" or "as good as" for application goals, we need an enrich-by-need analysis that respects the distinct nature of communication between the API and the protocol service. There is some recent research on analyzing protocols with state that could be relevant. But no special machinery is required beyond protocol messages: all we need to do is emulate the information passed between application and service as a secure channel independent from all others involved in the protocol.

Let Π be a protocol. An *API-enhanced version* of Π is a protocol Π' that has a set of new nodes api such that the result of omitting the nodes api from Π' yields Π. A goal formula Φ is an *API goal* if it refers only to nodes in api, and their parameters.

If Φ is an API goal then its truth or falsehood can be established by an appropriate enrich-by-need analysis of Π'. In other words, "good enough" can be established through enrich-by-need analysis just as was the case for protocol-level goals.

Furthermore, for any particular API goal, its antecedent references a certain subset of API role events and variables. Thus, it can be meaningful to compare two APIs (even for exactly the same underlying protocol) in terms of goals that are API goals for both APIs. In such circumstances, our notion of "as good as" applies here.

TLS renegotiation, revisited. Now we state an example of an application-level goal for TLS that addresses the interface concerns specific to the renegotiation flaw. In particular, the authors of [19] point out the dangers of an application being unaware of renegotiation. The flaw that arises from the renegotiation attacks is most easily understood from the application level. Here, we describe that goal in a formal manner.

The application is aware of data being exchanged over a TLS connection, and may also query for the status of the connection. Consider the following set of predicates:

- $\texttt{DataSend}(n)$: a command was issued at node n to send data over TLS.

- $\texttt{DataRecv}(n)$: a signal was received at node n that data was received from TLS.
- $\texttt{DataVal}(n, d)$: d is the data involved in the `DataSend` or `DataRecv` event occurring at node n.
- $\texttt{Self}(n, ID)$: ID is the identifier of the actor at node n.
- $\texttt{Status}(n)$: a status signal was received at node n about a TLS connection.
- $\texttt{EID}(n, eid)$: eid is the epoch ID involved in the `DataSend`, `DataRecv`, or `Status` event occurring at node n.
- $\texttt{Client}(n, cID)$: cID is either the ID of the client reported as the authenticated client in a status event, or "anon" otherwise.
- $\texttt{Server}(n, sID)$: sID is the ID of the server reported in a status event.

Informally, the goal we will describe is that if the server receives d over some TLS connection, and also gets a report about the status of that same connection, then either the status report identifies the client as anonymous, or the identified client actually sent the data d. Formally,

$$\begin{aligned} & \texttt{DataRecv}(n) \wedge \texttt{DataVal}(n, d) \wedge \texttt{Self}(n, s) \wedge \texttt{EID}(n, eid), \\ \wedge \quad & \texttt{Status}(m) \wedge \texttt{Server}(m, s) \wedge \texttt{Client}(m, c) \wedge \texttt{EID}(m, eid) \\ \Rightarrow \quad & c = \text{“anon”} \vee \\ & \exists n' : \texttt{DataSend}(n') \wedge \texttt{Self}(n', c) \wedge \texttt{DataVal}(n', d) \wedge \texttt{EID}(n', eid) \end{aligned}$$

for all values of the free variables.

Here the first line of the hypothesis assumes that a data reception signal occurred for s at node n, involving data d, epoch identifier eid. The second line assumes that a status was checked for epoch identifier eid, and the signal was received at node m and indicates that the client is c and the server is s. If both of these conditions are met for a common eid, the goal states that either c is "anon" (indicating that the client is not authenticated), or otherwise that the client c actually sent d. This last claim is the mirror-image of the first line: namely, that a data transmission command occurred for c at node n, involving data d and epoch identifier eid.

Note that the goal does not specify that $\texttt{Preceq}(m, n)$: in other words, the status may be reported even after the data is received and we still expect the status to reflect an accurate assessment of the identity of the client if the client is not anonymous. This is precisely what goes wrong in the renegotiation attack: the adversary initiates an anonymous session, causes data to be received, and then convinces an honest client to renegotiate the session so that it is later reported as authenticated.

Goals and mandates. Describing the goal for TLS at this level is natural, but the discussion ultimately must match the mandate of the standard itself: to specify the actual protocol messages and to advise about how applications are to be informed on its use. Delving into details of the interface, in this case, is not appropriate (but if it was, the notions of "good enough" and "as good as" apply just as well to the more complex interface-inclusive protocol model). However,

stakeholders present in the discussion will be able to comment on the constraints they are under.

One particular constraint is that a TLS API will ultimately aim to set up a simple type of data stream functionality in which status issues are separated from data signals. In other words, the `Status` and `DataRecv` events cannot occur at the same node. Another important constraint is that status reports be limited to a current status, so that the API is not responsible for maintaining an exhaustive status history.

7 Related Work and Conclusion

The full literature on the use of formal methods for analyzing cryptographic protocols is too vast to summarize here, although we would direct the reader to [15] for a (now classic) survey. As methods and tools have become more developed, they have been effectively applied to the analysis of published standards, demonstrating their maturity and applicability [14, 16, 1, 2, 4]. Many of these efforts have explicitly engaged with the the relevant standards body to ensure their input was reflected in the standard.

We are not the only ones to propose a logical language of security goals. Numerous efforts attempt to use a formal logic in which to reason directly about cryptographic protocols [3, 6]. While they do provide formal statements of security goals, the proof methods do not lend themselves to natural comparisons of the goals that various protocols might achieve. [12] contains a hierarchy of authentication goals demonstrating the relationship between the goals themselves, and [5, 1] integrate the hierarchy with an enrich-by-need analysis method.

The process of standardizing cryptographic protocols is both difficult and important. Getting a variety of stakeholders to converge on a single point of view requires careful consideration of all proposed options and a clear way of comparing them. Although many unchangeable constraints exist pertaining to issues such as efficiency, computational limitations, or backwards compatibility, the security of the designed protocol is typically of paramount importance. Unfortunately, the term "security" can mean different things in different contexts. A clear and precise formulation of the security requirements of a protocol can help focus group discussions on the precise outcomes that it is most important for a protocol to achieve. It can also help to distinguish those constraints that pertain to security from operational constraints, allowing committees to better understand the space of trade-offs for design decisions.

In this paper we demonstrated how automated tools based on formal methods can assist in this complicated decision-making process. We presented a formal language in which to express security goals. We focused on how the enrich-by-need method of protocol analysis integrates with this goal language and demonstrated its applicability to the historical case of mitigating a flaw in the PKINIT protocol. We then demonstrated how the goal language might be adapted to accommodate goals that lie at the intersection of security protocols and the ap-

plications they support, reinforced by the example of a previously discovered flaw in TLS renegotiation.

While we believe the goal language paired with enrich-by-need protocol analysis is particularly interesting, we also believe other formal languages and tools could be used in a quite similar way. The precision and clarity that comes from the abstraction of formal methods must be balanced against the practical considerations of potential implementations. We present here a vision for how the results of formal analyses can be incorporated with real-world decision making processes to focus discussion and strengthen the security of the resulting standards.

References

1. 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.
2. 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.
3. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM TRANSACTIONS ON COMPUTER SYSTEMS*, 8:18–36, 1990.
4. 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.
5. Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Springer, 2012.
6. Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (PCL). *Electr. Notes Theor. Comput. Sci.*, 172:311–358, 2007.
7. 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.
8. Daniel J. Dougherty and Joshua D. Guttman. Decidability for lightweight Diffie-Hellman protocols. In *IEEE Symposium on Computer Security Foundations*, 2014.
9. 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.
10. 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.
11. Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.
12. Gavin Lowe. A hierarchy of authentication specification. In *CSFW*, pages 31–44, 1997.
13. C. Meadows. The NRL protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.

14. Catherine Meadows. Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer. In *IEEE Symposium on Security and Privacy*, pages 216–231, 1999.
15. Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, 2003.
16. John C. Mitchell, Arnab Roy, Paul Rowe, and Andre Scedrov. Analysis of EAP-GPSK authentication protocol. In *ACNS*, pages 309–327, 2008.
17. 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.
18. John D. Ramsdell and Joshua D. Guttman. CPSA: A cryptographic protocol shapes analyzer, 2009. `http://hackage.haskell.org/package/cpsa`.
19. E. Rescorla, M. Ray, S. Dispensa, and N. Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. RFC 5746 (Proposed Standard), February 2010.
20. 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.
21. 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.
22. L. Zhu and B. Tung. Public Key Cryptography for Initial Authentication in Kerberos (PKINIT). RFC 4556 (Proposed Standard), June 2006. Updated by RFC 6112.