Cryptographic Protocol Composition via the Authentication Tests*

Joshua D. Guttman

The MITRE Corporation

Abstract. Although cryptographic protocols are typically analyzed in isolation, they are used in combinations. If a protocol was analyzed alone and shown to meet some security goals, will it still meet those goals when executed together with a second protocol? While not every choice of a second protocol can preserve the goals, there are syntactic criteria for the second protocol that ensure they will be preserved. Our main result strengthens previously known criteria.

Our method has three main elements. First, a language $\mathcal{L}(\Pi)$ in classical logic describes executions of a protocol Π, and expresses its security goals. Strand spaces provide the models for $\mathcal{L}(\Pi)$.

Second, the strand space "authentication test" principles suggest our syntactic criterion for security goals to be preserved.

Third, certain homomorphisms among models for $\mathcal{L}(\Pi)$ preserve the truth of formulas of the syntactic form that security goals take. This provides a way to extract—from a counterexample to a goal that uses both protocols—a counterexample using only the first protocol.

1 Introduction

Protocol analysis focuses largely on understanding the secrecy and authentication properties of individual, finished protocols. There is a good reason for this: Each security goal then definitely either holds or does not hold. However, an analysis is more reusable and of higher value, if we know which of its results will remain true after combination with other protocols, and perhaps other kinds of extensions to the protocol.

In practice, every protocol is used in combination with other protocols, often with the same long-term keys. Also, many protocols contain messages with "blank slots." Higher level protocols piggyback on them, filling the blank spots with their own messages. We want to find out when the goals that hold of a protocol on its own are preserved under combination with other protocols, and when these blanks are filled in.

Two results on composition. Two existing results, both within the Dolev-Yao model [13], are particularly relevant. We showed [17] that if two protocols manipulate disjoint sets of ciphertexts, then combining the protocols cannot undermine their security goals. We called this the *disjoint encryption* property.

* Supported by MITRE-Sponsored Research, 08-1441. Email: guttman@mitre.org.

A careful, asymmetric formulation of "disjoint encryption" allowed us to show that one protocol Π_1 may produce ciphertexts—in a broad sense including digital certificates as well as Kerberos-style tickets—consumed by another protocol Π_2, without Π_2 undermining any security goal of Π_1. The relation is asymmetric because faulty production of these ciphertexts by Π_1 may certainly undermine goals of Π_2.

Our result concerned only protocols that completely parse the messages they receive to atomic values, and do not handle messages containing unstructured blank slots. A second limitation was that it covered only protocols using atomic keys, not keys produced (e.g.) by hashing compound messages. A recent result by Cortier, Delaitre and Delaune [9] lifts these two limitations, but only in the symmetric case, where neither protocol produces ciphertexts that may be consumed by the other.

One stimulus for the present paper was to combine these two results.

Our approach. We view protocol executions—more specifically, the parts carried out by the rule-abiding, regular participants, but not the adversary—as forming objects we call *skeletons* [12]. A skeleton containing enough protocol behavior to have occurred, when combined with some adversary behavior, is called a *realized* skeleton. If more information about regular behavior is required to obtain a possible execution, then the skeleton is *unrealized*.

In this paper, we introduce a first order language $\mathcal{L}(\Pi)$ to describe skeletons of each protocol Π. The skeletons containing regular behaviors of Π provide a semantics, a set of models, for formulas of $\mathcal{L}(\Pi)$. Security goals are formulas G, of specific forms, belonging to $\mathcal{L}(\Pi)$. A skeleton $\mathbb{A}$ is a counterexample to a goal G when it is realized, but it satisfies G's negation, $\mathbb{A} \models \neg G$.

When we try to show that a protocol Π_2 does not undermine any security goal of a protocol Π_1, we are interested in two languages, namely $\mathcal{L}(\Pi_1)$ and $\mathcal{L}(\Pi_1 \cup \Pi_2)$. $\mathcal{L}(\Pi_1)$ is a sublanguage of $\mathcal{L}(\Pi_1 \cup \Pi_2)$, so that the security goals G_1 of $\mathcal{L}(\Pi_1)$ are some of the security goals of the larger language $\mathcal{L}(\Pi_1 \cup \Pi_2)$. Likewise, the skeletons of Π_1 are some of the skeletons of $\Pi_1 \cup \Pi_2$, namely those in which no Π_2 activity occurs.

Suppose that G_1 is a goal in $\mathcal{L}(\Pi_1)$, and $\mathbb{A} \models \neg G_1$ is a counterexample using behavior of $\Pi_1 \cup \Pi_2$. We are seeking a syntactic condition on Π_1, Π_2 such that we can always extract, from this $\Pi_1 \cup \Pi_2$-counterexample, a skeleton containing only Π_1 activity which is already a counterexample. So Π_2 behavior was not needed to build the counterexample; Π_1 alone already undermined the goal G_1.

In this paper, we define a syntactic relationship called *strong disjointness* between protocols Π_1, Π_2. When two protocols are strongly disjoint, then Π_2 does not undermine any goals G_1 achieved by Π_1 in isolation. The authentication test principles suggest both the definition of strong disjointness, and also the operations on skeletons that allow us to extract a suitable Π_1-counterexample $\mathbb{A}_1$ given a $\Pi_1 \cup \Pi_2$-counterexample $\mathbb{A}$.

In particular, there are two steps to extracting $\mathbb{A}_1$. One is simply a restriction that omits non-Π_1 behavior. The other is a step of generalization or removal, in which we insert blank slots in place of all encrypted units that do not belong

specifically to Π_1. Each of these operations preserves satisfaction for negated security goals. Together, they yield a realized Π_1 skeleton from any realized $\Pi_1 \cup \Pi_2$ skeleton, hence together they preserve counterexamples.

This approach, which is possibly of interest in itself, is essentially model-theoretic. Model theory characteristically combines two elements. The first consists of algebraic relations (embeddings and restrictions, automorphisms, homomorphisms, etc.) among the structures interpreting a logic. The second is the syntactic formulas, often focusing on formulas of particular logical forms. The expressive power of a set of formulas is limited when those formulas do not distinguish among certain models, i.e. the models satisfy the same formulas in this set. Typical model-theoretic results combine these two ingredients; a simple example is the fact that a homomorphism between two structures for first order logic preserves satisfaction of atomic formulas.

A second stimulus for the current paper was to illustrate the power of this model-theoretic approach to security protocols.

Structure of this paper. In Section 2, we provide an example. It is intended to illustrate the importance of the asymmetric relation between protocols Π_1, Π_2. It also illustrates the authentication tests and the intuitive disjointness property. Section 3 provides background on messages, strands, protocols, and homomorphisms. It is adapted from [12, 11]. One fine point distinguishing those two papers is that [12] worked in a framework without blank slots in protocols, while [11] established its completeness result for a stronger framework that includes blank slots. Section 4 summarizes the authentication tests, and the completeness result of [11], although in a far more compact and abstract form. This reformulation is another small contribution of the present paper. Section 5 formalizes the "multiprotocols" that we have informally written in the form $\Pi_1 \cup \Pi_2$, and proves some useful facts about the removal or generalization operator on skeletons.

In Section 6 we define the strong disjointness property. We prove a key result about realized skeletons $\mathbb{A}$ for a multiprotocol $\Pi_1 \cup \Pi_2$ with strong disjointness. Namely, restricting $\mathbb{A}$ to Π_1, and then generalizing by removing all encryptions that belong to Π_2, yields a realized skeleton $\mathbb{A}_1$ as its extracted result. Section 7 presents the languages $\mathcal{L}(\Pi)$ for expressing security goals G, and their skeleton-based semantics. We show that when $\mathbb{A} \models \neg G$, then for the extracted result $\mathbb{A}_1$, $\mathbb{A}_1 \models \neg G$, which is the main result of the paper.

Section 8 discusses some additional related work, particularly the Protocol Composition Logic of [10], and mentions future work on more general forms of protocol elaboration.

2 Example: Anonymous identities in trusted computing

We consider as an example a certificate distribution protocol for Trusted Platform Modules (TPMs). It allows a machine with a TPM to retrieve a certificate for a newly generated public key K, binding K to a made-up identity I, but ensuring that the corresponding private key K^{-1} is resident within a TPM. A

zero-knowledge protocol would be desirable for this purpose, and although such a protocol exists [6], called Direct Anonymous Attestation, it is neither simple nor indeed perfectly correct [3].

The earlier protocol that we describe is a conventional, certificate-based protocol [5], in which a certifying authority called a Privacy Certificate Authority (PCA) binds I, K, allowing K to be used as an "anonymous identity key" (AIK). If the public part of an AIK is properly certified, then its corresponding private part should be securely stored within some TPM. Thus, it can be used only in accordance with certain rules, and with less risk of compromise. Second, one should be unable to link any AIK with a particular TPM without the assistance of the TPM itself or the certifying PCA. Thus, the AIK provides the owner of the computer containing the TPM a guarantee of privacy or anonymity, while providing the consumer of statements signed with it the assurance that they were prepared within some genuine TPM.

Before creating this anonymous identity credential (AIC), the issuing PCA checks an "Endorsement Credential" signed by the TPM manufacturer. The EC asserts—about a public encryption key EK—that its inverse EK^{-1} is a TPM-resident private key. Since the AIC is transmitted encrypted with EK, the AIC can only be decrypted and freed within that TPM. The TPM is designed to verify that K^{-1} is available within this TPM, and is a suitable type of key, before releasing the AIC. A modified version of the protocol is shown in Fig. 1. We have omitted some details, and added a parameter x, which may be used to

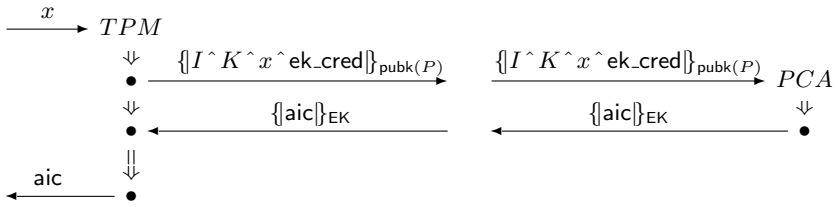

Fig. 1. TCG Anonymous Identity Protocol (modified)

restrict the use of the resulting AIC, for instance determining when it expires, or limiting it to specific later protocols. In this protocol, $\{|m|\}_{\mathsf{pubk}(A)}$ refers to m encrypted with a public key for which A holds the decryption key, and $[\![m]\!]_{\mathsf{sk}(A)}$ refers to m accompanied by a digitally signed hash, prepared using a signature key held by A. The tags ekctag and aictag are distinguishing bitpatterns that make an EC and an AIC distinguishable from other cryptographically prepared values.

Security Goals of AIP. There are two main goals for the protocol, which should hold whenever an AIC is observed, bearing the signature of an uncompromised PCA. The first is that there should have been a run of the PCA role that produced this certificate, and transmitted it encrypted with some key EK. The second is that there should have been a run of the TPM role, which received the AIC encrypted with this same EK, and retransmitted the AIC in the clear. These two goals are authentication goals, since they assert that uncompromised (regular) principals executed certain actions.

This protocol does not have confidentiality goals, since none of the values it transmits is a secret. The EC is transmitted only under encryption, to provide some protection, but nothing in the protocol depends on this public key certificate remaining undisclosed.

AIP and other protocols. This protocol is useful only if the resulting AIC can be used. Thus, the AIC must be consumed in other protocols, in which the keys K, K^{-1} are used.

As an example, suppose that the owner of K would like to use it for writing electronic checks against an account he holds with a bank B. He would like a protocol to prove possession of K^{-1} to B, thereby obtaining a check-writing certificate for K certified by B. This check-writing certificate would in turn be used in a protocol to prepare individual checks for particular purchases. A merchant would be able to associate a check with a particular individual only with B's collusion. It would be able to associate the check with a particular TPM only with the collusion of both B and the PCA.

A protocol, of the kind we have described, to retrieve a check-writing certificate has two crucial characteristics:

1. the protocol must consume an AIC, to ensure that the signature key K^{-1} is a genuine TPM-resident key;
2. the protocol cannot create new AICs, so that every AIC must still be generated as a result of a run of the original AIP.

The first condition implies that the Cortier-Delaitre-Delaune criterion [9] cannot apply to the AIP and our check-writing protocol. Moreover, it provides no guidance as to how to design successive stages in a protocol suite so as not to undermine the guarantees achieved by their predecessors.

The second condition tells us something crucial about how to achieve protocol independence. The AIC is transformed twice in this protocol: once, it is transformed from a state of non-existence into existence, but transmitted only in the special form $\{|\mathsf{aic}|\}_{\mathsf{EK}}$. This transformation is carried out by the PCA. Second, the client TPM transforms the AIC by extracting it from its encryption; it checks for the presence of K^{-1} before emitting aic. In order to respect condition 2, a secondary protocol must avoid exactly these transformations. It must never create an AIC; and it must never extract an AIC from an encrypted form.

Unfortunately, the Guttman-Thayer criterion [17] does not apply because of the unstructured blank slot x in the protocol.

There are two benefits of a systematic way to ensure that multiple protocols will not interfere. One benefit is that it simplifies protocol verification. The primary protocol is verified once in isolation. A second protocol that consumes its cryptographic outputs must be verified in combination with it; but the verification is independent of future protocols to be added later. In particular, state-space exploration methods for protocol verification (e.g. [18, 2]) have a smaller job, and invariant-based proof methods (e.g. [19]) have smaller case splits.

However, a second larger benefit, in our opinion, is that a protocol composition criterion provides a guide for designing supplementary protocols. The secondary protocol has specific rules it must "play by" in consuming values produced in the primary protocol, and in using keys embedded within those values.

How not to interfere with AIP. AIP depends on two central transformations. The first creates an aic, but only in the encrypted form $\{|\mathsf{aic}|\}_{\mathsf{EK}}$, and only after the endorsement credential has been checked. The second transformation extracts aic from its encrypted form, and does so only if the private part K^{-1} of the certified key is actually resident in the TPM performing the decryption. Thus, even if an adversary capturing an endorsement credential, he cannot trick the TPM containing EK^{-1} into freeing aics.

If AIP were supplemented with a protocol executing either of these transformations, then AIP's guarantees might fail. For instance, if any run of a supplementary protocol created any message that is also an instance of the form aic, then a consumer of these credentials could be misled. Or suppose that any run of a supplementary protocol—having received anything of the form aic only within an instance of $\{|\mathsf{aic}|\}_{\mathsf{EK}}$—could emit it outside this encryption. Then an aic might become available without a TPM having checked that K^{-1} is resident.

Hence, to avoid interference, the secondary protocol should never transform any of the "critical values" on which the primary protocol depends. This check is syntactic in nature: unification can identify locations where a critical value of the primary protocol might be created in a secondary protocol, or where a critical value, having been received by a run of a secondary protocol, might be retransmitted in any new form.

Crucial intuition. Thus, to ensure that a secondary protocol does not undermine security goals of a primary protocol, we must design it not to transform units belonging to the primary protocol. A primary unit may be an encryption e with an ingredient $t_1 \sqsubseteq e$ which some transformation will extract from e. This was the case with the transformation carried out by the TPM in the AIP protocol; i.e. e is $\{|\mathsf{aic}|\}_{\mathsf{EK}}$ and t_1 is aic. Alternatively, a primary unit may be an encryption e where the relevant transformation is to create a value of the form e. This was the case with the transformation carried out by the Privacy Certificate Authority when it first created the cryptographic value $\mathsf{aic} = [\![\, \mathsf{aictag}\ I \mathbin{\hat{}} K \mathbin{\hat{}} x \,]\!]_{\mathsf{sk(PCA)}}$, although in this example e happens to be created inside an encryption $\{|\mathsf{aic}|\}_{\mathsf{EK}}$.

In order to formalize this idea, we will define the key strand space ideas, including the authentication test principle.

3 Strands and other preliminaries

We represent the messages as an algebra A, freely generated by the two operations of tagged concatenation $\mathsf{tagname}\ t_0 \hat{}\ t_1$ and encryption $\{|t_0|\}_{t_1}$. Concatenations $\mathsf{nil}\ t_0 \hat{}\ t_1$ with the distinguished tag nil are written $t_0 \hat{}\ t_1$ without the tag. The operations generate A starting from from a set of *atoms* and from *indeterminates*.

The indeterminates represent blank slots, and they may be replaced by any message in A, like unsorted variables. The atoms are partitioned into sorts *atomic key*, *nonce*, *text*, *name*, etc. The operators $\mathsf{sk}(a)$ and $\mathsf{pubk}(a)$ map names to signature keys and public encryption keys respectively, and K^{-1} which maps an asymmetric atomic key to its inverse, and a symmetric key to itself. Non-atomic messages can also be used as keys, in which case they act as symmetric keys.

One message t_0 may be an *ingredient* in another t, written $t_0 \sqsubseteq t$, where $\sqsubseteq$ is the smallest reflexive, transitive relation on messages such that $t_0 \sqsubseteq t_0 \hat{}\ t_1$, $t_1 \sqsubseteq t_0 \hat{}\ t_1$, and $t_0 \sqsubseteq \{|t_0|\}_{t_1}$. The key used in an encryption is not an ingredient, but an aspect of how the ciphertext was prepared.[1] Observe also that $a \not\sqsubseteq \mathsf{sk}(a)$.

We view each message in A as an abstract syntax tree where the vertices are of four kinds, namely atoms, indeterminates, encryptions, and concatenations. Atoms and indeterminates are vertices that are always leaves of the tree, and these vertices are labeled with the particular atom or indeterminate involved. The internal vertices—concatenations and encryptions—each have two children. In an encryption vertex, the edge leading to the first child is the *plaintext edge* and the edge leading to the second child is the *key edge*. In a concatenation vertex, the vertex is labeled with a tag $\mathsf{tagname}$.

A *path* as a sequence of choices $\{\ell, r\}$. A path *leading to* a node, starting from t_1, is defined in the usual inductive way. We write $@_p(t_1)$ for the node to which p leads, starting from t_1. When p does not lead to any node starting from t_1, then we write $@_p(t_1) \uparrow$. This is the case when p is too long, because a proper subpath of p has reached a leaf.

The ingredient relation $t_0 \sqsubseteq t$ means that, for some p, $@_p(t_1) = t_0$, and p does not traverse any key edge.

Definition 1. Let S be a set of encryptions. A message t_0 *is found only within* S *in* t_1, written $t_0 \odot^S t_1$, iff for every path p such that $@_p(t_1) = t_0$, either (1) p traverses a key edge or else (2) p has a proper subpath p_1 such that $@_{p_1}(t_1) \in S$.

Message t_0 *is found outside* S *in* t_1, written $t_0 \dagger^S t_1$, iff it is not the case that $t_0 \odot^S t_1$. □

Thus, $t_0 \dagger^S t_1$ iff there exists a path p where $@_p(t_1) = t_0$, and (1) p traverses no key edge, and (2) p traverses no member of S before reaching t_0. If $t_0 \not\sqsubseteq t_1$, then $t_0 \odot^S t_1$ for all sets of encryptions S. Indeed, $t_0 \not\sqsubseteq t_1$ iff $t_0 \odot^\emptyset t_1$.

We formalize "transformation" using this notion. A role transforms a value t_0 if it transmits a message t_1 where $t_0 \dagger^S t_1$, when until this transmission, the role

[1] $\sqsubseteq$, long called the subterm relation in the strand space literature, frequently caused terminological confusion. Possibly the word "ingredient" will be better.

had sent and received t_0 in messages where $t_0 \odot^S t_1$. We express two apparently different things using the same terminology here.

- When $S \neq \emptyset$, the encryptions in S describe envelopes protecting t_0. When a message t is sent in which $t_0 \dagger^S t$, the sender has transformed t_0 by *removing* it from these enveloping forms. We also sometimes speak of transforming a member of S by removing an ingredient t_0 from it.
- When $S = \emptyset$, if $t_0 \odot^\emptyset t_1$ for all the previous messages t_1, and if a subsequent message t is sent in which $t_0 \dagger^\emptyset t$, then the sender has *created* the value t_0.

Homomorphisms. A *homomorphism* α on A is a function that maps atoms to atoms of the same sort, and indeterminates to any message in A, where α commutes with the operators $\mathsf{sk}(a)$, $\mathsf{pubk}(a)$, K^{-1}, $\mathsf{tagname}\ t_0 \hat{}\ t_1$, and $\{|t_0|\}_{t_1}$. So

$$\alpha(\{|t_0|\}_{t_1}) = \{|\alpha(t_0)|\}_{\alpha(t_1)}$$
$$\alpha(\mathsf{tagname}\ t_0 \hat{}\ t_1) = \mathsf{tagname}\ \alpha(t_0) \hat{}\ \alpha(t_1).$$

A homomorphism is determined by its action on atoms and indeterminates. In the tree model of messages, to apply a homomorphism means to walk through, copying the tree, but inserting $\alpha(a)$ every time an atom a is encountered, and inserting $\alpha(x)$ every time that an indeterminate x is encountered.

If $t_1 = \alpha(t_0)$, then each path p within t_1 is of one of two forms. If p is a path within t_0, and $@_p(t_0) = t$, then $@_p(t_0) = \alpha(t)$. Otherwise p is too long. In this case, $p = p_0 \frown p_1$, where $@_p(t_0) = x$ leads to an indeterminate x within t_0, and p_1 is a path within $\alpha(x)$. Then $@_{p_1}(\alpha(x)) = @_p(t_1)$.

We use homomorphisms for many purposes, e.g. to represent unifiers. We lift their action to larger objects that contain messages, including the local sessions of principals, which we represent as *strands*, and global executions, which we formalize as *realized skeletons*.

Strands and origination. We represent the behavior of a single principal in a single run of a protocol as a *strand*, i.e. as a finite sequence of nodes, each of which either transmits or receives some message. We regard transmission nodes as positive and reception nodes as negative. For instance, in Fig. 1, the two vertical columns of nodes connected by double arrows $\Rightarrow$ are strands.

A message t_0 *originates* at a node n_1 if (1) n_1 is a transmission node; (2) $t_0 \sqsubseteq \mathsf{msg}(n_1)$; and (3) whenever $n_0 \Rightarrow^+ n_1$, $t_0 \not\sqsubseteq \mathsf{msg}(n_0)$. Thus, t_0 was transmitted without having been either received or transmitted previously on the same strand. Values assumed to originate only on one node in an execution—"uniquely originating" values—formalize the idea of freshly chosen, unguessable values.

A looser relation than the ingredient relation $\sqsubseteq$ between terms and their parts is also useful. The *appears in* relation $\ll$ between messages is the smallest reflexive, transitive relation on messages such that $t_0 \ll t_0 \hat{}\ t_1$, $t_1 \ll t_0 \hat{}\ t_1$, and both $t_0 \ll \{|t_0|\}_{t_1}$ and $t_1 \ll \{|t_0|\}_{t_1}$. Viewing the terms as abstract syntax trees, $t_0 \ll t$ means that there is a path through the tree representing t that reaches a vertex representing t_0. It is permitted to traverse key edges.

Protocols. A *protocol* Π is a finite set of strands, representing the roles of the protocol. Two of the roles of the AIP are the strands shown in Fig. 1. The executions of a role $\rho \in \Pi$ are obtained by replacing values appearing in ρ by suitable values. For example, in Fig. 1, we may replace I, K, etc., by any name, asymmetric key, etc., and we may replace x by any (possibly compound) message. Thus, the instances of the roles $\rho \in \Pi$ are their images $\alpha(\rho)$ under homomorphisms α of the algebra A.

We impose a condition on how roles use indeterminates, which represent the "blank slots" in which a protocol participant manipulates a value without parsing it to atomic components. The indeterminates represent values received from protocol peers, or passed down from higher-level protocols as parameters. Thus, we stipulate:

If n_1 lies on $\rho \in \Pi$, and for some indeterminate x, $x \ll \mathsf{msg}(n_1)$,
then there is a negative node $n_0 \Rightarrow^* n_1$ such that $x \sqsubseteq \mathsf{msg}(n_0)$.

This ensures that an indeterminate is received as an ingredient in an incoming message before being sent as an ingredient in an outgoing message. The initial node on the TPM AIC role in Fig. 1 represents the indeterminate x being received from the higher level protocol that has invoked the TPM activity.

We assume that every protocol contains a special role, called the *listener* role $\mathsf{Lsn}[y]$ which has a single reception node in which y is received. The instances of this role are all strands containing a single reception event, for any message; they are used to document the fact that this value is available without any cryptographic protection. For instance, the instance $\mathsf{Lsn}[K]$ documents that the key K is compromised, since it is available (hence also available to the adversary) without being protected by any encryption. The three roles of AIP are the listener role together with the two shown in Fig. 1.

A principal executing a role such as the PCA's role in AIP may be partway through its run; for instance, it may have executed the first receive event without "yet" having executed its second event, the transmission node.

Skeletons. A *skeleton* $\mathbb{A}$ consists of any number of (possibly partially executed) role instances, which we represent as a finite set of nodes, $\mathsf{nodes}(\mathbb{A})$, annotated with three additional kinds of information:

1. A partial ordering $\preceq_{\mathbb{A}}$ on $\mathsf{nodes}(\mathbb{A})$;
2. A finite set $\mathsf{unique}_{\mathbb{A}}$ of nonces and atomic keys assumed uniquely originating in $\mathbb{A}$;
3. A finite set $\mathsf{non}_{\mathbb{A}}$ of atomic keys assumed non-compromised in $\mathbb{A}$.

We stipulate that $\mathsf{nodes}(\mathbb{A})$ is compatible with the strand behavior, and the ordering, in the sense that $n_1 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \Rightarrow n_1$ implies $n_0 \in \mathsf{nodes}(\mathbb{A})$ and $n_0 \preceq_{\mathbb{A}} n_1$. We also stipulate that if $a \in \mathsf{unique}_{\mathbb{A}}$, then a originates at most once in $\mathsf{nodes}(\mathbb{A})$. If $a \in \mathsf{non}_{\mathbb{A}}$, then we assume that a originates nowhere in $\mathsf{nodes}(\mathbb{A})$, though it or its inverse may have been used to prepare some cryptographic value appearing in $\mathsf{nodes}(\mathbb{A})$.

A skeleton is *realized* if the behavior it describes can occur without any additional activity of the regular participants. In particular, $\mathbb{A}$ is realized if, for every reception node n, the adversary can construct $\mathsf{msg}(n)$ using as inputs:

1. all messages $\mathsf{msg}(m)$ where $m \prec_{\mathbb{A}} n$ and m is a transmission node;
2. any atomic values a such that $a \notin (\mathsf{non}_{\mathbb{A}} \cup \mathsf{unique}_{\mathbb{A}})$, or such that $a \in \mathsf{unique}_{\mathbb{A}}$ but a originates nowhere in $\mathbb{A}$

using the adversary strands given in Definition 2.

Definition 2. A *penetrator strand* has trace of one of the following forms:

M_a: $\langle +a \rangle$ where a is atomic	M_g: $\langle +g \rangle$ where g is an indeterminate
$\mathsf{C}_{g,h}$: $\langle -g,\ -h,\ +g \mathbin{\hat{}} h \rangle$	$\mathsf{S}_{g,h}$: $\langle -g \mathbin{\hat{}} h,\ +g,\ +h \rangle$
$\mathsf{E}_{h,K}$: $\langle -K,\ -h,\ +\{\!|h|\!\}_K \rangle$	$\mathsf{D}_{h,K}$: $\langle -K^{-1},\ -\{\!|h|\!\}_K,\ +h \rangle$

A skeleton $\mathbb{A}$ is a *skeleton over* a particular protocol Π if every strand with nodes in $\mathsf{nodes}(\mathbb{A})$ is an instance of some role of Π.

If α is a homomorphism on A, we can sometimes regard it as a function on skeletons by applying α to the messages on all nodes of $\mathbb{A}$, as well as to the sets $\mathsf{unique}_{\mathbb{A}}$ and $\mathsf{non}_{\mathbb{A}}$. Naturally, α must respect the clauses for $\mathsf{unique}_{\mathbb{A}}$ and $\mathsf{non}_{\mathbb{A}}$ in the definition of skeleton; i.e. α cannot identify $K \in \mathsf{non}_{\mathbb{A}}$ with any atom that originates in $\mathbb{A}$, or identify $a \in \mathsf{unique}_{\mathbb{A}}$ with another atom if this would give $\alpha(a)$ two originating nodes in the resulting skeleton. We write $\alpha(\mathbb{A})$ for the resulting skeleton, when this is defined, and regard α as a homomorphism on skeletons. In [12] we use a compatible although more inclusive notion of homomorphism.

4 Authentication tests

The authentication test idea is that when certain transformations have been observed, they represent a kind of test that can be explained or "solved" in only a few ways.

Definition 3. A *cut* in a skeleton $\mathbb{A}$ is a partition of $\mathsf{nodes}(\mathbb{A})$ into two sets L, U such that (1) U is non-empty, and (2) if $m \in L$ and $n \in U$, then $n \not\preceq_{\mathbb{A}} m$. □

Suppose either $c = \{\!|t_0|\!\}_{t_1}$ is an encrypted message, or $c = a$ is an atom where $c \in \mathsf{unique}_{\mathbb{A}}$ originates uniquely on some regular node, and suppose that S is a set of encrypted messages such that $c \dagger^S \mathsf{msg}(n)$ for some $n \in \mathsf{nodes}(\mathbb{A})$. Letting

$$U = \{n \in \mathsf{nodes}(\mathbb{A}) \colon \exists m \,.\, m \preceq_{\mathbb{A}} n \wedge c \dagger^S \mathsf{msg}(m)\},$$

and $L = \mathsf{nodes}(\mathbb{A}) \setminus U$, it follows that L, U is a cut in $\mathbb{A}$. This cut separates all the nodes where c has never previously been found outside S from those nodes in which this is no longer true.

The authentication test principle states that there are only three ways that this can happen, i.e. that in a *realized* skeleton, one of three kinds of action must account for c "escaping" from the protection afforded to it by the set of encryptions S:

1. some regular transmission node sends c outside S; or
2. $c = \{\!|t_0|\!\}_{t_1}$ and the encryption key t_1 to construct c from t_0 was compromised prior to c's escape, as may be documented with a listener node; or else
3. a decryption key K^{-1} was compromised prior to c's escape—as may likewise be documented in a listener node that receives K^{-1}—and it allows the adversary to extract c from some member $\{\!|t|\!\}_K$ of S.

In fact, all the protocol analysis possible within this simple Dolev-Yao model is a systematic exploration of these cases [11].

Definition 4. Let $\mathbb{A}$ be a skeleton. A *critical value* for $\mathbb{A}$ is either an encrypted message $c = \{\!|t_0|\!\}_{t_1}$, or else an atom $c \in \mathsf{unique}_{\mathbb{A}}$ where c originates on some regular node $n_0 \in \mathsf{nodes}(\mathbb{A})$. Let S be a set of encrypted messages.

$\mathsf{Cut}(c, S, \mathbb{A})$, the *test cut for* c, S in $\mathbb{A}$, is the cut L, U (if one exists) such that c is a critical value and

$$U = \{n \in \mathsf{nodes}(\mathbb{A}) \colon \exists m \,.\, m \preceq_{\mathbb{A}} n \wedge c \dagger^S \mathsf{msg}(m)\}.$$

S is the *escape set* of $\mathsf{Cut}(c, S, \mathbb{A})$. □

Lemma 5. *If c is a critical value for $\mathbb{A}$, and $\exists n \in \mathsf{nodes}(\mathbb{A}) \,.\, c \dagger^S \mathsf{msg}(n)$, then:*

1. *$(L, U) = \mathsf{Cut}(c, S, \mathbb{A})$ is defined;*
2. *U has $\preceq_{\mathbb{A}}$-minimal nodes; and*
3. *If n_1 is $\preceq_{\mathbb{A}}$-minimal in U, then $c \dagger^S \mathsf{msg}(n_1)$, but for all $m \prec_{\mathbb{A}} n_1$, $c \odot^S \mathsf{msg}(m)$.*

A test cut is *solved* when there is an explanation for how c escaped from S:

Definition 6. $L, U = \mathsf{Cut}(c, S, \mathbb{A})$ is *solved* iff, for every $\preceq_{\mathbb{A}}$-minimal member n_1 of U, either:

1. n_1 is a transmission node; or
2. there is a listener node $m = \mathsf{Lsn}[t]$ with $m \prec_{\mathbb{A}} n_1$, and either
   (a) $c = \{\!|t_0|\!\}_t$, or else
   (b) for some $\{\!|t_0|\!\}_{t_1} \in S$, t is the corresponding decryption key $t = t_1^{-1}$.

The *solutions* for $\mathsf{Cut}(c, S, \mathbb{A})$ are the $\preceq_{\mathbb{A}}$-minimal transmission nodes in U together with the listeners satisfying the conditions for m. $\mathbb{A}$ is *saturated* if $\mathsf{Cut}(c, S, \mathbb{A})$ is solved whenever defined, for every c, S. □

We summarize the two main facts about saturated skeletons:

Theorem 1. *1. Every realized $\mathbb{A}$ has a realized saturated extension $\mathbb{A}'$, where:*
   (a) $\mathsf{nodes}(\mathbb{A}) \subseteq \mathsf{nodes}(\mathbb{A}')$, and every $m \in (\mathsf{nodes}(\mathbb{A}') \setminus \mathsf{nodes}(\mathbb{A}))$ is a listener node;
   (b) $\preceq_{\mathbb{A}} = \preceq_{\mathbb{A}'} \cap (\mathsf{nodes}(\mathbb{A}) \times \mathsf{nodes}(\mathbb{A}))$;
   (c) $\mathsf{unique}_{\mathbb{A}} = \mathsf{unique}_{\mathbb{A}'}$ and $\mathsf{non}_{\mathbb{A}} = \mathsf{non}_{\mathbb{A}'}$.
2. If $\mathbb{A}$ is saturated, then $\mathbb{A}$ is realized.

Clause 1 expresses the *soundness* of the authentication tests [11, Props. 2,3], and clause 2 expresses the *completeness* of the authentication tests [11, Prop. 5]. We call $\mathbb{A}'$ a *saturated extension* of $\mathbb{A}$ if it satisfies the properties given in clause 1.

A useful fact, for our present purposes, is that the escape set S may be restricted to "relevant" encryptions, where encryptions are relevant if they are ingredients in the skeleton:

Lemma 7. *Suppose S' includes all encryptions that are ingredients of messages in $\mathbb{A}$, i.e. for all encryptions e and all $n \in \mathsf{nodes}(\mathbb{A})$, $e \sqsubseteq \mathsf{msg}(n)$ implies $e \in S'$. Then:*

1. $\mathsf{Cut}(c, S, \mathbb{A}) = \mathsf{Cut}(c, S \cap S', \mathbb{A})$*, or else both are undefined, and*
2. $\mathbb{A}$ *is saturated iff all well-defined cuts of the form* $\mathsf{Cut}(c, S \cap S', \mathbb{A})$ *are solved.*

5 Multiprotocols

In our context, we are interested in an initially given protocol, the primary protocol Π_1, as well as a protocol Π which includes it. We have written Π in the form $\Pi_1 \cup \Pi_2$, but this is slightly misleading. We really want to distinguish the behaviors of Π_1 from those behaviors of Π that are definitely not behaviors of Π_1.

Definition 8. 1. (Π, Π_1) is a *multiprotocol* if Π, Π_1 are protocols, and every role of Π_1 is a role of Π.

2. A node n_j is *primary* if the strand segment $n_0 \Rightarrow \ldots \Rightarrow n_j$ leading to n_j is identical with an initial segment of some $\alpha(\rho_1)$ for $\rho_1 \in \Pi_1$. That is, the directed term $\pm t_i$ on n_i has the same direction as the i^{th} node m_i of ρ_1, and $t_i = \alpha(\mathsf{msg}(m_i))$, for each $i \leq j$. A node n_2 is *secondary* if it lies on some instance of some $\rho \in \Pi$, but it is not primary.
3. $E(\Pi_1)$ is the set of *primary encryptions*:

$$E(\Pi_1) = \{\alpha(e_1) \colon \exists n_1, \rho_1 \,.\, e_1 \sqsubseteq \mathsf{msg}(n_1) \wedge n_1 \text{ lies on } \rho_1 \wedge \rho_1 \in \Pi_1\},$$

i.e. instances $\alpha(e_1)$ of an ingredient of a node lying on a $\rho_1 \in \Pi_1$. Any non-primary encryption $e_2 \notin E(\Pi_1)$ is a *secondary encryption*. □

$E(\Pi_1) \neq \{e \colon \exists n_1, \rho_1, \alpha \,.\, e \sqsubseteq \mathsf{msg}(n_1) \wedge n_1 \text{ lies on } \alpha(\rho_1) \wedge \rho_1 \in \Pi_1\}$, since the latter will contain *all* encryptions, at least if Π_1 has any role with an indeterminate as ingredient. This is the naïve generalization of the definition used in [17], and the crucial change to make the definition useful for protocols with indeterminates (blank slots) is to use $E(\Pi_1)$ instead. $E(\Pi_1)$ contains the instances of encryptions that are *syntactically present* as ingredients in the roles of Π_1. Although changing the definition is not hard, proving that this provides a correct definition requires work.

We sometimes refer to the secondary nodes of (Π, Π_1) as a secondary protocol Π_2, but this is not really correct. For instance, Π_1 contains the listener role, so

listener nodes are primary, not secondary, and Π_2, which has no listener nodes, does not satisfy our definition of protocols.

When analyzing (Π, Π_1), we want to know whether all realized skeletons of Π satisfy the goals achieved by all realized skeletons of Π_1. Naturally, the analysis may be iterated: An enclosing protocol Π may be a subprotocol of some new Π'; thus, (Π', Π) is a multiprotocol that should not undermine the goals of Π.

Secondary encryptions never originate on primary nodes, and that is why, in the definition of protocol, we required each indeterminate to be received on a role $\rho \in \Pi$ before any other appearance:

Lemma 9. *Let e_2 be a secondary encryption, such that $e_2 \ll \mathsf{msg}(n)$, where $n = \alpha(n_1)$ for some n_1 lying on $\rho_1 \in \Pi_1$. For some negative m, $m \Rightarrow^* n$ and $e_2 \sqsubseteq \mathsf{msg}(m)$, so e_2 does not originate on n.*

Proof. Let $t_1 = \mathsf{msg}(n_1)$ and $t = \mathsf{msg}(n) = \alpha(t_1)$. By the definition of applying a homomorphism, for every path p to any encryption e in t, either $@_p(t_1)$ is also an encryption e_1, or else $p = p_1 \frown p_2$, where $@_{p_1}(t_1) = x$ is an indeterminate, and $@_{p_2}(t_1)$ leads to e within $\alpha(x)$. In the first case, $e = \alpha(e_1)$, so this case does not apply to the secondary encryption e_2. Thus, each location at which e_2 appears is of the form $p = p_1 \frown p_2$ where $@_{p_1}(t_1) = x$ is an indeterminate.

Since n_1 is positive, then there is a negative n_0 such that $n_0 \Rightarrow^+ n_1$ and $x \sqsubseteq \mathsf{msg}(n_0)$. Since:

$$e_2 \sqsubseteq \alpha(\mathsf{msg}(n_0)) \text{ and } \alpha(n_0) \Rightarrow^+ n,$$

we may take $m = \alpha(n_0)$. □

In constructing a Π_1-skeleton from a Π-skeleton that is a counterexample $\mathbb{A}$, we will use two operations:

1. Restricting $\mathbb{A}$ by discarding its secondary nodes, and
2. Removing encryptions that do not belong to Π_1, by undoing homomorphisms, so to speak.

We turn now to removals, the second of these, and will return to restrictions in the next section.

Removing eliminable encryptions. An encryption e is *ineliminable* if $e \in$

$$\{\alpha(e_1) \colon \exists e_1, n_1, \rho \,.\, e_1 \ll \mathsf{msg}(n_1) \wedge n_1 \text{ lies on } \rho_1 \in \Pi_1\},$$

that is, when e is an instance of some encryption that *appears* syntactically within any role of Π_1. An *eliminable encryption* e is an encryption that is not ineliminable. Thus, the eliminable encryptions are those that are not an instance of any encryption syntactically appearing in a primary role. Eliminable encryptions do not overlap primary encryptions, although there may also be encryptions that are neither eliminable nor primary. The ineliminable but non-primary encryptions are used as compound keys in creating primary encryptions, or perhaps are nested within these compound keys.

When eliminable encryptions appear in $\mathbb{A}\restriction\Pi_1$, we would like to remove them, in the sense of considering a preimage of $\mathbb{A}\restriction\Pi_1$ under a homomorphism, such that eliminable encryptions no longer appear in this preimage. That is, we would like to remove the eliminable encryptions by moving from $\mathbb{A}\restriction\Pi_1 = \alpha(\mathbb{A}_0)$ to $\mathbb{A}_0$, which may be properly chosen to contain no eliminable encryptions.

Suppose that a homomorphism $\alpha = [x_i \mapsto e_i]_{i\in I}$ maps the indeterminates x_i to encryptions e_i and leaves every other indeterminate y and every atom a unchanged. We regard α as a recipe for removing appearances of the e_i from t_1, obtaining some result t_0 where the following conditions hold:

1. $t_1 = \alpha(t_0)$;
2. no x_i appear in t_1, i.e. $x_i \not\ll t_1$; and
3. no e_i appears in t_0, i.e. $e_i \not\ll t_0$.

Given t_1, for every choice of $\{e_i\}_{i\in I}$, we may select new indeterminates $\{x_i\}_{i\in I}$ such that this can succeed. However, when some of the e_is may appear within others, there may be more than one result t_0 compatible with the conditions. For instance, if $e_i \ll e_j$, one may choose whether to remove an occurrence of e_j from t_1, or whether instead to remove an embedded occurrence of e_i. In either case, this may eliminate both e_i and e_j from the result.

We stipulate that the removal should always opt to excise the larger message, i.e. e_j in the example given. Let us say that p is a *distinguishing path* for t_0, t_1 if the topmost vertices of $@_p(t_0)$ and $@_p(t_1)$ do not have the same type and label, but for each proper subpath p' of p, the topmost vertices of $@_{p'}(t_0)$ and $@_{p'}(t_1)$ do have the same type and label. Let $\mathsf{Dist}(t_0, t_1)$ be the set of all distinguishing paths for t_0, t_1.

Definition 10. Let the set I be finite, let $\{x_i\}_{i\in I}$ be an I-indexed family of indeterminates, and let $\{e_i\}_{i\in I}$ be an I-indexed family of encryptions, both without repetitions. Let $\alpha = [x_i \mapsto e_i]_{i\in I}$ be the homomorphism that is the identity for all indeterminates $y \notin \{x_i\}_{i\in I}$, and for all atoms, while mapping each x_1 to e_i. Then α is a *removal for* a message t_1 if, for all $i \in I$, $x_i \not\ll t_1$.

The *result* of the removal α for t_1 is the term t_0 such that:

1. $\forall i \in I\,.\, e_i \not\ll t_0$; and
2. $\forall p \in \mathsf{Dist}(t_0, t_1)$, $\exists i \in I\,.\, @_p(t_1) = e_i$, and $@_p(t_0) = x_i$.

α is an *eliminative removal* if each e_i is eliminable. □

Algorithmically, to determine t_0 given t_1, $\{e_i\}_{i\in I}$ and $\{x_i\}_{i\in I}$, where the x_is do not appear in t_1, we walk the abstract syntax tree for t_1. We copy structure, except that as soon as we encounter any subtree equal to any e_i, we insert x_i instead.

In favorable cases, the relations $\odot$ and $\dagger$ are preserved by message removal:

Lemma 11. *Let $\alpha = [x_i \mapsto e_i]_{i\in I}$ be a removal for t_1 with result t_0:*

1. $c \odot^S t_0$ *implies* $\alpha(c) \odot^{\alpha(S)} \alpha(t_0)$, *when* $\forall i \in I$, $\alpha(c) \odot^{\alpha(S)} e_i$.
2. $c \dagger^S t_0$ *implies* $\alpha(c) \dagger^{\alpha(S)} \alpha(t_0)$, *when* $\forall i \in I, \forall t \in S$, $e_i \not\ll t$.

Proof. 1. Suppose that $p \in [\ell, r]^*$ is a path in $\alpha(t_0)$ leading to $\alpha(c)$ without traversing a key edge. By the definition of homomorphism, p is of one of two forms:
   (a) p is a path within t_0. Since $c \odot^S t_0$, p traverses some $t \in S$ within t_0. Thus, p traverses $\alpha(t)$ within $\alpha(t_0)$.
   (b) $p = p_1 \frown p_2$ where p_1 leads within t_0 to x, and p_2 leads within e to $\alpha(c)$. Then since $\alpha(c) \odot^{\alpha(S)} e$, p_2 traverses some member of $\alpha(S)$.

2. Let p be a path within t_0 leading to c without traversing either a key edge or a member of S before its end. Suppose that $p = p_1 \frown p_2$, where p_1 leads within t_0 to some value $t \notin S$; is it possible that $\alpha(t) \in \alpha(S)$? Equivalently, is there some $t' \in S$, such that $\alpha(t') = \alpha(t)$?
   If so, then t and t' differ only for paths that in one lead to x_i and in the other lead to e_i. By the definition of removal, we know that $e_i \not\ll t$. Likewise, by the assumption that e_i does not appear in any member of S, we know that $e_i \not\ll t'$. Thus, $t' = t \notin S$, contradicting the choice of t'. □

We may lift removals to strands and to Π_1-skeletons; α is a *(skeleton) removal for* $\mathbb{A}_1$ if it is a (message) removal for $\mathsf{msg}(n)$ for each node $n \in \mathsf{nodes}(\mathbb{A})$. If the e_i are eliminable encryptions, then we call this an *eliminative removal*. If $\mathbb{A}$ were a Π-skeleton but not a Π_1-skeleton, then the result of the eliminative removal α might include strands that are not instances of any role.

Lemma 12. *The result of an eliminative removal on a primary node is a primary node for the same role. I.e. let n_j be a primary node, and let α be an eliminative removal for the messages lying on $n_0 \Rightarrow \ldots \Rightarrow n_j$, an instance of some role ρ. Then the result of the removal $m_0 \Rightarrow \ldots \Rightarrow m_j$ consists of primary nodes for an instance of ρ.*

In particular, if c_1 is a primary encryption $c_1 = \alpha(e_1)$, then the result of the removal c_0 is a primary encryption of the form $c_0 = \alpha'(e_1)$.

Proof. By induction on the j-tuple of sets

$$\langle\, \mathsf{Dist}(\mathsf{msg}(m_i), \mathsf{msg}(n_i))\, \rangle_{i \leq j}$$

of minimal-length paths at which corresponding nodes m_i and n_i differ. We order these tuples pointwise, using the inclusion partial order on the sets of distinguishing paths.

Fix $m_0 \Rightarrow \ldots \Rightarrow m_j$, and assume that the lemma holds for all $n'_0 \Rightarrow \ldots \Rightarrow n'_j$ that differ from $m_0 \Rightarrow \ldots \Rightarrow m_j$ with smaller distinguishing tuple. In particular, $\langle\, \mathsf{Dist}(\mathsf{msg}(m_i), \mathsf{msg}(n_i))\, \rangle_{i \leq j} = \langle \emptyset, \ldots, \emptyset \rangle$, then $n_0 \Rightarrow \ldots \Rightarrow n_j$ is primary by assumption.

Since m_j is primary, there is a β and $\rho_1 \in \Pi_1$ such that the m_i are the successive nodes of an initial segment of $\beta(\rho_1)$. We assume that the x_i do not appear in ρ_1, perhaps because they are chosen distinct from all indeterminates appearing in all roles of Π.

If p is a path to some secondary e_2 in $\mathsf{msg}(n_i)$, then $p = p_1 \frown p_2$, where p_1 reaches some indeterminate y within $\rho_1 \downarrow i$, and p_2 reaches e_2 within $\beta(y)$.

Choosing z not appearing in the nodes $\langle n_i \rangle_i, \langle m_i \rangle_i$, or roles of Π, $[z \mapsto e_2]$ is a removal for $\beta(y)$. Let β' differ from β only in that $\beta'(y)$ is the result of the removal $[z \mapsto e_2]$ for $\beta(y)$. Then $\beta'(\rho_1 \downarrow i)$ is a primary node by definition. Moreover, $\beta'(\rho_1 \downarrow i)$ differs from m_i at a smaller set of paths than does n_i. □

Skeletons being finite structures containing messages of finite size, for every skeleton, there is a finite set of eliminable encryptions appearing anywhere in it. Hence, there are *full eliminative removals* that remove all of them. For a full eliminative removal for a skeleton $\mathbb{A}$, we take the finite set

$$\{e_i\}_{i \in I} = \{e \colon \exists n \in \mathsf{nodes}(\mathbb{A}) \,.\, e \ll \mathsf{msg}(n) \wedge e \text{ is eliminable}\}.$$

Primary cuts are those created by the behavior of the primary protocol:

Definition 13. $\mathsf{Cut}(c, S, \mathbb{A})$ is *a primary cut* if

1. $\mathsf{Cut}(c, S, \mathbb{A})$ is well-defined;
2. if c is an atom, then c originates on a primary node;
3. if c is an encryption, then $c \in E(\Pi_1)$; and
4. $S \subseteq E(\Pi_1)$. □

Lemma 14. *Let α be a full eliminative removal for $\mathbb{A}_1$, with result $\mathbb{A}_0$, and α be a removal for S_1 with result S_0. If $\mathsf{Cut}(c_0, S_0, \mathbb{A}_0) = L_0, U_0$ is a primary cut, then (1) $\mathsf{Cut}(\alpha(c_0), S_1, \mathbb{A}_1) = L_1, U_1$ is a primary cut; and $\alpha(U_0) \subseteq U_1$.*

Proof. 1. Since $\mathsf{Cut}(c_0, S_0, \mathbb{A}_0)$ is well-defined, there is a minimal $n_0 \in U_0$ where $c_0 \dagger^{S_0} \mathsf{msg}(n_0)$. Since α is a removal for S_1 with result S_0, we may apply Lemma 11, Clause 2, inferring that $\alpha(c_0) \dagger^{S_1} \alpha(\mathsf{msg}(n_0))$. Thus, $\mathsf{Cut}(\alpha(c_0), S_1, \mathbb{A}_1) = L_1, U_1$ is well-defined. The remaining conditions are immediate.

2. If $n \in U_0$, then there is a minimal $n_0 \in U_0$ with $n_0 \preceq_{\mathbb{A}_0} n$. Thus, $\alpha(n_0) \preceq_{\mathbb{A}_1} \alpha(n)$. As we have just seen, $\alpha(c_0) \dagger^{S_1} \alpha(\mathsf{msg}(n_0))$, so $\alpha(n_0) \in U_1$. Hence $\alpha(n) \in U_1$. □

6 Strong disjointness

The key notion—for ensuring that a multiprotocol does not interfere with the goals of its primary protocol—concerns how the secondary nodes transform encryptions. To create an encryption is one way to transform it, or another way is to remove some ingredient—such as a smaller encryption or a nonce or key—from inside it.

Definition 15. 1. Suppose that $e \in E(\Pi_1)$. If e originates on any secondary transmission node n_2, then n_2 is an *encryption creation conflict*.

2. Suppose that $e \in E(\Pi_1)$, and $t_1 \sqsubseteq e \sqsubseteq \mathsf{msg}(n_2)$, where n_2 is a secondary reception node. Suppose $n_2 \Rightarrow^+ n_2'$, where n_2' is a transmission node, and $t_1 \sqsubseteq \mathsf{msg}(n_2')$. Node n_2' is an *extraction conflict* if for some set $S \subseteq E(\Pi_1)$:

$$(\forall m \,.\, m \Rightarrow^+ n_2' \;\supset\; t_1 \odot^S \mathsf{msg}(m)) \quad \wedge \quad t_1 \dagger^S \mathsf{msg}(n_2').$$

3. Π, Π_1 has *strongly disjoint encryption* (s.d.e.) iff it has neither encryption creation conflicts nor extraction conflicts. □

Strong disjointness, as defined here, may not appear entirely syntactic, because its definition talks about all strands of the protocols Π and Π_1. However, it may be checked in a syntactic way using the definitions of the finitely many roles of Π and Π_1. To check whether there is an encryption creation conflict, we consider each encryption e_1 that is an ingredient in any node of a role $\rho_1 \in \Pi_1$. Suppose that e_1 unifies with some encryption e_2 that is an ingredient in a transmission node m', with most general unifier α, where $\alpha(\mathsf{msg}(m'))$ is a secondary node. Then there should be a node m such that

$$m \Rightarrow^+ m' \text{ and } \alpha(e_2) \sqsubseteq (\alpha(\mathsf{msg}(m))).$$

If this is always the case, then there are no encryption creation conflicts. The use of unification to check extraction conflicts is more elaborate but similar.

Our main theorems—Thm. 2 here and Thm. 3 in Section 7—show that s.d.e. suffices to be able to extract a realized Π_1-skeleton from a realized Π-skeleton, and that these Π_1-skeletons witness for the fact that Π does not undermine security goals achieved by Π_1.

Lemma 16. *Let Π, Π_1 have s.d.e., with $\mathbb{A}$ a Π-skeleton, and let $\mathsf{Cut}(c, S, \mathbb{A}) = L, U$ be primary. Solutions for $\mathsf{Cut}(c, S, \mathbb{A})$ are primary nodes.*

Proof. A solution is either a listener node m—which is primary—or else a $\preceq_{\mathbb{A}}$-minimal transmission node $m_1 \in U$. So suppose m_1 is secondary.

If c is an atom, then since it originates uniquely, and on a primary node, there are nodes $m_0 \Rightarrow^+ m_1$, where $c \sqsubseteq \mathsf{msg}(m_0)$. By Lemma 5, c is found only within S in all the nodes preceding m_1 on the same strand, but $c \dagger^S \mathsf{msg}(m_1)$, so m_1 is an extraction conflict.

If c is an encryption, then it is a primary encryption by the definition of a primary cut. If c originates on m_1, then m_1 is an encryption creation conflict. Thus, there are nodes $m_0 \Rightarrow^+ m_1$, where $c \sqsubseteq \mathsf{msg}(m_0)$. By Lemma 5, c is found only within S in all the nodes preceding m_1 on the same strand, but $c \dagger^S \mathsf{msg}(m_1)$, so m_1 is an extraction conflict. □

Lemma 17. *Let Π, Π_1 have s.d.e., with $\mathbb{A}$ a saturated Π-skeleton. Suppose $\mathsf{Cut}(c, S, \mathbb{A}) = L, U$ is primary. If for some $m \in L$ and secondary encryption e_2, $e_2 \sqsubseteq \mathsf{msg}(m)$, then either $c \odot^S e_2$ or else $\mathsf{Cut}(c, S, \mathbb{A}) = L, U$ has a listener solution.*

Proof. If $c \not\sqsubseteq e_2$, the conclusion is immediate, so assume that $c \sqsubseteq e_2$. Suppose that $c \dagger^S e_2$.

First, suppose that c is an atom. Then by the definition of primary cut, c originates on a primary node n_0. However, by Lemma 9, e_2 does not originate on n_0. Also, e_2 is not an ingredient in the message of any earlier node, since it contains c. Thus, e_2 is not an ingredient in $\mathsf{msg}(n_0)$. Hence, letting $S' = \{e \in S \colon e_2 \not\sqsubseteq e\}$, in fact $c \odot^{S'} \mathsf{msg}(n_0)$.

Since $m \in U'$, where $\mathsf{Cut}(c, S', \mathbb{A}) = L', U'$, there exists a $\preceq_{\mathbb{A}}$-minimal $n_1 \preceq_{\mathbb{A}} m$ in U'. If n_1 has a listener solution, then because $S' \subset S$, so does $\mathsf{Cut}(c, S, \mathbb{A}) = L, U$.

So suppose that n_1 is a transmission node. By Lemma 16, n_1 is a primary node. In particular, by Lemma 9, e_2 does not originate on n_1. Moreover, since $c \dagger^{S'} e_2$, e_2 is not an ingredient in any earlier node on the strand of n_1. Therefore, $e_2 \not\sqsubseteq \mathsf{msg}(n_1)$. Hence, in fact, $c \dagger^{S} \mathsf{msg}(n_1)$. contradicting the assumption that $n_1 \preceq_{\mathbb{A}} m \in L$.

Second, suppose that c is an encryption. If the earliest $n_0 \preceq_{\mathbb{A}} m$ such that $c \sqsubseteq \mathsf{msg}(n_0)$ is a reception node, then since $\mathbb{A}$ is saturated, $\mathsf{Cut}(c, \emptyset, \mathbb{A})$ is solved, and there is a listener solution preceding n_0. This is also a listener solution for $\mathsf{Cut}(c, S, \mathbb{A})$. Alternatively, suppose that n_0 is a transmission node on which c originates. From this point, the argument follows the same form as when c is an atom. □

Restricting to protocol Π_1. When a Π-skeleton $\mathbb{A}$ is a counterexample to a goal of Π_1, we are interested in the subskeleton of $\mathbb{A}$ which omits all secondary nodes.

Definition 18. If $\mathbb{A}$ is a Π-skeleton, then its Π_1 restriction, $\mathbb{A} \restriction \Pi_1$, is the subskeleton $\mathbb{A}_1$ of $\mathbb{A}$ such that:

1. $m \in \mathsf{nodes}(\mathbb{A}_1)$ iff $m \in \mathsf{nodes}(\mathbb{A})$ and m is primary.
2. $m \preceq_{\mathbb{A}_1} n$ iff $m \preceq_{\mathbb{A}} n$ and $m, n \in \mathsf{nodes}(\mathbb{A}_1)$.
3. $\mathsf{unique}_{\mathbb{A}_1} = \mathsf{unique}_{\mathbb{A}}$ and $\mathsf{non}_{\mathbb{A}_1} = \mathsf{non}_{\mathbb{A}}$. □

Definition 8 declares a node primary if the strand segment leading to it is identical with an initial segment of an instance of a role $\rho_1 \in \Pi_1$; it does not matter what "would come later."

Theorem 2. *Let Π, Π_1 have strong disjoint encryption, and let $\mathbb{A}$ be a realized Π-skeleton. If $\mathbb{A}_0$ is the result of a full eliminative removal α applied to $\mathbb{A} \restriction \Pi_1$, then $\mathbb{A}_0$ is a realized Π_1-skeleton.*

Proof. We first prove that the theorem holds in case $\mathbb{A}$ is *saturated*. By Thm 1, it suffices to show that $\mathbb{A}_0$ is also saturated. Let $\mathbb{A}_1 = \mathbb{A} \restriction \Pi_1$. Since $\mathbb{A}_1$ is a Π_1-skeleton, the result $\mathbb{A}_0$ of the removal α is a Π_1-skeleton.

Supposing $\mathsf{Cut}(c_0, S_0, \mathbb{A}_0) = L_0, U_0$ is well-defined, we would like to show that it is solved. Since in $\mathbb{A}_0$ no secondary encryptions whatever appear, by Lemma 7, we may assume that S_0 is chosen to be primary. Since $c_0 \dagger^{S_0} n_1$ for some $n_1 \in \mathsf{nodes}(\mathbb{A}_0)$, $c_0 \sqsubseteq \mathsf{msg}(n_1)$. Hence if c_0 is an encryption, it is a primary encryption. So $\mathsf{Cut}(c_0, S_0, \mathbb{A}_0)$ is a primary cut.

Let $\alpha(c_0) = c$, $\alpha(S_0) = S$. $\mathsf{Cut}(c, S, \mathbb{A}_1) = L, U$ is well defined by Lemmas 5, 11. The other clauses for a primary cut are clearly satisfied. Thus, $\mathsf{Cut}(c, S, \mathbb{A}_1)$ is primary.

By the definition, $\mathsf{Cut}(c, S, \mathbb{A})$ is also a primary cut, and since by assumption $\mathbb{A}$ is realized, Thm. 1 implies it is solved. Moreover, Lemma 16 implies

that all of its solutions lie on primary nodes, so $\mathsf{Cut}(c, S, \mathbb{A}_1)$ is also solved; let $\mathsf{Cut}(c, S, \mathbb{A}_1) = L_1, U_1$.

Let $n_0 \in U_0$ be $\preceq_{\mathbb{A}_0}$-minimal. By Lemma 14, $\alpha(n_0) \in U_1$, although it may not be minimal. However, let $n_1 \preceq_{\mathbb{A}_1} \alpha(n_0)$ be $\preceq_{\mathbb{A}_1}$-minimal in U_1. If n_1 is solved with a listener node m, then its result under the removal solves $n_0 \in U_0$.

So suppose that $n_1 \in U_1$ is minimal, and solved only by a primary transmission node m_1. The node m_1 lies on some strand $\bullet \Rightarrow^* m_1$, and the result of applying the removal α is some strand $\bullet \Rightarrow^* m_0$ in $\mathbb{A}_0$. Is the latter a solution for $\mathsf{Cut}(c_0, S_0, \mathbb{A}_0) = L_0, U_0$?

First, observe that for any node $\bullet$ prior to m_0, c_0 is found only within S_0. Otherwise, Lemma 11 Clause 2 would imply that c appears outside S prior to m_1, so that it would not be a solution. By Lemma 11 Clause 1, we may infer that c_0 is found outside S_0 in $\mathsf{msg}(m_1)$, unless $c \odot^S e_i$ for one of the removed, secondary e_i. However, in this case, by Lemma 17, $\mathsf{Cut}(c, S, \mathbb{A})$ has a listener solution, returning us to an earlier case.

Finally, suppose that $\mathbb{A}$ is realized. Then, by Thm 1, $\mathbb{A}$ has a realized saturated extension $\mathbb{A}'$ that differs only in what listener strands it contains. Apply the preceding to obtain a realized Π_1 skeleton $\mathbb{A}'_0$. The desired $\mathbb{A}_0$ is the subskeleton of $\mathbb{A}'_0$ that discards listener nodes from $\mathbb{A}'_0$ when they are in $\mathbb{A}'$ but not in $\mathbb{A}$. □

Our last job is to show that $\mathbb{A}_1$ is a Π_1-counterexample to any Π_1-security goal to which $\mathbb{A}$ is a counterexample. But for that, we need a definite notion of what a security goal is.

7 A language $\mathcal{L}(\Pi)$ to describe skeletons of Π

Given a protocol Π, we define a first order vocabulary for talking about the skeletons over Π.

Definition 19. $\mathcal{L}(\Pi)$ consists of:

Variables Var ranging over nonces (e.g. N, N_a), atomic keys (e.g. K), atoms (e.g. a), messages (e.g. x, y), as well as variables ranging over nodes (e.g. m, n, n_1). We treat the sorts in the manner of order-sorted algebra [15];

Predicate symbols equality $u = v$, falsehood false (no arguments), and:

- $\mathsf{Non}(v)$, $\mathsf{Unq}(v)$, and $\mathsf{UnqAt}(n, v)$, meaning respectively that v is assumed non-compromised, that v is assumed uniquely originating, and that v is assumed uniquely originating and moreover originates at node n;
- $\mathsf{Pos}(n)$ and $\mathsf{Neg}(n)$, meaning respectively that node n is positive, i.e. a transmission node, or negative, i.e. a reception node;
- $n_0 \Rightarrow n_1$ and $n_0 \prec n_1$, meaning respectively that n_0 immediately precedes n_1 on the same strand, and that n_0 precedes n_1 in the partial ordering $\prec_{\mathbb{A}}$;

– One role predicate for each node on each role in Π. For each role of length k, there are k predicates. The arguments to the i^{th} role predicate consist of one node variable m, together with one variable for each parameter to the role that has appeared in any of the first i messages. □

Thus, every protocol has a role predicate $\mathsf{Lsn1}(m, x)$, meaning that m is the first node on a listener strand, on which message x has been received. In the case of the AIP TPM role, there are four role predicates, of which the first two are:

– $\mathsf{aip\_tpm1}(m, x)$, meaning that m is a reception node not preceded by any other on its strand, and the message received is on node m is just the parameter x, as dictated by the definition of the AIP TPM role;
– $\mathsf{aip\_tpm2}(m, x, i, \mathsf{ks}, f, e, \mathsf{ke})$, meaning that m lies on the second position on its strand after a node m' such that $\mathsf{aip\_tpm1}(m', x)$, and m transmits message:

$$\{\!|i \hat{} k \hat{} x \hat{} [\![\, \mathsf{ekctag}\ f \hat{} e \,]\!]_{\mathsf{ks}}|\!\}_{\mathsf{ke}}.$$

These predicates are certainly not independent, given the valid formula:

$$\mathsf{aip\_tpm2}(m_2, x, i, k, f, e, p) \supset \exists m_1 \,.\, m_1 \Rightarrow m_2 \wedge \mathsf{aip\_tpm1}(m_1, x).$$

If Π_1 is a subprotocol of Π in the sense that every role of Π_1 is a role of Π, then $\mathcal{L}(\Pi_1)$ is a sublanguage of $\mathcal{L}(\Pi)$.

Two ingredients are conspicuously missing from $\mathcal{L}(\Pi)$. Normally, one would expect this sort of language to have function symbols for the message constructors, encryption and concatenation. One would also expect a function which, given a node, would return the message sent or received on that node. We have omitted them for two reasons. First, we would like to make $\mathcal{L}(\Pi)$ as insensitive to the notational specifics of the protocol Π as possible, so that it describes the goals of the protocol without excessive reliance on choices about the syntax of messages. Second, we want to focus the security goals on the roles and their parameters. Assertions about compound messages embedded within parameters would provide artificial ways to construct counterexamples to our protocol independence theorem.

However, if we wanted to reason axiomatically about protocols, we would need to work within an expanded language with message constructors and a function to extract the message sent or received on a node. Goals would continue to be expressed in the sublanguage $\mathcal{L}(\Pi_1)$.

Semantics. The semantics for $\mathcal{L}(\Pi_1)$ are a straightforward classical semantics, given the requirement that each structure form a skeleton for the protocol Π. This requirement builds the permissible behaviors of Π directly into the semantics without requiring an explicit axiomatization.

Definition 20. Let $\mathcal{L}(\Pi)$ be the language for Π, and $\mathbb{A}$ be a skeleton for Π.

An *assignment* σ for $\mathbb{A}$ is a function from variables of $\mathcal{L}(\Pi_1)$ to messages, as well as to nodes. Two assignments σ, σ' are *similar up to* v, written $\sigma \sim_v \sigma'$, if for every variable v', different from v, $\sigma(v') = \sigma'(v')$.

Satisfaction $\mathbb{A}, \sigma \models \Phi$ is defined inductively:

$\mathbb{A}, \sigma \models \Phi \wedge \Psi$ iff	$\mathbb{A}, \sigma \models \Phi$ and $\mathbb{A}, \sigma \models \Psi$;
$\mathbb{A}, \sigma \models \Phi \vee \Psi$ iff	$\mathbb{A}, \sigma \models \Phi$ or $\mathbb{A}, \sigma \models \Psi$;
$\mathbb{A}, \sigma \models \neg\Phi$ iff	$\mathbb{A}, \sigma \not\models \Phi$;
$\mathbb{A}, \sigma \models \forall v\,.\,\Phi$ iff	for every σ', if $\sigma \sim_v \sigma'$, then $\mathbb{A}, \sigma' \models \Phi$;
$\mathbb{A}, \sigma \models \exists v\,.\,\Phi$ iff	for some σ', $\sigma \sim_v \sigma'$ and $\mathbb{A}, \sigma' \models \Phi$;
$\mathbb{A}, \sigma \models u = v$ iff	$\sigma(u) = \sigma(v)$;
$\mathbb{A}, \sigma \models \mathsf{Non}(v)$ iff	$\sigma(v) \in \mathsf{non}_{\mathbb{A}}$;
$\mathbb{A}, \sigma \models \mathsf{Unq}(v)$ iff	$\sigma(v) \in \mathsf{unique}_{\mathbb{A}}$;
$\mathbb{A}, \sigma \models \mathsf{UnqAt}(m, v)$ iff	$\sigma(m) \in \mathsf{nodes}(\mathbb{A})$, and $\sigma(v) \in \mathsf{unique}_{\mathbb{A}}$, and $\sigma(v)$ originates at node $\sigma(m)$;
$\mathbb{A}, \sigma \models \mathsf{Pos}(m)$ iff	$\sigma(m) \in \mathsf{nodes}(\mathbb{A})$, and $\sigma(m)$ is a transmission node;
$\mathbb{A}, \sigma \models \mathsf{Neg}(m)$ iff	$\sigma(m) \in \mathsf{nodes}(\mathbb{A})$, and $\sigma(m)$ is a reception node;
$\mathbb{A}, \sigma \models m \Rightarrow n$ iff	$m, n \in \mathsf{nodes}(\mathbb{A})$, and $\sigma(m) \Rightarrow \sigma(n)$;
$\mathbb{A}, \sigma \models m \prec n$ iff	$\sigma(m) \prec_{\mathbb{A}} \sigma(n)$;

$\mathbb{A}, \sigma \models \mathsf{RoleRhoJ}(m, v_1, \ldots, v_k)$ iff

> $\sigma(m) \in \mathsf{nodes}(\mathbb{A})$, and $\sigma(m)$ is the j^{th} node on its strand, and the successive transmissions and receptions on nodes $n \Rightarrow^* m$ are identical with the respective transmissions and receptions on the first j nodes of role ρ, instantiated with the values $\sigma(v_1), \ldots, \sigma(v_k)$.

We write $\mathbb{A} \models \Phi$ when $\mathbb{A}, \sigma \models \Phi$ for all σ. □

When n is a variable over nodes, although $\sigma(n) \notin \mathsf{nodes}(\mathbb{A})$ is permitted, in that case, whenever $\phi(n)$ is an atomic formula, $\mathbb{A}, \sigma \not\models \phi(n)$.

In the satisfaction definition, the only noteworthy clause is the last, which stipulates that only the nodes preceding $\sigma(m)$ on its strand are relevant to whether it satisfies $\mathsf{RoleRhoJ}(m, v_1, \ldots, v_k)$. In protocols where there are two different roles ρ, ρ' that differ only after their first j nodes—typically, because they represent different choices at a branch point after the j^{th} node [16, 14]—the two predicates $\mathsf{RoleRhoJ}$ and $\mathsf{RoleRho'J}$ are equivalent.

When one has a notion of homomorphism for a class of structures, one would expect (unnegated) atomic formulas to be preserved under homomorphisms. Moreover, the homomorphisms we use in this paper are bijections between the nodes of $\mathbb{A}$ and those of $\alpha(\mathbb{A})$, and they do not enrich the ordering on nodes.

Lemma 21. *1. If $\mathbb{A}$, $\sigma \models \phi$ and ϕ is an atomic formula, then $\alpha(\mathbb{A})$, $\alpha \circ \sigma \models \phi$.*

2. If $\alpha(\mathbb{A})$, $\alpha \circ \sigma \models \phi$, and ϕ is of any of the forms:
$m = n$, $\mathsf{Pos}(n)$, $\mathsf{Neg}(n)$, $m \prec n$, *and* $m \Rightarrow n$,
with m, n variables over nodes, then $\mathbb{A}$, $\sigma \models \phi$.

3. If α is injective, and $\alpha(\mathbb{A})$, $\alpha \circ \sigma \models \phi$, and ϕ is of any of the forms:
$x = y$, $\mathsf{Non}(v)$, $\mathsf{Unq}(v)$, *and* $\mathsf{UnqAt}(m, v)$,
then $\mathbb{A}$, $\sigma \models \phi$.

4. If α is an eliminative removal, and $\alpha(\mathbb{A})$, $\alpha \circ \sigma \models \phi$, and ϕ is a role predicate, then $\mathbb{A}$, $\sigma \models \phi$.

Proof. 1. By cases on the definition. 2. By the injectiveness of the skeleton homomorphism α on nodes. 3. By injectiveness and the definition of skeleton homomorphisms α. 4. By Lemma 12. □

What is a security goal? A security goal is either an authentication or a confidentiality property. An authentication goal requires a peer to have executed some behavior. If the authentication goal is a goal of Π_1 this behavior should be a strand belonging to Π_1. The peer's behavior may have expected values for all parameters, or else for just some of the parameters, in which case the remaining parameters may freely take any value. A realized $\mathbb{A}$ is a counterexample when $\mathbb{A}$ does not contain any such behavior.

A confidentiality goal requires some desired secret t not to be shared as a parameter of another strand. In the usual case, this other strand is a listener strand $\mathsf{Lsn}[t]$, in which case the goal ensures that t can never be transmitted in plaintext, without being protected by encryption. A realized $\mathbb{A}$ is a counterexample to this if it contains a listener strand $\mathsf{Lsn}[t]$ of the form $\stackrel{t}{\rightarrow} \bullet$. More generally, a realized $\mathbb{A}$ is a counterexample to a confidentiality goal if it contains a strand of the prohibited kind, with the given parameter t. We consider here only "full disclosure" goals, rather than "partial information" goals, in which the adversary learns that the regular behavior is compatible with some values of t, but not others. Much is known about the relation between full disclosure goals and partial information goals, e.g. [4, 8].

A security goal also has assumptions. These assumptions may concern unique origination of some values. They may concern long-term keys that are uncompromised, and thus non-originating. Finally, the assumptions may concern which roles have been instantiated, e.g. asserting that something must be true if a responder strand has occurred. Alternatively, we may assume that part of a role has been instantiated, e.g. asserting that something must be true if the first two nodes on a responder strand have occurred. The languages $\mathcal{L}(\Pi)$ have been designed to express these properties.

Definition 22. 1. A *security hypothesis* is an atomic formula of $\mathcal{L}(\Pi)$.

2. An *authentication conclusion* is a conjunction Ψ of atomic formulas of $\mathcal{L}(\Pi)$.
3. Suppose that G_0 is $\Phi \supset \exists v_0 \ldots v_j \,.\, (\Psi_1 \vee \ldots \vee \Psi_k)$, where Φ is a conjunction of security hypotheses and each Ψ_i is an authentication conclusion. If $k = 0$, this is the empty disjunction false. Suppose that, for every variable n over nodes occurring free in G_0, one of the hypotheses in Φ is a role predicate $\mathsf{RoleRhoJ}(n, u, \ldots, w)$.
   Then G, the universal closure of G_0, is a *security goal* of Π. G is an authentication goal if $k > 0$ and a confidentiality goal if $k = 0$.
4. A *counterexample* to G is a realized Π-skeleton $\mathbb{A}$ such that $\mathbb{A} \not\models G$. Π *enforces* G if G has no counterexamples. □

For a protocol to enforce goal G, we require only that $\mathbb{A} \models G$ hold for *realized* $\mathbb{A}$. Nevertheless, satisfaction is defined for all skeletons $\mathbb{A}$, and this eases stating the invariants we need in proving Thm. 3.

Main result. Our main theorem now depends only a further lemma, which says that being a counterexample to a Π_1-security goal is preserved under our two operations. This lemma does not assume Π, Π_1 strongly disjoint.

Lemma 23. *Let Π, Π_1 be a multiprotocol, with $G_1 \in \mathcal{L}(\Pi_1)$ a security goal for Π_1. Let $\mathbb{B}$ be a Π-skeleton, with $\mathbb{B} \models \neg G_1$.*

1. *$\mathbb{B} \restriction \Pi_1 \models \neg G_1$;*
2. *If $\mathbb{A}$ results from $\mathbb{B}$ by an eliminative removal γ, then $\mathbb{A} \models \neg G_1$.*

Proof. Let $G_1 = \forall v_0, \ldots, v_j \,.\, \Phi_0 \wedge \ldots \wedge \Phi_k \supset \exists u_0, \ldots, u_\ell (\Psi_0 \vee \ldots \vee \Psi_i)$. Let $\mathbb{B}, \sigma \models \Phi_0 \wedge \ldots \wedge \Phi_k$, where there is no τ differing from σ only on $u_0, \ldots, u_\ell$ such that $\mathbb{B}, \tau \models \Psi_0 \vee \ldots \vee \Psi_i$. So, for every such τ,

$$\mathbb{B}, \tau \models \Phi_0 \wedge \ldots \wedge \Phi_k \wedge \neg\Psi_0 \wedge \ldots \wedge \neg\Psi_i.$$

We must show that all of the Φ^s remain true, while all of the Ψ^s remain false.

1. In this case, any atomic formula that does not mention a variable over nodes will have its truth value unchanged.

Suppose that one of the Φ^s is a role predicate $\mathsf{RoleRhoJ}(n, u, \ldots, w)$. Since G_1 is a security goal of $\mathcal{L}(\Pi_1)$, $\mathsf{RoleRhoJ}$ concerns a primary role, and since the atomic formula is satisfied in $\mathbb{B}$, $\sigma(n)$ is a primary node. Therefore, $\sigma(n) \in \mathsf{nodes}(\mathbb{B} \restriction \Pi_1)$, as are the predecessors of n on the same strand. Thus, $\mathbb{B} \restriction \Pi_1 \models \mathsf{RoleRhoJ}(n, u, \ldots, w)$.

If Φ is another atomic formula that contains a node variable n, then by the definition of security goal, there is a Φ' in which n is also the node variable. Thus, as we have just observed, it is a primary node $\sigma(n) \in \mathsf{nodes}(\mathbb{B} \restriction \Pi_1)$. From the definitions of restriction and satisfaction, we can see that all of the formulas $m = n$, $\mathsf{UnqAt}(n, v)$, $\mathsf{Pos}(n)$, $\mathsf{Neg}(n)$, $m \prec n$, and $m \Rightarrow n$ are preserved under restriction when the relevant nodes belong to the restricted skeleton.

Consider now a (false) security conclusion Ψ. Since Ψ is a conjunction, for some conjunct ψ of Ψ, $\mathbb{B}, \tau \models \neg\psi$. If ψ does not mention a node variable, its falsehood is certainly preserved. Suppose it mentions a node variable n, and suppose first that $\sigma(n) \in \mathsf{nodes}(\mathbb{B} \restriction \Pi_1)$ is primary. If $\mathbb{B} \restriction \Pi_1, \tau \models \psi$, then this fact would be preserved under the embedding of $\mathbb{B} \restriction \Pi_1$ into $\mathbb{B}$ (Lemma 21, Clause 1). If instead $\sigma(n)$ is secondary, then $\sigma(n) \notin \mathsf{nodes}(\mathbb{B} \restriction \Pi_1)$, so that we certainly have $\mathbb{B} \restriction \Pi_1, \tau \models \neg\psi$. An atomic formula is true only if the nodes it mentions are present in the skeleton.

2. Let τ' be the result of the eliminative removal γ to τ; i.e. $\tau'(x)$ is the result of the removal γ on $\tau(x)$, for each variable x.

If any of the Ψ^s were satisfied by some $\mathbb{A}, \tau'$, then Lemma 21 Clause 1 would imply that it is satisfied by $\mathbb{B}, \tau$. For the Φ^s, we use the remaining clauses of Lemma 21. □

Theorem 3. *Let Π, Π_1 have strongly disjoint encryption, and let $G_1 \in \mathcal{L}(\Pi_1)$ be a security goal. If for some realized Π-skeleton, $\mathbb{A} \models \neg G_1$, then for some realized Π_1-skeleton $\mathbb{A}_1$, $\mathbb{A}_1 \models \neg G_1$.*

8 Conclusion

In this paper, we have established a result about composition of protocols, using a new, model-theoretic approach. It is model-theoretic because it combines reasoning about the logical form of formulas—in Section 7, about the security goals G—with operations on the structures that furnish models of these formulas. In our case, these structures are the skeletons and especially realized skeletons, and the operations on them are homomorphisms and removals, as used in Sections 5 and 6. The authentication tests are crucial, because they give us a criterion for when a skeleton is realized, and this criterion suggests the definition of strong disjointness (Def. 15).

Thm. 3 is useful, apart from any interest of the model-theoretic method used to establish it. First, it simplifies establishing that two protocols combine to provide a protocol achieving some desired goals. If strong disjointness holds, any goals of the joint protocol that are only about the primary protocol may be verified without reference to the secondary protocol. Goals that involve the secondary protocol must be verified using the whole joint protocol, unless of course the relation is symmetric. If the joint protocol can be regarded as a strongly disjoint multiprotocol in both of two ways, with each of the two subprotocols as primary protocol, then goals about each one may be verified without reference to the other. This is the Cortier-Delaitre-Delaune case [9]. Unfortunately, their proof method appears not to extend to the asymmetric case.

There is a second reason why our composition result is useful. It can also be read as a prescription—or at least a heuristic—for protocol design. Protocols can be built from subprotocols that provide some of the intermediate cryptographic values that they require. Our addition of protocols for a payment process on top of the TCG authenticated identity protocol is an example of this process. In this context, Thm. 3 gives the constraints that a protocol designer must adhere to, in enriching an existing framework of protocols. His new operations must be strongly disjoint from the existing protocols, regarded as a primary protocol.

Related work. Probably the outstanding work on protocol derivation and composition is a group of articles by Datta, Derek, Mitchell, and Pavlovic; for our purposes the central member of the group is [10]. In this paper the authors explore a variety of protocols that have common ingredients, and show how they can be viewed within a sort of family tree, related by a number of operations on protocols.

Our notion of composition corresponds directly to their parallel composition, but their sequential composition also fits within our context without any real additional work. They do have other operations: Refinement is an operation that enriches the message structure of a protocol to obtain another protocol. Transformation is an operation that moves information from one message in a protocol to a different message, either to reduce the number of messages or to provide a tighter binding among the protocol parameters.

Although they have defined a very rich palette of operations, their main results about them are restricted to parallel and sequential composition. [10,

Thm. 4.4] concerns parallel composition and [10, Thm. 4.8] concerns sequential composition. In each case, the theorem applies to a particular proof of a particular security goal G_1 about a protocol Π_1. This proof relies on some set Γ of invariant formulas that Π_1 preserves. Thm. 4.4 states that if a secondary protocol Π_2 respects Γ, then G_1 holds of the parallel composition $\Pi_1 \cup \Pi_2$. Thm 4.8 is a more elaborate but comparable result for sequential composition.

By contrast, our result is more uniform. It makes an assertion about all security goals, rather than a single proof of a single goal. The strong disjointness property ensures that a secondary protocol will respect all usable invariants of the primary protocol. Thus, one need not check through the details of many proofs to find what invariants to re-establish. A syntactic property, checked once, suffices permanently.

Universal composability [7] appears to be a related property, although expressed in a very different underlying model. Universal composability is typically realized by randomly choosing a tag to be inserted in all the messages of a protocol, this tag being chosen at implementation time. Thus, the symmetric disjointness of any two protocols is ensured with overwhelming probability. An additional consequence of the model is that random values generated within the protocol must not be used as keys in the protocol run.

Andova, Cremers, et al. [1] also develop an approach to protocol composition which supports sequential as well as parallel composition. They suggest use of tags or of distinct sets of keys as implementation strategies, as in our [17], and independently propose a definition [1, Def. 25] akin to the symmetric definition of [9].

Future work. A question with strong interest is whether there is a theorem akin to our Thm. 3 which applies to the refinement and transformation operations [10]. We believe that at least for a specific, limited definition of *refinement*, the answer should be affirmative. Such a result would be extremely useful for guiding protocol design.

Acknowledgments. I am grateful to Javier Thayer for conversations. A personal communication of Stephanie Delaune explained that extending [9] to the asymmetric case appeared infeasible.

References

1. S. Andova, C.J.F. Cremers, K. Gjøsteen, S. Mauw, S.F. Mjølsnes, and S. Radomirović. Sufficient conditions for composing security protocols. *Information and Computation*, 2007.
2. Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The avispa tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

3. Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the Direct Anonymous Attestation protocol. In *IEEE Symposium on Security and Privacy*, 2008.
4. Michael Backes and Birgit Pfitzmann. Relating cryptographic and symbolic key secrecy. In *Proceedings, 26th IEEE Symposium on Security and Privacy*, May 2005. Extended version, `http://eprint.iacr.org/2004/300`.
5. Boris Balacheff, Liqun Chen, Siani Pearson, David Plaquin, and Graeme Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, 2003.
6. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *ACM Conference on Communications and Computer Security (CCS)*, 2004. Full version available at `http://eprint.iacr.org/2004/205`.
7. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Report 2000/067, Internation Association for Crryptographic Research, October 2001. Extended Abstract appeared in proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001.
8. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proceedings, Theory of Cryptography Conference (TCC)*, March 2006.
9. Véronique Cortier, Jérémie Delaitre, and Stéphanie Delaune. Safely composing security protocols. In V. Arvind and Sanjiva Prasad, editors, *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, Lecture Notes in Computer Science, New Delhi, India, December 2007. Springer.
10. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.
11. Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Completeness of the authentication tests. In J. Biskup and J. Lopez, editors, *European Symposium on Research in Computer Security (ESORICS)*, number 4734 in LNCS, pages 106–121. Springer-Verlag, September 2007.
12. Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 523–538. Springer, March 2007. Extended version at URL:`http://eprint.iacr.org/2006/435`.
13. Daniel Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
14. Sibylle Fröschle. Adding branching to the strand space model. In *Proceedings of EXPRESS'08*, Electronic Notes in Theoretical Computer Science. Elsevier, 2008. To appear.
15. Joseph A. Goguen and José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
16. Joshua D. Guttman, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Programming cryptographic protocols. In Rocco De Nicola and Davide Sangiorgi, editors, *Trust in Global Computing*, number 3705 in LNCS, pages 116–145. Springer, 2005.
17. Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings, 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, July 2000.

18. Gavin Lowe. Casper: A compiler for the analysis of security protocols. In *10th Computer Security Foundations Workshop Proceedings*, pages 18–30. IEEE Computer Society Press, 1997.
19. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 1998. Also Report 443, Cambridge University Computer Lab.