Analysing Layered Security Protocols

Thomas Gibson-Robinson
St Catherine's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Trinity 2013

Abstract

Many security protocols are built as the composition of an *application-layer* protocol and a *secure transport protocol*, such as TLS. There are many approaches to proving the correctness of such protocols. One popular approach is *verification by abstraction*, in which the correctness of the application-layer protocol is proven under the assumption that the transport layer satisfies certain properties, such as confidentiality. Following this approach, we adapt the strand spaces model in order to analyse application-layer protocols that depend on an underlying secure transport layer, including *unilaterally authenticating secure transport protocols*, such as unilateral TLS. Further, we develop proof rules that enable us to prove the correctness of application-layer protocols that use either unilateral or bilateral secure transport protocols. We then illustrate these rules by proving the correctness of WebAuth, a single-sign-on protocol that makes extensive use of unilateral TLS.

In this thesis we also present a full proof of the model's soundness. In particular, we prove that, subject to a suitable independence assumption, if there is an attack against the application-layer protocol when layered on top of a particular secure transport protocol, then there is an attack against the abstracted model of the application-layer protocol. In contrast to existing work in this area, the independence assumption consists of eight statically-checkable conditions, meaning that it can be checked statically, rather than having to consider all possible runs of the protocol.

Lastly, we extend the model to allow protocols that consist of an arbitrary number of layers to be proven correct. In this case, we prove the correctness of the intermediate layers using the high-level strand spaces model, by abstracting away from the underlying transport-layers. Further, we extend the above soundness results in order to prove that the multi-layer approach is sound. We illustrate the effectiveness of our technique by proving the correctness of a couple of simple multi-layer protocols.

Acknowledgements

I cannot express sufficient thanks to my supervisor, Gavin Lowe, for helping me immeasurably during my thesis. Not only has he been a wonderful collaborator to do research with, always providing useful ideas, checking proofs and proof-reading, but he has provided me with so many opportunities throughout my degree. I am truly grateful to Gavin for this.

I am very grateful to Sadie Creese, Michael Goldsmith and Bill Roscoe for all providing useful feedback on my work, and providing me with many interesting opportunities in other research areas.

I would also like to thank my family and friends for providing me with much needed distractions during my DPhil. I'm also very grateful to Sze-Kie for her support throughout my doctorate, and for putting up with me when I have been stuck on a proof.

Last, but certainly not least, I would like to thank all of my friends in the department for providing me not just with countless stimulating discussions in so many different areas, but also for being such excellent sources of laughter, entertainment and impromptu pub visits!

Contents

Chapter 1

Introduction

A *security protocol* is a sequence of messages that are exchanged between a number of agents in order to allow the agents to conclude that particular *security goals* have been achieved. For example, a security protocol might allow a user to securely login to a server, or to transfer money securely between accounts.

Modern security protocols tend to be constructed as two separate layers. The bottom layer, known as the *secure transport layer*, is a generic secure transport protocol, such as TLS [DR08], that can send arbitrary messages protected in some way. For example, the secure transport layer might guarantee confidentiality of the messages (i.e. the messages can only be read by the intended recipient) or authenticity (i.e. the recipient can be sure of the identity of the sender). The second layer, known as the *application layer*, uses the guarantees provided by the transport layer to provide some useful functionality. For example, WebAuth [SA09] is an application-layer protocol that uses the security guarantees provided by the transport layer to provide a single-sign-on service, so that users can login to multiple websites with just one account.

It is highly desirable to be able to prove the correctness of security protocols, given that the correctness of even trivial protocols is often difficult to precisely reason about. There are numerous examples in the literature (e.g. [Low95], [ACC$^+$08]) of protocols that have been assumed correct for lengthy periods of time before an attack was found. There are two main tactics for verifying layered security protocols. One option is to verify a combined protocol formed from the composition of the application and the transport-layer protocols, using standard protocol analysis techniques. Alternatively, *verification by abstraction* can be used to prove the application-layer protocol correct in isolation, assuming only that the transport protocol satisfies certain abstract properties, rather than considering a particular implementation. For example, WebAuth requires only that the transport layer ensures confidentiality of messages and also authenticates the server to the client.

There are several advantages to verifying protocols by abstraction. Firstly, the proofs are substantially simpler, as the combined protocols are often very large, and transport-layer protocols (particularly TLS) are generally highly complex to model. It also permits the proof of transport layer correctness to be reused for multiple different application-layer protocols. Further, the application-layer proof no longer depends on a particular transport layer being used, meaning that any protocol that provides equivalent (or stronger) properties can be used instead. Lastly, such a

proof often reveals precisely why the transport layer has to satisfy some property to guarantee the security of the application layer. This can be particularly informative for the protocol designer, who may be able to alter the application-layer protocol to weaken its requirements, or even change the transport-layer protocol required.

Proof by Abstraction The first contribution of this thesis is an extension to the *strand spaces model* [TFHG99], the *high-level strand spaces model*, that can be used to prove application-layer protocols correct by abstracting away from the transport-layer protocol. Our model allows the guarantees of a wide variety of transport-layer protocols, including TLS, to be modelled. In contrast to many other techniques, we are also able to model the guarantees provided by so-called *unilaterally authenticating* secure transport protocols, such as *unilateral* TLS. In these protocols, the server is authenticated to the client, but the client is not authenticated to the server, although the server can assume all messages come from the same source. We are also able to model the guarantees provided by transport-layer protocols that provide session-based properties and those protocols that, like TLS, prevent message reordering. In addition, we provide a number of proof rules that can be used to prove the correctness of application-layer protocols. We then illustrate the usefulness of the model by proving the correctness of WebAuth [SA09].

The high-level strand spaces model defined in this thesis is an extension of that developed by Kamil and Lowe [KL09, Kam10]. The main difference between the two versions is that in this thesis, as discussed above, we add support for modelling the guarantees provided by unilaterally authenticating secure transport protocols. Further, the way in which transport-layer protocols that group messages into sessions and prevent message reordering has been improved. More detailed comparisons are made after defining the high-level strand spaces model in Section 3.4.

The majority of existing approaches to verifying layered security protocols [ACC07, DL08, MV09a, BF08, KL09] have focused, in contrast to the high-level strand spaces model, on modelling the security guarantees provided by *bilateral* transport-layer protocols, such as bilateral TLS, where each participant is authenticated to the other. However, many application-layer protocols, in particular those used on the web, are unable to make use of bilateral TLS, because each party must possess a public-key certificate. Therefore, web-based protocols almost exclusively make use of *unilateral* TLS, making it important that protocol analysis techniques can model its guarantees. Further comparisons with related work are given in Section 3.4.

Soundness Proving a layered security protocol correct by abstraction introduces a obligation to prove that the analysis captures all the attacks that verifying the combined protocol would consider. Clearly, such a proof will require a number of assumptions on the application and transport-layer protocols, as it is possible to construct protocols that deliberately break the security guarantees of others by, for example, deliberately leaking keys. Further, it is highly desirable for these assumptions to be *statically checkable*, in that, for a wide class of protocols, the condition should be checkable by syntactic inspection of the protocol messages, or using pre-existing protocol verification tools.

The second contribution of this thesis is a proof of the soundness of the high-level

strand spaces model, with respect to the Dolev-Yao model [DY83]. In particular, we prove that, subject to statically-checkable conditions, whenever there is an attack against the combined protocol, then there is an attack against the application-layer protocol in the high-level model. Further, our condition is satisfied by a wide variety of application and transport-layer protocols, including TLS (although this does require an attack-preserving transformation).

This is a particularly difficult problem for a number of reasons. The central challenge was to ensure that the proof did not impose restrictions that prevented real-world protocols from being modelled. For example, we could have proven the result comparatively easily had we insisted that the messages of the the application and transport layer were entirely disjoint. Clearly, this would have substantially restricted the utility of the proof. Instead, we believe that our static assumptions are satisfied by large classes of both application and transport-layer protocols, including WebAuth and TLS. The other main complication was that the correctness proof cannot assume a particular format for the transport-layer messages, as it requires reasoning about arbitrary compositions of transport and application-layer protocols.

The statically-checkable conditions that we have developed take some inspiration from [GT00] in that the majority of the conditions are concerned with what encryptions can be shared between the application and transport layers. For example, we prohibit any encrypted term from being shared between the application layer and certain parts of the transport layer. We also make several assumptions about what the penetrator initially knows and how the regular agents behave. These assumptions allow us to prove our main result as follows.

In the following, a *bundle* is an collection of runs of a protocol by various agents, possibly involving the penetrator, that satisfies a number of well-formedness conditions, including that every message received must have been sent and that there are no cycles in the bundle.

1. We prove that there exists a class of low-level bundles, known as *interference-free* bundles, that can be converted into high-level bundles that have corresponding (in a formal sense) application-layer behaviour. This is what Chapter 4 achieves by giving the translation between low and high-level bundles. A small part of this proof is based on the work by Kamil and Lowe [KL10, Kam10]. In particular, the way in which values are related between the two strand spaces is based in a large part on the work by Kamil and Lowe. This is clarified further in Section 4.6.

2. Assuming that our statically-checkable condition holds, we prove that any low-level bundle can be transformed to a corresponding (again, in a formal sense) interference-free low-level bundle. This is done in Chapter 5 by specifying the transformation steps required.

3. Lastly, we show that the above transformations preserve protocol failures. In particular, we prove that the transformations from Chapter 5 ensure that if a bundle does not satisfy a particular protocol-correctness property, then neither does the resulting bundle. Further, we show that whenever a low-level bundle does not satisfy a protocol-correctness property, then neither does the high-level bundle created by the transformations of Chapter 4. In order to prove this

result in Chapter 6, we define a logic in which protocol correctness properties can be expressed.

Using the above, we can then observe that the high-level strand spaces model is sound, in the sense that a proof of correctness at the high-level implies there can be no bundle at the low-level that does not satisfy the property.

The soundness of general protocol composition has been widely considered before. Most work [GT00, DDMR07, CD08, ACG+08, CC10] considers when it is sound to compose protocols in *parallel* (i.e. what [GM11] refers to as *horizontal* protocol composition). Generally, the results of these papers require various message disjointness properties to hold, not unlike the properties we consider in this thesis. However, the problem that they consider is subtly different in that we are attempting to prove the correctness of the layers independently, even though messages are formed from those of multiple layers.

There has been relatively little work done on the soundness of techniques that prove protocol correctness by abstraction. The work closest to ours is [GM11], in which the authors show that their analysis technique is sound, assuming a statically-checkable condition. Compared to the assumptions developed in this thesis, the conditions the authors specify are more restrictive and, in particular, TLS does not satisfy their definition. Further comparisons are given in Section 7.4.

Multiple Layers Many application-layer protocols themselves actually consist of more than one layer, or can be viewed as the composition of a number of layers in order to ease analysis. For example, many web-based application-layer protocols authenticate the user by using a username/password exchange on top of a unilateral TLS connection. This can be viewed as the composition of a three-layer protocol, where the bottom layer is unilateral TLS, the middle layer is the username/password protocol and the top layer is the application-layer protocol. In the above, the real application-layer protocol is using the combination of the unilateral TLS channel and the username/password protocol as a bilateral TLS-like channel.

The third and final contribution of this thesis is an extension of the high-level strand spaces model to allow protocols that consist of an arbitrary number of layers, such as the above example, to be verified. In particular, it provides a way of modelling transport-layer protocols, such as the aforementioned username/password protocol, in the high-level strand spaces model.

In order to prove that the multi-layer model is sound, we firstly define what it means for a single channel to be equivalent to the layering of two other channels. Note that this essentially specifies how implementors of multi-layer protocols have to implement the layering. Then, we prove that whenever the combined transport-layer protocol can be attacked (i.e. does not satisfy the required security guarantees), then providing the bottom transport-layer protocol (e.g. unilateral TLS) is correct, there is an attack against the upper of the transport-layers (e.g. the username/password protocol). Further, this attack will be present in the high-level model, thus showing that the high-level analysis is sufficient.

1.1 Related Work

In this section we give a brief overview of relevant related techniques. We make detailed comparisons with: related protocol verification models in Section 3.4; related soundness proofs in Section 7.4; and related multi-layer analysis techniques in Section 8.6.

ProVerif ProVerif [Bla01] is an automated tool that can be used to verify the correctness of security protocols. By using an over-approximation, ProVerif is able to verify systems that are of unbounded size, both in terms of the number of participants in the system, and in the number of runs of the protocol that each participant executes. It is able to verify secrecy properties, and via some extensions [Bla09], also authenticity properties. It does not have explicit support for verifying layered protocols by abstraction, and therefore can only be used to analyse the application-layer protocol in conjunction with a particular secure transport-layer protocol (although there is no reason to suppose that adding support for such analyses would be particularly difficult).

In ProVerif, the protocol being modelled is represented by a set of Horn clauses. Further, the penetrator is modelled by a set of Horn clauses that allow him to manipulate messages that he has received. An efficient solving algorithm is then used to determine if a certain fact can be deduced given the horn clauses. Thus, secrecy of a term can be checked by determining if the term can be derived using the horn clauses. The algorithm is highly effective and is normally able to terminate quickly even on complex protocols. When ProVerif is unable to prove the security of a protocol, it instead attempts to construct an attack on the protocol which it can then report to the user.

Scyther Scyther [Cre08a, Cre08b] is another automated tool that can automatically verify both secrecy and authenticity properties of a wide class of single-layer protocols. Like ProVerif, it does not have explicit support for verification of layered protocols, and thus one has to analyse an explicit combination of the two layers. It is also able to verify an unbounded number of sessions. In contrast to other techniques that provide unbounded results, Scyther is guaranteed to terminate. Further, even when Scyther cannot prove results for an unbounded number of sessions, it is able to provide useful results. In particular, it gives guarantees that are essentially equivalent to the guarantees given by bounded verification tools up to certain bound.

In order to analyse protocols, Scyther considers the traces obtained by running a protocol, and then generates a *characterisation* of the protocol in the form of a finite set of trace patterns. This set is guaranteed to cover the, possibly infinite, set of all possible traces that the protocol could produce. In order to verify secrecy properties, Scyther considers the pattern that corresponds to a violation of the security property. It then checks to see if any of these patterns are produced by the protocol. Interestingly, and unlike many other tools that support unbounded verification, including ProVerif, Scyther does not apply any abstractions to the protocol. This means that by construction, Scyther never produces a false counterexample.

Tamarin Tamarin [SMCB12] is an automated tool that is designed to verify properties of security protocols using a theorem-proving style approach. It has been designed to analyse Authenticated Key Exchange protocols, and is distinguished from other automated techniques in that it supports more powerful penetrators (such as the eCK adversary [LLM07]) who can, for example, compromise certain cryptographic keys. Further, it is able to support equational theories such as Diffie-Hellman, thus allowing many more protocols to be proven correct than using other techniques. In contrast to other automated techniques, Tamarin also supports an interactive mode in which the user can guide it in proving the correctness of a protocol.

Tamarin models the protocol and the penetrator using multiset rewriting rules, thus allowing for the easy specification of many protocols. Correctness properties are specified in a fragment of first-order logic that is sufficient to express all of the usual secrecy and authenticity properties. Tamarin then uses constraint solving in order to search (symbolically) for an execution of the protocol that falsifies the desired property.

Strand Spaces The strand spaces model [TFHG99] allows non-layered security protocols to be proven correct. In contrast to other approaches, it has been principally developed to prove protocols correct, rather than trying to find attacks against a protocol. Since it is a proof technique, it is able to verify protocols that consist of an arbitrary number of sessions. In the strand spaces model, each local session of an honest agent is represented as a sequence of message transmissions and receptions. The penetrator has complete control over all message transmissions and receptions, and is provided with simple ways of manipulating messages. One particular strength of the strand spaces model is its ability to track where a message came from and how the penetrator constructs a term. The latter is particularly useful when proving the soundness results of Chapter 5.

The strand spaces model can also be used to automatically verify protocols by using the tool CPSA [DGT07]. This works by enumerating all of the essentially different *shapes* that the protocol interactions might have, and then checks to see if each shape satisfies the required guarantees. It is principally focused around trying to prove the correctness of protocols that make use of various forms of authentication checks, such as challenge-response exchanges.

The strand spaces model has also been extended to the high-level strand spaces model [KL09, Kam10] by Kamil and Lowe to allow protocols that consist of precisely two layers to be verified. This model allows the guarantees offered by bilaterally authenticating secure transport protocols to be modelled. It also provides limited support for protocols that group messages into sessions and protocols that prevent message from being reordered.

We give more detailed comparisons between Kamil and Lowe's version of the high-level strand spaces model and the version presented in this thesis in Section 3.4.

CSP-Based Approaches In [DL08, Dil11] Dilloway and Lowe use the process algebra CSP [Hoa85, Ros10] to verify, by abstraction, application-layer protocols layered on a single secure transport protocol. Their model allows the guarantees provided by bilateral TLS-like transport-layers to be modelled. They also define a full hierarchy of secure transport protocols, giving examples of each.

In their approach, each honest participant is represented by a CSP process that can send and receive messages from the network. The intruder is represented by a process that is parameterised by the set of values that he knows. The system is then constructed by connecting each of the honest participants to the intruder: thus, in a sense, the intruder is the network. The different guarantees offered by the various secure transport channels are modelled by only allowing the penetrator to overhear certain values. Thus, for example, if a message m is sent over a confidential channel, the intruder is not allowed to add m to his set of known values but can prevent m from being received.

Dilloway and Lowe also extended Casper [Low98] to automatically generate the CSP scripts required to model the protocol. Since the input format for Casper is relatively easy to use, this makes their technique useable by a large number of people. However, their technique can only verify systems that consist of a finite size, for example a single client and server. This means that it cannot be used to formally prove the correctness of protocols, although it can be used to gain a *reasonable* level of confidence. Casper can also be used to verify non-layered protocols and has been extended to verify systems that consist of an unbounded number of sessions [KR05], but only for single-layer protocols.

Pi-Calculus In [BF08] the authors analyse layered security protocols using another process algebra, the pi-calculus [Mil99]. They explain the motivation for their work by observing that, traditionally, in the pi-Calculus most application-layer protocols are implemented using *private channels*, which are extremely difficult to implement in practice. In particular, a private channel ensures messages are always received by the intended recipient, traffic analysis cannot be used to discover the contents of message, and that making a private channel public does not compromise the messages that were sent earlier in the session. These are very difficult guarantees to implement in practice.

In order to analyse layered security protocols, they extend the asynchronous pi-calculus to add support for communications that are sent over secure transport-layer protocols. In this, they use a simple penetrator model that allows the penetrator to intercept, forward or replicate messages that are sent over secure transport layers. The intercept mode has two variants: if a message is sent to an honest agent, then the penetrator can only obtain the transport-layer payload, whilst otherwise the penetrator can obtain the application-layer message itself. Hence, in some sense, this is not *pure* verification by abstraction, since the application-layer analysis has to take into account the format of the transport-layer messages.

Inductive Approach The inductive approach [Pau98] allows protocols that consist of a single layer to be proven correct using the theorem prover Isabelle [NWP02]. In contrast to many of the techniques above, the analysis is not automatic, and instead the user has to guide the theorem prover in order to help it prove the required goals. However, it does support unbounded verification, meaning that it can be used for actually proving the correctness of protocols. The inductive approach is formalised using *event lists*, which are conceptually execution traces of the system. Each event of the system either denotes an agent injecting a message into the network, retrieving a message from the network, or changing internal state. The honest

agents and the intruder are then defined as predicates over event lists.

[BLP03] extends the inductive approach to verify application-layer protocols that are layered on a simple type of secure transport protocol. This allows the guarantees provided by confidential and authentic channels to be specified relatively easily. In order to add support for this new events are added that correspond to messages being sent over confidential and authentic channels. The penetrator is then given more restricted rules for manipulating messages that are sent over such channels.

Pseudonymous Channels In [MV09a] the authors define a model-checking based approach that allows application-layer protocols that are layered on either unilaterally or bilaterally authenticating secure transport protocols to be verified. The authors formalise their model in the context of *The Intermediate Format*, or IF, that is a transition system along with a set of rules that specify which states are considered to be attack states. Each state corresponds to a set of *facts* that are true, whilst the transition relation relates sets of facts. The penetrator model is constructed by adding transitions that correspond to each of the ways the penetrator can manipulate values (e.g. concatenating, encryption etc.). A protocol is then considered to be secure if none of the attack states are reachable from the initial state.

Protocols defined in this model can be verified using the symbolic model-checker, OFMC [MV09b]. Originally, OFMC could only be used to verify systems of some particular finite size. However, it has since been extended [BM10] in order to allow systems of an unbounded size to be verified. In order to do this, OFMC computes progressively tighter abstractions of the protocol, until a fixpoint that contains no false attacks is reached. This is then passed to a modified version of the Isabelle theorem prover [NWP02], which uses this fixpoint in order to try and construct a proof of the protocol correctness.

LTL In [ACC07] the authors give an LTL-based model checking approach that allows security protocols that consist of no more than two layers to be verified by abstraction. The formalism supports some basic channel types that are confidential and may or may not prevent replay attacks. It does not support the guarantees provided by unilaterally authenticating secure transport protocols.

This LTL-based approach is formalised in a similar way to the above approach using OFMC. Again, the protocol is represented as a transition system where each state represents a set of *facts* that are true. As with OFMC, transitions relate sets of facts and the penetrator is modelled as a set of symbolic transitions between symbolic states that allow him to deduce new facts, assuming that he knows other facts. Given this model, it is easy to model the guarantees provided by different secure transport-layers. For example, a confidential channel that ensures messages are only readable by the intended recipient will only allow the penetrator to deduce m if the intended recipient is the penetrator. As with other model checking based techniques, this does not support verification of systems of unbounded size, but instead only checks to see if there is an attack for which the trace length is bounded by a constant k. Thus, this technique is not really able to formally verify protocols, but it can be used in order to find attacks.

1.2 Outline

In Chapter 2 we give an overview of the standard low-level strand spaces model of [TFHG99]. In Chapter 3 we define the high-level strand spaces model, concentrating on modelling the guarantees provided by both unilaterally and bilaterally authenticating secure transport protocols. We also give a number of proof rules that can be used to prove application-layer protocols correct. We then demonstrate the effectiveness of the rules by proving the correctness of WebAuth [SA09].

In Chapter 4 we formalise the relationship between the two different verification techniques and then prove that, subject to a *semantic* assumption, the high-level strand spaces model is sound. In Chapter 5 we develop our statically checkable assumption and prove that, if a particular combination of application and transport-layer protocols satisfies the condition, then it can be transformed to a bundle that satisfies the semantic condition of the previous section. In Chapter 6 we define a logic in which protocol-correctness properties can be expressed and then prove that correctness property dissatisfaction is preserved by the transformations of the previous section and by abstraction. In Chapter 7, we prove that TLS can be transformed to a form that satisfies our statically checkable assumptions whilst preserving any attacks.

In Chapter 8 we extend the high-level strand spaces model to allow protocols with an arbitrary number of layers to be proven correct. In particular, we define a way of modelling and proving the correctness of transport-layer protocols in the high-level strand spaces model. We also prove, using the results from the previous chapters, that any such analysis is sound. Lastly, in order to illustrate the model's effectiveness, we prove the correctness of a couple of simple example protocols.

Lastly, in Chapter 9 we summarise the results of the thesis, give further comments on related work and detail some future work. In Appendix A we give a summary of the notation used throughout the thesis. In Appendix B we summarise the assumptions that were made in the thesis.

Chapter 2

Introduction to Strand Spaces

The *strand spaces* model, first proposed in [TFHG99], is principally concerned with providing a theory in which protocols can be proven correct under the Dolev-Yao assumptions [DY83]. One major advantage of it is that it makes the conditions under which protocols are secure explicitly clear. Its main strength as a proof technique is that it allows the sources and ordering of the messages to be easily deduced.

In this chapter we review how the strand spaces model can be used to analyse standard security protocols (i.e. protocols with no layering). Firstly, in Section 2.1, we describe how messages are represented in the model. Then, in Section 2.2 we describe how protocol interactions are represented before, in Section 2.3, defining how the penetrator can interact with messages. In Section 2.4 we describe how to simplify proofs by reducing the number of possible penetrator interactions.

2.1 Term Algebra

In this section we define how protocol messages are represented in the strand spaces model. The basic components of protocol messages are *terms*; a term could be an atom, a concatenation of several terms, an encryption of a term or a cryptographic key. In order to allow transport-layer protocols to be modelled, we extend the strand spaces model to allow encryption keys to be *generated* from other terms.

Definition 2.1 (From [TFHG99])**.** The set of atoms $\mathcal{T}$ includes atomic messages and two distinct subsets, firstly of names $\mathcal{T}_{names}$ and secondly of cryptographic keys $\mathcal{K}$ with a distinct subset of symmetric keys $\mathcal{K}_{sym}$. We assume that the set of names is partitioned into two distinct[1] subsets, $\mathcal{T}_{names}^{reg}$ and $\mathcal{T}_{names}^{pen}$ that represent the set of names that honest (regular) agents and dishonest (penetrator) agents use, respectively. We also assume that there exists a function giving the inverse of a key $k \in \mathcal{K}$, written k^{-1} and functions giving the public and secret key of a name, which we write as $PK(A)$ and $SK(A)$ for $A \in \mathcal{T}_{names}$.

The set of *terms* $\mathcal{A}$ is freely generated from the closure of $\mathcal{T}$ under encryption (written as $\{|m|\}_k$) and concatenation (written $t_1 \hat{} t_2$). $\hat{}$ is defined as being right associative, i.e. $t_1 \hat{} t_2 \hat{} t_3$ is interpreted as $t_1 \hat{} t_2 \hat{} t_3$.

The set of *key generation functions* Kgf consists of functions that generate symmetric keys from terms; hence for all $g \in \mathsf{Kgf}$, $g : \mathcal{A} \to \mathcal{K}$. We assume that each

[1] We relax this slightly in the next section.

function $g \in \mathsf{Kgf}$ is injective and that distinct key generation functions have disjoint ranges. A key $k \in \mathcal{K}$ is *complex* iff there exists $g \in \mathsf{Kgf}$ and $t \in \mathcal{A}$ such that $g(t) = k$, and is *simple* otherwise. If k is complex then k is symmetric, i.e. $k = k^{-1}$.

We assume an ordering relation, the *subterm* relation, $\sqsubseteq$, over $\mathcal{A}$ such that $t_1 \sqsubseteq t_2$ iff t_1 can be obtained from t_2 by a finite number of decryption and splitting operations. The relation $\mathsf{ingredient}$ is defined as the least reflexive transitive relation such that: k $\mathsf{ingredient}$ $\{\!|m|\!\}_k$, m $\mathsf{ingredient}$ $\{\!|m|\!\}_k$, t_1 $\mathsf{ingredient}$ $t_1 \hat{} t_2$, t_2 $\mathsf{ingredient}$ $t_1 \hat{} t_2$, and t $\mathsf{ingredient}$ $g(t)$.

The set $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}$ of public terms is the set of terms that the penetrator initially knows. The sets $\mathcal{T}_\mathcal{P} = \mathcal{A}_\mathcal{P} \cap \mathcal{T}$ and $\mathcal{K}_\mathcal{P} = \mathcal{A}_\mathcal{P} \cap \mathcal{K}$ are the sets of atoms and keys initially known to the penetrator, respectively[2].

Note that the $\sqsubseteq$ relation is defined such that $k \not\sqsubseteq \{\!|m|\!\}_k$; intuitively, k is not contained within (and cannot be extracted from) $\{\!|m|\!\}_k$ as it is used only to construct the message. In contrast, k $\mathsf{ingredient}$ $\{\!|m|\!\}_k$, and indicates that k is required in order to construct the message $\{\!|m|\!\}_k$.

Whilst the term algebra does not include explicit support for hash functions, it is possible to model protocols that use hash functions by making use of key generation functions. For example, hashing a term t using a hash function h could be modelled as encrypting 0 using h as a key-generation function; i.e. $\{\!|0|\!\}_{h(t)}$. Note that this is equivalent to hashing since the algebra provides no way of inverting the key generation function to reveal the the value of t.

Since $\mathcal{A}$ is freely generated from $\mathcal{T}$ it follows, for example, that $m_1 \hat{} m_2 = m_3 \hat{} m_4$ iff $m_1 = m_3$ and $m_2 = m_4$. Further, no concatenation can ever be mistaken for an encryption, or vice-versa. This does preclude *type-flaw attacks* where the intruder exploits the fact that a message can be interpreted in two different ways.

Note that the way we model key-generation functions differs from [Kam10, KL11] where the authors instead defined key-generation functions as taking an arbitrary number of *atoms* as arguments, rather than terms. Further, they added explicit support for hash functions in the algebra. We believe that the above model is more general, in that it allows for keys to be generated from arbitrary terms. Further, by not differentiating between hash functions and key-generation functions, we simplify some of the formalisation. We discuss further differences between the approaches after Definition 2.15.

2.2 Strand Spaces

We now describe how to actually model the protocol interactions. A *strand* represents one principal's view of one run of the protocol; in particular it consists of a sequence of message transmissions (written $+t$) and receptions (written $-t$). For example, the strand $\langle -t, +t \rangle$ represents a principal who received one message, t, and then immediately echoed it out. A *strand space* is an arbitrary set of strands; however, generally a strand space will be defined relative to a particular protocol and will contain all strands that are possible in that protocol.

[2]Note that this differs from [TFHG99], which assumed a set of public atoms only. We use a set of public terms instead in order to allow the penetrator to initially know values that are encrypted, but for which he does not know the plaintext. This is useful when modelling protocols, such as WebAuth (cf. Section 3.3), where participants are issued with tokens that only the issuer can decrypt.

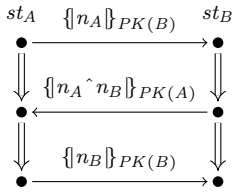


Figure 2.1: A graphical representation of a bundle containing one complete run of the Needham-Schroder protocol.

It is worth noting that the strand spaces model makes no assumption on how messages are sent and received; it assumes only that when receiving a message there is a unique sending strand and when sending a message that multiple strands may simultaneously receive it.

Definition 2.2 (From [TFHG99])**.** A *signed term* is a pair (σ, a) with $a \in \mathcal{A}$ and $\sigma \in \{+, -\}$. A signed term is written as either $+t$ or $-t$. The set of finite sequences of signed terms is denoted by $(\pm\mathcal{A})^*$ and a typical element is of the form $\langle(\sigma_1, t_1), \ldots, (\sigma_n, t_n)\rangle$.

Definition 2.3 (From [TFHG99])**.** A *strand space* over $\mathcal{A}$ is a set Σ together with a trace mapping $tr : \Sigma \to (\pm\mathcal{A})^*$. Fix a strand space Σ:

1. A *node* is a pair (s, i) with $s \in \Sigma$ and $i \in \{1..|tr(s)|\}$. We say that the node (s, i) *belongs* to the strand s. Note that every node belongs to a unique strand. The set of nodes is denoted by $\mathcal{N}$.

2. If $n = (s, i) \in \mathcal{N}$ then $index(n) = i$ and $strand(n) = s$. Furthermore, $msg(n)$ is defined to be $tr(s)_i$.

3. There is an edge $n_1 \to n_2$ if and only if there exists $a \in \mathcal{A}$ such that $msg(n_1) = +a$ and $msg(n_2) = -a$.

4. There is an edge $n_1 \Rightarrow n_2$ if and only if $n_1 = (s, i)$ and $n_2 = (s, i + 1)$. The transitive closure of $\Rightarrow$ is written $\Rightarrow^+$.

5. An unsigned term t originates at a positive node $n \in \mathcal{N}$ iff $t \sqsubseteq msg(n)$ and, for all nodes n' such that $n' \Rightarrow^+ n$, $t \not\sqsubseteq msg(n')$.

6. An unsigned term t is *uniquely originating* iff t originates on a unique $n \in \mathcal{N}$.

A *bundle*, defined formally below, represents possible real-world runs of the protocol; in particular it is a set of strands such that every message that is received by a strand is sent by another strand in the bundle. For example, given the Needham-Schroeder public-key protocol [NS78], the set containing the following strands:

$$\langle +\{\!|n_A|\!\}_{PK(B)}, -\{\!|n_A \hat{} n_B|\!\}_{PK(A)}, +\{\!|n_B|\!\}_{PK(B)}\rangle,$$
$$\langle -\{\!|n_A|\!\}_{PK(B)}, +\{\!|n_A \hat{} n_B|\!\}_{PK(A)}, -\{\!|n_B|\!\}_{PK(B)}\rangle$$

is a bundle representing exactly one complete run of the protocol. A graphical representation of this bundle can be seen in Figure 2.1.

Observe that the above definition of a strand space allows us to view a strand space as a connected graph $G = (\mathcal{N}, \rightarrow \cup \Rightarrow)$. We can therefore define a bundle as a sub-graph of G that satisfies certain well-formedness conditions, as follows.

Definition 2.4 (From [TFHG99])**.** Suppose $\rightarrow_{\mathcal{B}} \subset \rightarrow$, $\Rightarrow_{\mathcal{B}} \subset \Rightarrow$ and $\mathcal{B} = (\mathcal{N}_{\mathcal{B}}, \rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}})$ is a subgraph of $(\mathcal{N}, \rightarrow \cup \Rightarrow)$. Then $\mathcal{B}$ is a *bundle* iff:

1. $\mathcal{B}$ is finite;
2. If $n_2 \in \mathcal{N}_{\mathcal{B}}$ and $msg(n_2)$ is negative then there exists a unique n_1 such that $n_1 \in \mathcal{N}_{\mathcal{B}}$ and $n_1 \rightarrow_{\mathcal{B}} n_2$;
3. If $n_2 \in \mathcal{N}_{\mathcal{B}}$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_{\mathcal{B}} n_2$;
4. $\mathcal{B}$ is acyclic.

The set of regular nodes in a bundle $\mathcal{B}$ is denoted by $\mathcal{N}_{\mathcal{B}}^{reg}$.

Note that it is possible to define a bundle such that certain values (e.g. certain encryption keys, or nonces) are created only on one strand (or on no strands at all). This is very important since most security protocols depend on certain values only originating once for their security and integrity. Therefore, almost all strand spaces proofs include an assumption that a certain value is only created on one strand in a particular bundle.

Definition 2.5 (From [TFHG99])**.** A node n is *in* a bundle $\mathcal{B} = (\mathcal{N}_{\mathcal{B}}, \rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}})$, written $n \in \mathcal{B}$, iff $n \in \mathcal{N}_{\mathcal{B}}$. A strand s is in $\mathcal{B}$ iff a non-empty subset of its nodes are in $\mathcal{N}_{\mathcal{B}}$.

It is sometimes useful to be able to express that a particular principal in a protocol has got beyond a certain point. For example, in the case of the Needham-Schroeder protocol one correctness property could be expressed as: if the responder has just received the third message then there must exist an initiator strand of length 3 such that they agree on n_a and n_b. Formally we define the notion of strand *height*, which allows us to express such properties, as follows.

Definition 2.6 (From [TFHG99])**.** If $\mathcal{B}$ is a bundle then the $\mathcal{B}$-*height* of a strand s is the largest i such that $(s, i) \in \mathcal{B}$.

When considering where a term originated it is often necessary to reason about the ordering of several nodes, possibly on different strands. Therefore, we define a relation $\prec$ that orders any pair of nodes where there is a path between them in the bundle graph.

Definition 2.7 (From [TFHG99])**.** If $\mathcal{S}$ is a set of edges, i.e. $\mathcal{S} \subseteq \rightarrow \cup \Rightarrow$, then $\prec_{\mathcal{S}}$ is the transitive closure of $\mathcal{S}$ and $\preceq_{\mathcal{S}}$ is the reflexive, transitive closure of $\mathcal{S}$.

Lemma 2.8 (From [TFHG99])**.** Let $\mathcal{B}$ be a bundle; then $\preceq_{\mathcal{B}}$ is a partial order (i.e. a reflexive, antisymmetric, transitive relation). Every non-empty subset of the nodes in $\mathcal{B}$ has $\preceq_{\mathcal{B}}$-minimal members (this is abbreviated to $\preceq$ when the bundle is clear; further minimal will always mean $\preceq_{\mathcal{B}}$-minimal).

2.3 The Penetrator

The capabilities of the penetrator are defined by the operations that he can perform. In the case of the strand spaces model this means that we need to define strands that allow the penetrator to manipulate messages in appropriate ways to expose attacks. In particular we give the penetrator the ability to:

- Encrypt any message for which he knows the key and the message;
- Decrypt any message for which he knows the decryption key;
- Concatenate and split messages;
- Send any message he initially knows (i.e. any message from $\mathcal{A}_\mathcal{P}$).

Note that we do not allow the penetrator to perform any cryptanalysis as we assume perfect encryption as in the Dolev-Yao model. We define the *penetrator strands* that correspond to these abilities below.

Definition 2.9 (From [TFHG99] and [KL11])**.** A penetrator strand is one of the following:

M Text message: $\langle +t \rangle$ where $t \in \mathcal{A}_\mathcal{P}$;

C Concatenation: $\langle -t_0, -t_1, +t_0 \hat{} t_1 \rangle$;

S Separation into components: $\langle -t_0 \hat{} t_1, +t_0, +t_1 \rangle$;

E Encryption: $\langle -k, -t, +\{\!|t|\!\}_k \rangle$;

D Decryption: $\langle -k^{-1}, -\{\!|t|\!\}_k, +t \rangle$;

KG Key generation: $\langle -t, +g(t) \rangle$ where $g \in \mathsf{Kgf}$.

On a D or E strand the incoming edge with k on is known as the *key edge*.

It is possible to model a more powerful penetrator by defining extra penetrator strands. For example, if it was desired to verify a protocol under the assumption that the encryption being used was vulnerable to known-plaintext attacks the penetrator strand $\langle -m, -\{\!|m|\!\}_k, +k \rangle$ for $m \in \mathcal{A}$, $k \in \mathcal{K}$, could be added.

Definition 2.10 (Adapted From [TFHG99])**.**

- An *infiltrated strand space* is a tuple $(\Sigma, P, \mathcal{A}_\mathcal{P})$ with Σ a strand space, $P \subseteq \Sigma$ and containing only penetrator traces, and $\mathcal{A}_\mathcal{P} \subseteq \mathcal{A}$ containing the penetrator's initial knowledge.
- A strand s is a *penetrator strand* iff $s \in P$ and a regular strand otherwise.
- A node n is a *penetrator node* iff the strand it lies on is a penetrator strand and a regular node otherwise.

One useful concept is that of *penetrator paths*; these are sequences of penetrator nodes where the first and last nodes are also allowed to be regular and that correspond to transformations performed by the penetrator on a message.

Definition 2.11 (From [GT02])**.** A *path* p through a bundle $\mathcal{B}$ is a finite sequence of nodes and edges such that for each consecutive pair of nodes m and n either $m \rightarrow n$, or $m \Rightarrow^+ n$ with $msg(m)$ negative and $msg(n)$ positive. A *penetrator path* is a path where every node, aside from possibly the first and the last, is a penetrator node.

Two bundles are defined to be equivalent if they have the same regular behaviour; i.e. it does not matter how the penetrator achieves certain regular behaviour, only that he has managed to cause it.

Definition 2.12 (From [GT02])**.** Bundles $\mathcal{B}$, $\mathcal{B}'$ of a strand space Σ are *equivalent* iff they have the same regular nodes.

2.4 Penetrator Efficiency

Clearly, the number of interleavings of the penetrator strands defined above is extremely large since the penetrator may carry out arbitrarily many sessions in parallel and may put the strands together in any order. This complicates proofs of protocol correctness and thus, in this section, we consider how to reduce the number of possible combinations.

In [GT02] Guttman and Thayer develop the notion of *normal* bundles as well as considering *redundancies* in bundles. They prove that for any bundle there is an equivalent normal, redundancy-free bundle.

Guttman and Thayer identify an inefficiency in [GT02] when the penetrator builds a value and then immediately destroys it. There are two different ways in which this can be done:

1. The intruder first encrypts a value t with k (on a E strand) and then decrypts it using k^{-1} (with a D strand);

2. The intruder first concatenates t_0 and t_1 (using a C strand) before splitting them (on a S strand).

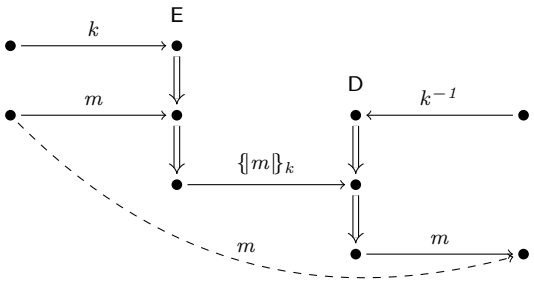
Clearly in both these cases the redundancies can be easily eliminated as show in Figure 2.2. We formalise this as follows.

Definition 2.13 (From [GT02])**.** A *redundancy* in a bundle $\mathcal{B}$ is any labelled subgraph of $\mathcal{B}$ that consists of either an E strand followed by a D strand, or a C strand followed by a S strand.

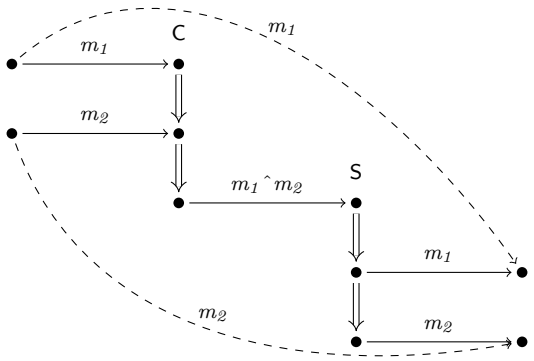
Lemma 2.14 (From [GT02])**.** Given any bundle $\mathcal{B}$ there exists an equivalent bundle $\mathcal{B}'$ that contains no redundancies.

We can generalise the above concept by categorising the penetrator strands into three different types; *constructive*, *destructive* and *initial*. A constructive strand creates a term larger than its input; a destructive strand creates a term smaller than its input and an initial strand introduces a new term.

Definition 2.15 (Based on [GT02])**.** A $\Rightarrow^+$-edge is *constructive* if it is part of a E, KG or C strand. It is *destructive* if it is part of a D or S strand. A penetrator node is *initial* if it is a M node.



(a) Eliminating E-D redundancies.



(b) Eliminating C-S redundancies.

Figure 2.2: Eliminating redundancies from bundles; the dashed lines represent how the $\rightarrow$ relation is redefined in the resulting redundancy-free bundle.

In [Kam10, KL11] the authors instead defined KG strands as destructive, rather than constructive. This made sense in their model where all of the inputs to a key generation strand had to be atoms, and thus would naturally occur after other destructive strands. Since KG strands are defined as taking terms in this thesis, rather than atoms, it is no longer natural to define KG strands as destructive. Defining KG strands as constructive will actually simplify many of the proofs, particularly in Chapter 5.

Definition 2.16 (Based in [GT02])**.** A bundle $\mathcal{B}$ is *normal* if for any penetrator path of $\mathcal{B}$, whenever a constructive edge sends to a destructive edge, the constructive edge must be a KG strand (i.e. the KG strand must feed into the key-edge).

Intuitively, a normal bundle is one where the penetrator first de-constructs any term that it needs in order to build the new term, and only then starts constructing the outgoing term. In particular, if the bundle does not contain KG strands, then every destructive edge precedes every constructive edge (i.e. no destructive edge occurs after a constructive edge).

Lemma 2.17 (Bundle Normal Form Lemma — Based on [GT02])**.** For any bundle there exists an equivalent normal bundle.

Proof. This is a straightforward adaptation of the proof from [GT02]. □

Chapter 3

Verifying Protocols by Abstraction

In this chapter we define an extension of the strand spaces model of the previous chapter, the *high-level strand spaces model*, which can be used to verify protocols that consist of precisely two layers. In particular, it enables protocols to be proven correct by abstracting away from the underlying transport-layer protocol. Further, the model is able to support the guarantees provided by a wide-variety of transport-layer protocols, including both unilaterally and bilaterally authenticating secure transport protocols.

The model we present in this chapter is based on that given by Kamil and Lowe in [KL09]. The main difference between the two versions is that in this chapter, as discussed in Chapter 1, we add support for modelling the guarantees provided by unilaterally authenticating secure transport protocols. Further, the way in which transport-layer protocols that group messages into sessions and prevent message reordering has been improved. We discuss the precise differences in Section 3.4.

In Section 3.1 we define the high-level strand spaces model. In Section 3.2, we define and prove some proof rules that can be used to easily construct correctness proofs for application-layer protocols. In Section 3.3 we then prove the utility of these proof rules by proving the correctness of WebAuth [SA09], a single-sign-on protocol.

The contents of this chapter previously appeared in [GRL11]. The only difference with the version that is presented here is that the model is now able to model the guarantees provided by transport-layer protocols that prevent messages from being reordered.

3.1 The High-Level Strand Spaces Model

In this section we actually define the high-level strand spaces model. We start, in Section 3.1.1, by defining the basics of the high-level strand spaces model, including the necessary adaptations to support unilaterally authenticating secure transport protocols. In Section 3.1.2 we define the operations that the penetrator is allowed to perform, in particular adding support for the penetrator manipulating messages sent over secure channels. Lastly, in Section 3.1.3, we adapt the low-level notion of normality defined in Definition 2.16 to the high-level strand spaces model, in order to simplify the proofs that we give.

3.1.1 Basic Definitions

We consider a *channel* as an object that allows two participants to exchange messages: messages sent at one end of the channel are intended to be received at the other end. A fundamental property of many transport protocols, including TLS, is that a principal is able to send or receive on a channel only if she has the relevant cryptographic keys; these are different in different channels, which prevents messages being replayed between sessions.

A *channel end* conceptually encapsulates all the information that is required to communicate on a channel. However, as we abstract away from the details of the transport protocol, we treat channel ends as opaque values. For generality, we also consider channels (for protocols that are weaker than TLS) that do not provide such a separation between sessions; we denote the channel end by ? in such cases. Messages will be addressed by *channel endpoints* that consist of a name and a channel end. In channel endpoints we also permit the name to be ?; this corresponds to a message where the sender has no name, as in the case of unilateral TLS. We therefore expand $\mathcal{T}_{names}$ to include ?.

Definition 3.1. We assume a set of *channel types*, *Channels*, that contains a value $\perp$ that represents the channel that provides no security guarantees. In examples, we write $\mathit{TLS}^{C \to S}$ and $\mathit{TLS}^{S \to C}$ to represent the channel types of a unilateral TLS connection from client to server and server to client respectively.

$\mathcal{C}$ denotes the set of *channel ends*. It has two subsets: *penetrator channel ends*, $\mathcal{C}^{pen}$, known to the penetrator; and *regular channel ends*, $\mathcal{C}^{reg}$, known only to regular agents; we have $\mathcal{C}^{reg} \cap \mathcal{C}^{pen} = \{?\}$.

The set of *regular endpoints* $\mathcal{I}^{reg}$ is defined as $\mathcal{T}^{reg}_{names} \times \mathcal{C}^{reg}$; the set of *penetrator endpoints* $\mathcal{I}^{pen}$ as $\mathcal{T}^{pen}_{names} \times \mathcal{C}^{pen}$; and the set of *endpoints* $\mathcal{I}$ as $\mathcal{I}^{reg} \cup \mathcal{I}^{pen}$ (note that $\mathcal{I}^{reg} \cap \mathcal{I}^{pen} = \{(?, ?)\}$). We denote a typical member, $(A, \psi) \in \mathcal{I}$, as A_ψ. Given an endpoint A_ψ, $\mathit{name}(A_\psi) = A$ and $\mathit{end}(A_\psi) = \psi$.

The set of sequence numbers, denoted $\mathcal{S}$, is defined as $\mathbb{N}^+ \cup \{\_\}$ where _ indicates no sequence number[1] (which is used when the protocol does not consider the order in which messages are received). We assume honest agents check that incoming messages are being received in the correct order, and that any messages they send are numbered correctly.

High-level terms model data being sent across the network. They are of the form $(A_\psi,\ B_\phi,\ i,\ m,\ c)$, which represents that this is the i^{th} application-layer message sent from A's channel end ψ to B's channel end ϕ along a channel of type c containing an application-layer term m.

Definition 3.2 (Based on [KL09])**.** A *high-level term* is a tuple of the form $\sigma(S_\psi,\ R_\phi,\ i,\ m,\ c)$ where:

- $\sigma \in \{+, -\}$ which represents a message being sent or received respectively;
- $S_\psi \in \mathcal{I}$: is the claimed sender of m;

[1] There is no reason why we could not allow an arbitrary partially ordered set to be used as the set of sequence numbers. In particular, the soundness proofs of later sections would not require any alteration. We make the above restriction in order to simplify the presentation.

- $R_\phi \in \mathcal{I}$: is the intended recipient of m;
- $i \in \mathcal{S}$: is the sequence number of this message;
- $m \in \mathcal{A}$ is the application-layer message;
- $c \in \mathit{Channels}$ is the channel type along which the term is communicated.

Let $\hat{\mathcal{A}}$ denote the set of high-level terms. The set of finite sequences of high-level terms is denoted by $\hat{\mathcal{A}}^*$. We abbreviate $(S_\psi,\ R_\phi,\ \_,\ m,\ c)$ to $(S_\psi,\ R_\phi,\ m,\ c)$. Given a high-level term $t = (S_\psi,\ R_\phi,\ s,\ m,\ c)$, we define $\mathit{sender}(t) \mathrel{\widehat{=}} S_\psi$, $\mathit{recipient}(t) \mathrel{\widehat{=}} R_\phi$, $\mathit{seqno}(t) \mathrel{\widehat{=}} s$, $\mathit{appmsg}(t) \mathrel{\widehat{=}} m$ and $\mathit{chan}(t) \mathrel{\widehat{=}} c$.

From the above definition of high-level terms, *high-level nodes*, *strands*, *high-level strand spaces*, *high-level bundles* and high-level origination can be defined analogously to the standard case presented in Chapter 2. We also lift the definitions of $\mathit{sender}(t)$, $\mathit{recipient}(t)$ etc to be defined over high-level nodes by defining $\mathit{sender}(n)$ as $\mathit{sender}(\mathit{msg}(n))$, for example.

Channel types determine the acceptable format of high-level terms. In particular, they determine whether the sequence number is _, or has to be from $\mathbb{N}^+$ and, in addition, they determine the permissible channel endpoints. For example, if two high-level terms both have the same channel type then either both have _ as the sequence number, or neither do. Further, if the channel was a bilateral TLS channel then the sender and recipient endpoints A_ψ and B_ϕ could not contain ?, either as a name or a channel end; conversely, if the channel was a $\textit{TLS}^{C \to S}$ channel, where the sender's name is not authenticated, then the sender's channel end would be of the form $?_\psi$. These assumptions are formalised in the following assumption.

In the following we also assume that a high-level term only uses sequence numbers if it also uses channel ends. This is because, without channel ends, it is not possible to group messages into logical sessions and thus it is impossible to decide which messages must have consecutive sequence numbers.

Assumption 3.3. For every channel $c \in \mathit{Channels}$:

1. Either every high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $S = ?$, or none has $S = ?$;
2. Either every high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $\psi = ?$, or none has $\psi = ?$;
3. Either every high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $R = ?$, or none has $R = ?$;
4. Either every high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $\phi = ?$, or none has $\phi = ?$;
5. Either every high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $i = \_$, or none has $i = \_$.
6. If a high-level term $(S_\psi,\ R_\phi,\ i,\ m,\ c) \in \hat{\mathcal{A}}$ has $i \neq \_$, then $\psi \neq ?$ and $\phi \neq ?$.

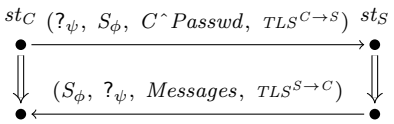


Figure 3.1: An example bundle illustrating how a regular agent on strand st_C can retrieve email from a server on strand st_S via a hypothetical protocol.

Note that if a strand makes exclusive use of bilateral protocols then the sender's name will typically be the same on each node and would be the name contained in her certificate.

Definition 3.4. Fix a strand space Σ. The function $endpoints : \Sigma \to \mathcal{P}(\mathcal{I}^{reg})$ gives the set of endpoints that a regular strand uses. If n is a node on regular strand st with $msg(n) = \sigma(S_\psi,\ R_\phi,\ m,\ c)$, then: if $\sigma = +$ then $S_\psi \in endpoints(st)$; and if $\sigma = -$ then $R_\phi \in endpoints(st)$. Similarly, $ends : \Sigma \to \mathcal{P}(\mathcal{C}^{reg})$ gives the set of channel ends that a regular strand uses. Further, we assume that in every high-level bundle, channel ends are partitioned by strand. Formally, for every pair of regular strands st and st' in $\mathcal{B}$:

$$st \neq st' \implies ends(st) \cap ends(st') \subseteq \{?\}. \tag{3.1}$$

As an example consider Figure 3.1. This bundle represents a client retrieving e-mail via a hypothetical protocol that makes use of unilateral TLS. It contains a client strand st_C and a server strand st_S, such that $ends(st_C) = \{\psi\}$ and $ends(st_S) = \{\phi\}$. All messages are sent over a unilateral TLS channel where the server is authenticated and the client is not; therefore, the client is identified as ? in every high-level term.

This example also illustrates the necessity of channel ends. Note that we need to ensure that *Messages* is sent back only to the source of *Passwd*, even though the client herself isn't authenticated by the transport channel. Without the channel ends, a message sent along a unilateral channel c to the unauthenticated end would be represented by $(S,\ ?,\ m,\ c)$. As this representation does not identify the recipient in any way, it would not be possible to decide if the penetrator was allowed to receive the message. Later we will specify that, for TLS-like channels, the penetrator is able to send and receive messages only on penetrator channel ends (i.e. in $\mathcal{C}^{pen}$), thus ensuring he is unable to receive the second message.

We now make an extra assumption that sequence numbers, if used, are properly implemented. In particular, we need to assume that, on a regular strand between any two channel ends, the sequence numbers of the messages (if not _) are contiguous and start from 1 (i.e. they are a prefix of $\langle 1..\rangle$).

Assumption 3.5. For all regular strands st in Σ, the function $th^{st} : \mathcal{C} \times \mathcal{C} \to \mathcal{S}^*$, which returns the list of sequence numbers sent between two channel ends on a strand, is defined by:

$$th^{st}(\psi, \phi) \mathrel{\widehat{=}} \langle seqno(msg(st, i)) \mid i \in \langle 1..\rangle, i \leq |st|,\\ end(sender(msg(st, i))) = \psi, end(recipient(msg(st, i))) = \phi\rangle.$$

For all regular strands st, and all $\psi, \phi \in \mathcal{C}$, either $th^{st}(\psi, \phi) \leq \langle 1..\rangle$, or $th^{st}(\psi, \phi) \in \{\_\}^*$.

3.1.2 The Penetrator

We now consider how to model the capabilities of the penetrator. We start with messages that are sent over the unprotected channel $\perp$ (i.e. not over a secure transport protocol). Clearly, we need to model the penetrator as the full Dolev-Yao penetrator and therefore we allow the penetrator to perform all the usual actions.

Definition 3.6 (From [KL09])**.** The *application-layer penetrator strands* are strands of the following form:[2]

M Text message: $\langle +(?,\ ?,\ \_,\ r,\ \perp)\rangle$ where $r \in \mathcal{A}_{\mathcal{P}}$;

C Concatenation: $\langle -(?,\ ?,\ \_,\ t_0,\ \perp), -(?,\ ?,\ \_,\ t_1,\ \perp), +(?,\ ?,\ \_,\ t_0\hat{}t_1,\ \perp)\rangle$;

S Separation: $\langle -(?,\ ?,\ \_,\ t_0\hat{}t_1,\ \perp), +(?,\ ?,\ \_,\ t_0,\ \perp), +(?,\ ?,\ \_,\ t_1,\ \perp)\rangle$;

E Encryption: $\langle -(?,\ ?,\ \_,\ k,\ \perp), -(?,\ ?,\ \_,\ t,\ \perp), +(?,\ ?,\ \_,\ \{|t|\}_k,\ \perp)\rangle$ where $k \in \mathcal{K}$;

D Decryption: $\langle -(?,\ ?,\ \_,\ k^{-1},\ \perp), -(?,\ ?,\ \_,\ \{|t|\}_k), +(?,\ ?,\ \_,\ t,\ \perp)\rangle$ where $k \in \mathcal{K}$;

KG Key generation: $\langle -(?,\ ?,\ \_,\ t,\ \perp), +(?,\ ?,\ \_,\ g(t),\ \perp)\rangle$ where $g \in \mathsf{Kgf}$.

We now consider what the penetrator can do with messages sent over a secure transport channel. For ease of exposition we firstly consider a restricted model in which the only secure transport protocols are bilateral and unilateral TLS. Clearly, we must allow the penetrator to receive messages intended for him, and to send messages coming from himself.

Definition 3.7 (Based on [KL09])**.** The *transport-layer penetrator strands* for TLS-like protocols are of the following form, where $c \neq \perp$:

SD Send: $\langle -(?,\ ?,\ \_,\ m,\ \perp), +(P_\psi,\ B_\phi,\ i,\ m,\ c)\rangle$ where $P_\psi \in \mathcal{I}^{pen}$, $B_\phi \in \mathcal{I}^{reg}$, $i \in \mathcal{S}$;

RV Receive: $\langle -(A_\psi,\ P_\phi,\ i,\ m,\ c), +(?,\ ?,\ \_,\ m,\ \perp)\rangle$ where $P_\phi \in \mathcal{I}^{pen}$, $A_\psi \in \mathcal{I}^{reg}$.

Note that the side conditions $B_\phi \in \mathcal{I}^{reg}$ and $A_\psi \in \mathcal{I}^{reg}$ in the above avoid redundancies caused by the penetrator sending a message to himself, but do not otherwise restrict him.

Observe that SD strands allow the penetrator to send messages to regular strands from the client end of a unilateral TLS connection, and to claim to be some honest agent A within the application-layer message, e.g. $+(?_\psi,\ B_\phi,\ A\hat{}\ldots,\ TLS^{C\to S})$. However, from the server end of a unilateral TLS (or bilateral TLS) connection, he has to use a penetrator identity $P \in \mathcal{T}^{pen}_{names}$ such that $P \neq ?$, e.g. $+(P_\psi,\ ?_\phi,\ m,\ TLS^{S\to C})$, so couldn't claim to have an identity other than P within m. Similarly RV strands allow the penetrator to receive, at the client end, messages intended for regular agents, e.g. $-(A_\psi,\ ?_\phi,\ B\hat{}\ldots,\ TLS^{S\to C})$. But at the server end, such a message would have to have a penetrator identity as the recipient to allow a RV strand, e.g. $-(?,\ P_\phi,\ P\hat{}\ldots,\ TLS^{C\to S})$.

[2]We abbreviate $?_?$ to $?$, for simplicity of notation.

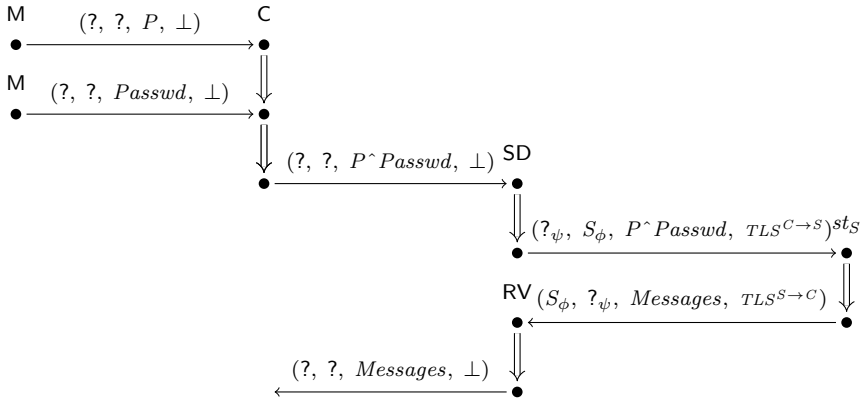


Figure 3.2: A figure illustrating how the penetrator can check his email, using the same protocol as Figure 3.1.

Further, the definition captures session properties of TLS-like protocols. The condition $P_\psi \in \mathcal{I}^{pen}$ ensures that if a regular strand receives two messages $(?_\psi,\ B_\phi,\ m,\ c)$ and $(?_\psi,\ B_\phi,\ m',\ c)$, either both came from another regular strand (if $?_\psi \in \mathcal{I}^{reg}$), or both come from penetrator SD strands (if $?_\psi \in \mathcal{I}^{pen}$). Thus we claim that this model accurately captures the penetrator's capabilities when using TLS-like protocols.

Figure 3.2 gives an example illustrating how the penetrator can use these strands to check his email, using the same hypothetical protocol as earlier.

Generalising the secure transport protocol

For secure transport protocols that are weaker than TLS, there are many other ways for the penetrator to interact with transport messages. We consider how to generalise the above in order to model such protocols. In particular, we follow the approach taken in [DL08, KL09] to define a more general penetrator that can also:

- Learn a message sent to an honest endpoint (i.e. overhear);
- Fake a message as coming from an honest endpoint;
- Hijack a message, redirecting it to a different endpoint, and/or re-ascribing it as coming from a different endpoint (changing name and/or channel end);
- Alter the sequence numbers of messages (i.e. reorder messages).

Later, we will restrict the use of such strands to channels that do not provide confidentiality (in the case of learning) or authentication (in the case of faking, hijacking or reordering). Formally, we exclude penetrator strands from the strand spaces according to the guarantees it provides (noting that this introduces a proof obligation on the transport protocol). For example, in the case of bilateral TLS we exclude all of the above penetrator strands from the strand space.

We define strands corresponding to each of the above behaviours below.

Definition 3.8 (Based on [KL09])**.** The *transport-layer penetrator strands* are SD and RV strands, as above, and strands of the following forms, where $c \neq \bot$:

LN Learn: $\langle -(A_\psi,\ B_\phi,\ i,\ m,\ c), +(?,\ ?,\ \_,\ m,\ \bot)\rangle$ where $A_\psi, B_\phi \in \mathcal{I}^{reg}$;

FK Fake: $\langle -(?,\ ?,\ \_,\ m,\ \bot), +(A_\psi,\ B_\phi,\ i,\ m,\ c)\rangle$ where $A_\psi, B_\phi \in \mathcal{I}^{reg}$, $i \in \mathcal{S}$;

HJ Hijack: $\langle -(S_\psi,\ R_\phi,\ i,\ m,\ c), +(S'_{\psi'},\ R'_{\phi'},\ i,\ m,\ c)\rangle$ providing either $S_\psi \neq S'_{\psi'}$ or $R_\phi \neq R'_{\phi'}$;

RN Renumber: $\langle -(A_\psi,\ B_\phi,\ i,\ m,\ c), +(A_\psi,\ B_\phi,\ i',\ m,\ c)\rangle$ where $i' \in \mathcal{S}$ and $i' \neq i$.

Note that by Assumption 3.3 it is not possible for a HJ strand to change a message that uses ? as, for example, the sending channel end to one that uses a value $\neq ?$.

Having defined the set of penetrator strands, we lift the usual definitions of penetrator paths, penetrator nodes and bundle equivalence from the low-level strand space definitions in the obvious way.

The restrictions on ? in HJ strands prevent a unilateral protocol from being transformed into a bilateral protocol, or vice-versa. Similarly, the restriction on _ in RN strands prevents protocols that do not have sequence numbers being transformed into those that do, and vice-versa.

As an example of how these strands can be used by the penetrator consider the following protocol that allows a user to send a message to another user via a trusted server.

$$1.\, U_\psi \to S_\phi : A$$
$$2.\, U_\psi \to S_\phi : m$$
$$3.\, S_\phi \to A_\chi : m$$

There are a number of attacks against this protocol, even when it is layered on top of a reasonably secure transport protocol. For example, suppose the secure transport protocol provides a guarantee of confidentiality, but does allow some hijacks. Further, suppose that the user attempts to send two messages, m_1 and m_2, to two different users $P \in \mathcal{T}^{pen}_{names}$ and $B \in \mathcal{T}^{reg}_{names}$ over two separate connections. It follows that the penetrator can take message 1 of connection 1 and change the sending channel endpoint to match that of connection 2 so that it appears as message 1 of connection 2. Therefore, m_2 will be routed to the penetrator which could potentially be a breach of confidentiality. Figure 3.3 illustrates this attack.

Note that even if the protocol disallows HJ strands it is still possible to find attacks against a slightly generalised version of the protocol. For example, suppose we extend the protocol so that it allows a user to send multiple messages, each to a

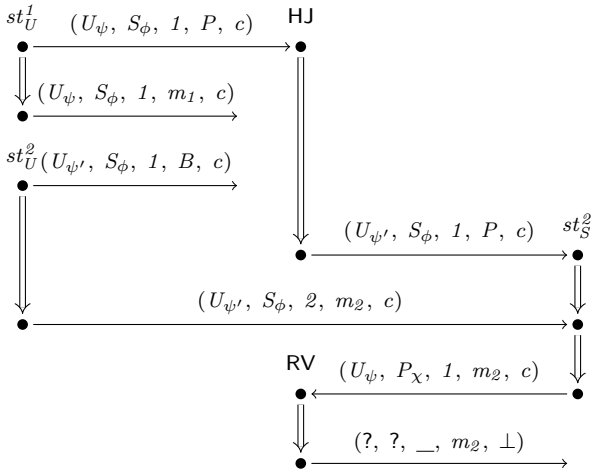


Figure 3.3: A graphical representation of a re-ascribe hijack attack by the penetrator, assuming that $P \in \mathcal{T}_{names}^{pen}$ but $B \in \mathcal{T}_{names}^{reg}$. The above RV strand should not be allowed as the intended recipient was B, who is honest.

different recipient, over one connection:

$$
\begin{aligned}
&1.U_\psi \rightarrow S_\phi : P \\
&2.U_\psi \rightarrow S_\phi : m_1 \\
&3.S_\phi \rightarrow P_\chi : m_1 \\
&4.U_\psi \rightarrow S_\phi : B \\
&5.U_\psi \rightarrow S_\phi : m_2 \\
&6.S_\phi \rightarrow B_{\chi'} : m_2 \\
&\quad \ldots
\end{aligned}
$$

Suppose the user attempts to send two messages, m_1 and m_2, to two different users, $P \in \mathcal{T}_{names}^{pen}$ and $A \notin \mathcal{T}_{names}^{pen}$ over a channel that does allow renumbering. Clearly, the penetrator could copy, renumber and then re-send message 1 as message 4 and thus receive m_2, which he is not entitled to receive. This attack is illustrated in Figure 3.4.

Channel Properties

We now define channel properties that restrict the penetrator's behaviour. There are many different channel properties. We consider only the most important; other definitions could be made if specialised applications require them.

The first property we consider is *confidentiality*. Intuitively this needs to prohibit the penetrator from learning any value that was sent along a confidential channel to a regular agent. Clearly, this requires LN strands to be prohibited, however, this is not sufficient. For example, suppose A_ψ sends a message to B_ϕ along a confidential channel c; if the penetrator could redirect the message to $P_\chi \in \mathcal{I}^{pen}$, then he would be able to do a receive and thus can obtain the message indirectly; we therefore

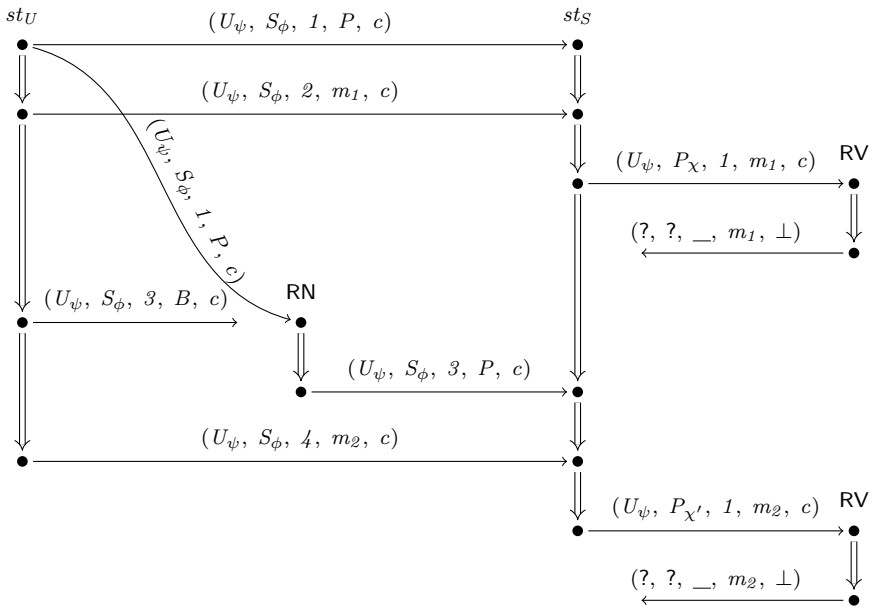


Figure 3.4: A graphical representation of a renumbering attack by the penetrator, assuming that $P \in \mathcal{T}^{pen}_{names}$ but $B \in \mathcal{T}^{reg}_{names}$. The second RV strand should not be allowed as the intended recipient was B, who is honest.

prohibit such behaviours. Lastly, renumbering has to be prohibited since, as was shown in Figure 3.4, it can be used to break confidentiality.

Definition 3.9 (Confidential)**.** Let channel c satisfy $\mathcal{C}$. Then the strand space contains no LN or RN strands on c, or HJ strands of the form $\langle -(S_\psi,\ R_\phi,\ m,\ c),\ +(S'_{\psi'},\ R'_{\phi'},\ m,\ c)\rangle$ where $R_\phi \neq R'_{\phi'}$ and $R_\phi \in \mathcal{I}^{reg}$.

Note that the above definition also prohibits changing the recipient from Y_ψ to $Y_{\psi'}$ where $Y \in \mathcal{T}^{reg}_{names}$ and $\psi, \psi' \in \mathcal{C}^{reg}$, i.e. it prevents the receiving channel end from being changed, even if it is still being received by the same entity. Intuitively this can be justified by considering that a single agent may be running multiple services that hold data in different levels of security. If the agent also broadcasts data that was received at a lower security level, then it is essential that data cannot be re-routed to different services and thus different channel ends.

Many application-layer protocols require some guarantee that messages came from a certain source and that they were intended for a particular destination, i.e. that the channel is an *authenticated* one. Clearly, this means that FK strands must be prohibited on this channel. Furthermore, hijacks must also be prohibited since these allow messages to be sent to unintended destinations and for messages to have incorrect purported senders respectively.

Definition 3.10 (Authenticated)**.** Let channel c satisfy $\mathcal{A}$. Then the strand space contains no FK or HJ strands on c.

The conjunction of the $\mathcal{A}$ and the $\mathcal{C}$ properties is known as $\mathcal{AC}$. Note that, the only penetrator strands allowed are SD and RV strands and therefore, as discussed

earlier, it corresponds to the security guarantees modelled by TLS[3].

3.1.3 High-Level Normality

As with the low-level strand spaces model, there are a large number of ways for the penetrator to construct and send a given message. Such redundancies have the potential to complicate proofs and therefore, as in the low-level strand spaces model, we will require a *normal* form for high-level bundles that avoids various redundancies.

In particular, we will require not only that bundles contain no low-level redundancies, as per Definition 2.16, but also that there are no redundancies amongst the high-level strands. For example, multiple adjacent RN or HJ strands will be prohibited, as will SD or FK strands which are followed by RV or LN strands. In this section we define what it means for a bundle to be normal and then prove that every bundle is equivalent to a normal bundle.

We proceed as per the low-level case, and begin by defining what it means for $\Rightarrow^+$-edges to be *constructive* and *destructive*. This definition is then used to define several types of penetrator paths, which are then used to define what it means for a bundle to be normal. We formalise these notions as follows.

Definition 3.11. A $\Rightarrow^+$-edge is *constructive* if it lies on a E, C, KG, SD or FK strand, *destructive* if it lies on a D, S, RV or LN strand, or *normal* if it lies on a HJ or RN strand.

A penetrator path p in a high-level bundle $\mathcal{B}$ is *constructive* iff it consists entirely of constructive edges, *destructive* iff it consists entirely of destructive edges, or *normal* iff either:

- p is a single HJ or RN strand; or
- p is the concatenation of a HJ and RN strand, in no specified order; or
- Whenever a constructive edge sends to a destructive edge in p, the constructive edge must be a KG strand..

A high-level bundle $\mathcal{B}$ is *normal* iff every penetrator path that does not traverse a key edge is normal.

In order to prove that every high-level bundle is equivalent to a normal bundle, we will require several closure-like assumptions on what transport strands the strand space contains. For example, by the above definition, penetrator paths that consist of a SD strand followed by a HJ strand are prohibited. Thus, in order to prove our result we need to replace this by a single FK or SD strand, that sends the message to the correct agent immediately. Therefore, we need to ensure that whenever the strand space contains the SD-HJ strand pair, it also allows the equivalent FK or SD strand. More generally, we need a number of (reasonable) assumptions about what strands the strand space has to allow, given that it allows a particular strand or pair of strands. In general, the following assumptions are justified by observing that the behaviour performed is identical.

[3]Of course this requires a proof that TLS does indeed satisfy the claimed properties. Such a proof can be found in [KL11].

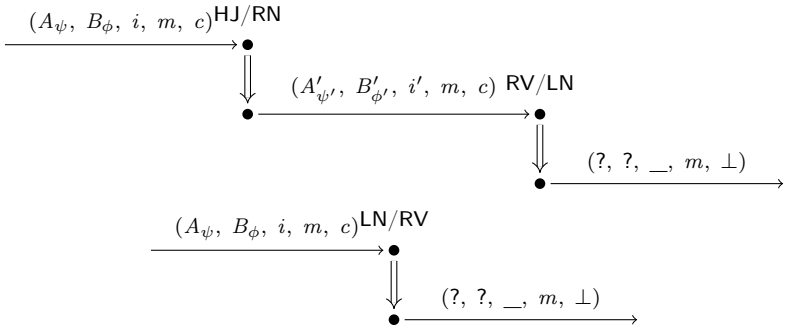


Figure 3.5: An illustration of one of the transformations of Lemma 3.13. This also illustrates Assumption 3.12 (2).

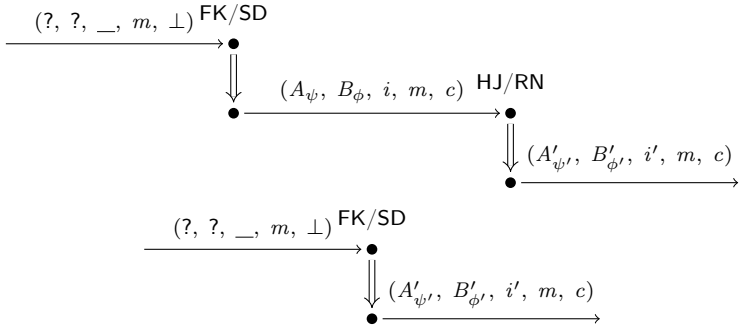


Figure 3.6: An illustration of one of the transformations of Lemma 3.13. This also illustrates Assumption 3.12 (3).

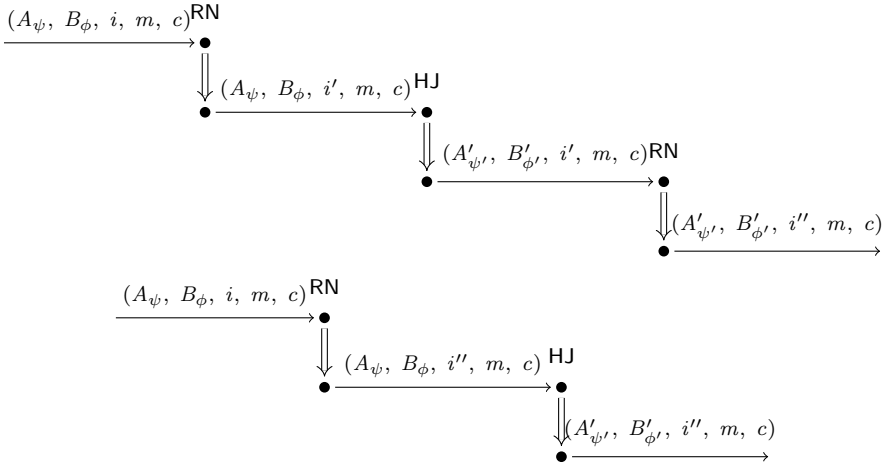


Figure 3.7: An illustration of one of the transformations of Lemma 3.13.

Assumption 3.12. Σ satisfies the following properties:

1. SD, FK, RV, LN and HJ strands are not disallowed on the basis of sequence numbers. Formally, if $st_i = \langle (A_\psi,\ B_\phi,\ i,\ m,\ c), (A'_{\psi'},\ B'_{\phi'},\ i,\ m',\ c') \rangle$ is such a strand, then whenever st_i is in Σ and $i \neq \_$, $st_{i'}$ is in Σ for all $i' \in \mathbb{N}^+$.

2. If a HJ strand followed by a RV or LN strand is in the strand space then the equivalent RV or LN strand is also in the strand space. This is illustrated in Figure 3.5.

3. If a SD or FK strand followed by a HJ strand is in the strand space then the equivalent SD or FK strand is also in the strand space. This is illustrated in Figure 3.6.

4. If multiple adjacent HJ strands are in the strand space then then the equivalent singleton HJ strand is also in the strand space.

5. Suppose $st_{i'}$ is a RN strand that alters a sequence number from i to i'. If st_i is in the strand space then $st_{i'}$ is in the strand space for each $i' \in \mathbb{N}^+$.

6. If a HJ followed by a RN followed by a second HJ is in the strand space, then an equivalent HJ-RN or RN-HJ strand pair must also be in the strand space.

This assumption is sufficiently general to permit all sensible channel types: in particular, the strands that are removed from the strand space by $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{AC}$ are compatible with the above definition. Note that if a channel type that did not satisfy the above assumption then, by altering what it means for a bundle to be normal, it may be possible to permit it.

Using the above we can now show that every bundle is equivalent to a normal bundle. This is used throughout the remainder of this thesis in order to simplify proofs.

Lemma 3.13. For every high-level bundle $\mathcal{B}$ there exists an equivalent normal bundle $\mathcal{B}'$.

Proof. Let $\mathcal{B}$ be a high-level bundle. Suppose that a given penetrator path p is not normal. We show how to transform p to a penetrator path that is normal by applying the following transformations inductively, noting that each preserves equivalence. Termination follows from the fact that each case strictly decreases the size of the penetrator path.

Note that the following cases are applied sequentially (i.e. case *i + 1* is considered only if case i does not apply).

1. p contains both normal and non-normal (i.e. constructive or destructive) edges. Thus, there must exist a transmission edge between a non-normal edge and a normal edge. There are two possible types of such transmission edges, noting that the normal edge must lie on either a RN or HJ strand:

   (a) The transmission edge is from a non-normal edge to a normal edge. Hence, the non-normal edge must lie on a FK or SD strand. Thus, we apply the transformation from Figure 3.6 (formally, this uses Assumption 3.12 (1) if the normal edge is on a RN strand, or Assumption 3.12 (3) if it lies on a HJ strand).

(b) The transmission edge is from a normal edge to non-normal edge. Thus, the non-normal edge must either lie on a RV or a LN strand. Therefore, the transformation from Figure 3.5 is applied (formally, this uses Assumption 3.12 (1) if the normal edge lies on a RN strand, or Assumption 3.12 (2) if it lies on a HJ strand)

2. p contains multiple normal edges and no non-normal edges. It thus follows that p contains only RN and HJ strands. There are several cases to consider:

   (a) If p contains adjacent RN strands then they can be combined into a single RN strand, by Assumption 3.12 (5).

   (b) If p contains adjacent HJ strands they they can be combined into a single HJ strand, by Assumption 3.12 (4).

   (c) p contains no adjacent HJ or RN strands, but contains a RN strand, followed by a HJ strand, followed by a RN strand. We therefore apply the transformation illustrated in Figure 3.7. Formally, by Assumption 3.12 (5), the first RN strand can be altered to change the sequence number to the output of the second RN strand, meaning the second RN strand can be removed. Further, by Assumption 3.12 (1), the HJ strand is still within the strand space.

   (d) Otherwise, p must consist of a HJ, then a RN, then a HJ. Thus, by Assumption 3.12 (6), this is equivalent to the concatenation of a HJ and RN strand (in some order).

3. p contains a destructive edge after a constructive edge and the constructive edge is not a KG strand. As p contains no normal edges it follows that there must exist n_1, n_1', n_2 and n_2' such that $n_1 \Rightarrow^+ n_1' \rightarrow n_2 \Rightarrow^+ n_2'$ form such a pair of edges. Given the type restrictions on penetrator strands, it follows that such a pair must consist of a E-D, C-S, SD-LN or a FK-RV strand pair. In all cases, the redundant pair can simply be removed in the obvious way (Figure 2.2 illustrates how redundant C-S and E-D strand pairs can be eliminated). This results in a penetrator path that is strictly shorter.

Thus, the above rules can be applied inductively to each non-normal penetrator path that does not traverse a key edge until a normal bundle is reached, which is defined as $\hat{\mathcal{B}}'$. This approach is guaranteed to terminate as each case strictly reduces the size of the bundle. Note that $\hat{\mathcal{B}}'$ must be equivalent to $\hat{\mathcal{B}}$ as no regular nodes are altered by the above transformations. □

3.2 Verifying Layered Protocols

In this section we give proof rules that are of use when proving the correctness of application-layer protocols that use either unilateral or bilateral secure transport protocols that provide authenticated or confidential channels. We have also developed a prototype tool that is able to prove the correctness of application-layer protocols by applying the proof rules from this section, as we discuss in Section 9.1. In Section 3.3

we show the effectiveness of these rules by proving the correctness of a single-sign-on protocol.

The first proof rule allows the existence of a regular node to be deduced given a message that is purported to have been sent by a regular agent.

Authentication Proof Rule. Let $\mathcal{B}$ be a high-level bundle and $n \in \mathcal{B}$ be a node on a regular strand st such that $msg(n) = -(A_\psi,\ B_\phi,\ m,\ c)$ for some $A_\psi \neq ?_?$, B_ϕ, m and c. Then, providing c satisfies $\mathcal{A}$, and $A_\psi \in \mathcal{I}^{reg}$, there must exist a regular node n' such that $n' \rightarrow n$ and $msg(n') = +(A_\psi,\ B_\phi,\ m,\ c)$. Furthermore, if $\phi \neq ?$ and $n' \rightarrow n''$ then n'' must lie on the same strand as n.

Proof. Consider the node n' such that $n' \rightarrow n$ and suppose for a contradiction that n' is a penetrator node; then as $A_\psi \in \mathcal{I}^{reg}$ it follows that the only type of penetrator strand that n' could be on is a FK strand. However, these are prohibited by the assumption that c satisfies $\mathcal{A}$. Therefore n' is a regular node. Thus, as $n' \rightarrow n$, it immediately follows that $msg(n') = +(A_\psi,\ B_\phi,\ m,\ c)$.

Let n'' be a node on a strand st'' such that $n' \rightarrow n''$ and suppose $\phi \neq ?$. Then, it immediately follows that $msg(n'') = -(A_\psi,\ B_\phi,\ m,\ c)$ and thus that $\phi \in ends(st'')$. However, as $\phi \in ends(st)$, it follows by Equation 3.1 that $st'' = st$, and thus that n'' lies on the same strand as n, as required. □

The second proof rule extends the above rule by not only proving the existence of the regular node, but also proving that it lies on the same strand as another regular node (providing the channel ends match).

Session Proof Rule. Let $\mathcal{B}$ be a high-level bundle and $n \in \mathcal{B}$ be a node on a regular strand st such that $msg(n) = -(A_\psi,\ B_\phi,\ m,\ c)$ for some $A_\psi \in \mathcal{I}^{reg}$, B_ϕ, m and c. Further, let st' be a regular strand such that $A_\psi \in endpoints(st')$. Then, providing c satisfies at least $\mathcal{A}$ and $\psi \neq ?$ there must exist a regular node n' on st' such that $n' \rightarrow n$.

Proof. This follows from the previous rule by Equation 3.1. □

The third proof rule is mainly applicable to unilaterally authenticating secure transport channels. Informally, it proves that if a term t is only known by regular agents, then any node whose application-layer message includes t at the top level (i.e. not inside an encryption) must be regular. In order to define this property we firstly define what it means for a term to be known only by regular agents, as follows.

Definition 3.14. A term t is *confidential* in a high-level bundle $\mathcal{B}$ iff no equivalent bundle contains a positive node n such that $msg(n) = +(?,\ ?,\ \_,\ t,\ \bot)$.

If the above definition holds then it follows it is impossible for the penetrator to obtain the term t in any way. Note that we quantify over all equivalent bundles in order to ensure we consider all possible ways in which the penetrator could possibly obtain the value, without altering the regular behaviour.

Using the above we can now state and prove the proof rule as follows.

Authentication via Confidentiality Proof Rule. Let $\mathcal{B}$ be a bundle and t be a confidential term in $\mathcal{B}$. Further, let $n \in \mathcal{B}$ be a regular node such that $msg(n) = -(A_\psi,\ B_\phi,\ i,\ m,\ c)$ for some A_ψ, B_ϕ, i, m and c, such that $m = \ldots \hat{}\, t \,\hat{}\ldots$, and c satisfies $\mathcal{AC}$. Then there exists a regular node $n' \in \mathcal{B}$ such that $n' \rightarrow n$.

Proof. Let n' be the node such that $n' \rightarrow n$. Suppose, for a contradiction, that n' is a penetrator node. Then as c satisfies $\mathcal{AC}$ it follows that n' must lie on a SD strand and thus there must exist nodes n_1 and n_2 such that $n_2 \rightarrow n_1 \Rightarrow n'$ and $msg(n_2) = +(?,\ ?,\ m,\ \bot)$. Hence there exists an equivalent bundle $\mathcal{B}'$ that contains the same regular nodes, together with extra S strands to extract t from m to obtain a node n'' such that $msg(n'') = (?,\ ?,\ t,\ \bot)$. However, this contradicts the fact that t is confidential and therefore n' must be a regular node. □

The premises of the above lemma can be weakened to only require the channel to satisfy $\mathcal{C}$ (rather than $\mathcal{AC}$) in return for weakening the conclusion to only prove that there is some penetrator *path* between n' and n.

In order to use the above proof rule we need to be able to prove that certain terms are confidential. In [Kam10] Kamil showed that a useful class of atoms known as *safe* atoms are confidential, which we now adapt slightly.

Firstly, we define the set of terms *deducible by the penetrator*, denoted $\mathcal{A}^*_{\mathcal{P}}$. This consists of not only all terms that the penetrator initially possesses, but also those he may be able to extract from these later on. For example, if $\{\!|m|\!\}_k \in \mathcal{A}_{\mathcal{P}}$ then $m \in \mathcal{A}^*_{\mathcal{P}}$ as m may be deducible by the penetrator later on, if he obtains k^{-1}.

Definition 3.15. The set of terms *deducible by the penetrator*, denoted $\mathcal{A}^*_{\mathcal{P}}$, is defined as $\{t' \mid t \in \mathcal{A}_{\mathcal{P}} \wedge t' \sqsubseteq t\}$.

In order to define what it means for an atom to be safe, we firstly define under what conditions a term sent in a high-level message can only be received by a regular nodes.

Definition 3.16 (From [Kam10])**.** Let $\mathcal{B}$ be a high-level bundle. A term t is *sent confidentially* in a high-level term $(A_\psi,\ B_\phi,\ m,\ c)$ iff $t \sqsubseteq m$, c satisfies $\mathcal{C}$, and $A_\psi, B_\phi \notin \mathcal{I}^{pen}$.

We now define what it means for an atom to be safe. This definition is based on the definition given in [Kam10], but it has been altered to permit it to be used more simply[4].

Definition 3.17 (Based on [Kam10])**.** The set of *safe* atoms in a high-level bundle $\mathcal{B}$ is defined inductively by $\mathcal{M}(\mathcal{B}) = \bigcup_i \mathcal{M}_i(\mathcal{B})$ where:

- $a \in \mathcal{M}_0(\mathcal{B})$ iff $a \notin \mathcal{A}^*_{\mathcal{P}}$, a is not complex and, for all positive regular nodes $n \in \mathcal{N}_{\mathcal{B}}$, if $a \sqsubseteq appmsg(n)$, then a is sent confidentially in $msg(n)$.
- $\mathcal{M}_{i+1}(\mathcal{B}) = \mathcal{M}_i(\mathcal{B}) \cup X_{i+1}(\mathcal{B})$ where $a \in X_{i+1}(\mathcal{B})$ iff $a \notin \mathcal{A}^*_{\mathcal{P}}$, a is not complex and, for all positive regular nodes $n \in \mathcal{N}_{\mathcal{B}}$, if $a \sqsubseteq appmsg(n)$, then either a is sent confidentially in $msg(n)$ or a occurs only within the set of terms $\{\{\!|t|\!\}_k \mid t \in \mathcal{A} \wedge k^{-1} \in \mathcal{M}_i(\mathcal{B})\}$ in $msg(n)$.

We say that r *occurs safely in* $\mathcal{B}$ iff $r \in \mathcal{M}(\mathcal{B})$.

We can now prove that whenever t is a safe atom, t must be confidential in $\mathcal{B}$. This proof is based on that in [Kam10], but has been altered to match the new definitions above.

[4]In particular, [Kam10] insisted that if t is safe then t is always sent confidentially. We weaken this condition to only require t to be sent confidentially by regular nodes.

Lemma 3.18 (Based on [KL09])**.** Let $\mathcal{B}$ be a bundle. If t a safe atom in $\mathcal{B}$ then t is confidential in $\mathcal{B}$.

Proof. Let $\mathcal{B}$ be a bundle and t a safe atom in $\mathcal{B}$. Since t is safe in $\mathcal{B}$, it follows that $t \in \mathcal{M}_i(\mathcal{B})$ for some i. We prove the result by induction on i.

If $i = 0$, the it follows that $t \notin \mathcal{A}^*_{\mathcal{P}}$, t is not complex and, for all positive regular nodes $n \in \mathcal{N}_{\mathcal{B}}$, if $a \sqsubseteq \mathit{appmsg}(n)$, then a is sent confidentially in $\mathit{msg}(n)$. Suppose, for a contradiction, that t is not confidential in $\mathcal{B}$. It therefore follows that there exists an equivalent bundle $\mathcal{B}'$ that contains a positive node n such that $\mathit{msg}(n) = (?,\ ?,\ t,\ \bot)$. n cannot be regular since t is sent confidentially in $\mathcal{B}$ and therefore in $\mathcal{B}'$ (as the regular nodes of the bundles are the same). Thus, n must be a penetrator node. Let $n' \preceq n$ be the positive node at which t originates via a penetrator path p. Note that n' cannot be a penetrator node, since, by assumption, $t \notin \mathcal{A}^*_{\mathcal{P}}$ and t is not a complex key. Thus, n' must be a regular node. Consider the node n'' on p such that $n' \rightarrow n''$. Since a is sent confidentially in $\mathit{msg}(n')$, by assumption, it follows that n'' is a regular node since $\mathit{recipient}(n') \notin \mathcal{I}^{pen}$ and $\mathit{chan}(n')$ satisfies $\mathcal{C}$, by Definition 3.16. Therefore, by a trivial induction, it follows that all nodes along p must be regular, contradicting the fact that n is a penetrator node. Thus, t is confidential in $\mathcal{B}$.

For the inductive case, suppose that all $a \in \mathcal{M}_i(\mathcal{B})$ are confidential in $\mathcal{B}$. We prove that all $a \in \mathcal{M}_{i+1}(\mathcal{B})$ are confidential in $\mathcal{B}$. By definition of $\mathcal{M}_{i+1}(\mathcal{B})$, either $a \in \mathcal{M}_i(\mathcal{B})$ or $a \in X_{i+1}(\mathcal{B})$. In the former case, the required result immediately follows by the inductive hypothesis. Otherwise, suppose $a \in X_{i+1}(\mathcal{B})$. It therefore follows that $a \notin \mathcal{A}^*_{\mathcal{P}}$, a is not complex and, for all positive regular nodes $n \in \mathcal{N}_{\mathcal{B}}$, if $a \sqsubseteq \mathit{appmsg}(n)$, then either a is sent confidentially in $\mathit{msg}(n)$ or a occurs only within the set of terms $\{\{\!|t|\!\}_k \mid t \in \mathcal{A} \wedge k^{-1} \in \mathcal{M}_i(\mathcal{B})\}$ in m. Again, suppose, for a contradiction, that a is not confidential in $\mathcal{B}$. It therefore follows that there exists an equivalent bundle $\mathcal{B}'$ that contains a positive node n such that $\mathit{msg}(n) = (?,\ ?,\ a,\ \bot)$. n cannot be regular since, by assumption, a is sent confidentially within $\mathcal{B}$ from positive regular nodes (noting that the second branch of X_{i+1} cannot apply, since $\mathit{appmsg}(n) = a$). Thus, n must be a penetrator node. Let n' be the last positive regular node on p, noting that such a node has to exist since a must originate at on regular node, since $a \notin \mathcal{A}^*_{\mathcal{P}}$ and a is not complex. By assumption, there are two cases to consider:

- n' sends a confidentially in $\mathit{msg}(n')$. However, since $\mathit{chan}(n')$ satisfies $\mathcal{C}$ and $\mathit{recipient}(n) \notin \mathcal{I}^{pen}$, it immediately follows that any node n'' such that $n' \rightarrow n''$ must also be regular, contradicting the fact that n' is the last such node.

- a occurs only within the set of terms $\{\{\!|t|\!\}_k \mid t \in \mathcal{A} \wedge k^{-1} \in \mathcal{M}_i(\mathcal{B})\}$ in $\mathit{msg}(n')$. Hence, the penetrator must extract a from its enclosing encryptions and therefore, must be able to obtain k^{-1} for some $\{\!|m|\!\}_k$ that encloses a in $\mathit{msg}(n)$. However, such an inverse key is confidential by assumption, and therefore there cannot exist a node n'' such that $\mathit{msg}(n'') = (?,\ ?,\ k^{-1},\ \bot)$, as required.

Thus, since we derive a contradiction in either case, it follows that a is confidential in $\mathcal{B}$, as required. □

The fourth proof rule is based on a rule from [GT02]. In [GT02] Guttman and Thayer develop the notion of *authentication tests* as a method of proving the correctness of the authentication goals of security protocols. The *Unsolicited Authentication Test* allows the existence of a regular node to be deduced, given a message encrypted by a confidential key. The correctness of this rule can be observed by noting that the penetrator could not have performed the encryption since the key is confidential and thus unobtainable by the penetrator.

Unsolicited Authentication Test. Suppose that there is a negative node $n_1 \in \mathcal{B}$ such that $msg(n_1) = -(A_\psi,\ B_\phi,\ i,\ m,\ c),\ t = \{\!|t_0|\!\}_K \sqsubseteq m$ and K is confidential in $\mathcal{B}$. Providing $t \notin \mathcal{A}^*_\mathcal{P}$ then there exists a regular node $n'_1 \prec_\mathcal{B} n_1$ such that t originates on n'_1.

The last proof rule allows RN strands to be ignored under certain circumstances. This can be deduced by observing that the model above allows three different types of transport layer protocols to be differentiated:

1. Protocols that do not have a concept of sequence numbers;
2. Protocols that do have sequence numbers, but allow them to be altered;
3. Protocols that do have sequence numbers and do not allow them to be altered.

Under the assumption that the application layer protocol does not have direct access to the sequence numbers, it follows that any attack against an application layer protocol using a transport layer protocol of the second type could also be found by considering the transport layer protocol as one of the first type. This is formalised in the following Lemma.

Lemma 3.19. Let $\mathcal{B}$ be a high-level bundle containing RN strands over a channel c. Then there exists a high-level bundle $\mathcal{B}'$ containing no RN strands on c and where each high-level term $(A_\psi,\ B_\phi,\ i,\ m,\ c)$ has been replaced by $(A_\psi,\ B_\phi,\ \_,\ m,\ c)$ (i.e. $\mathcal{B}$ and $\mathcal{B}'$ are equivalent, modulo sequence numbers).

Proof. Let $\mathcal{B}$ be such a bundle; we construct the bundle $\mathcal{B}'$ directly as follows. The nodes of $\mathcal{B}'$ are those of $\mathcal{B}$, but with each term $(A_\psi,\ B_\phi,\ i,\ m,\ c)$ replaced by $(A_\psi,\ B_\phi,\ \_,\ m,\ c)$. Further, each RN strand is replaced by a direct message send strand. Clearly this bundle satisfies the bundle conditions as the bundle structure has not been altered. Further, all the malformed RN strands that resulted from applying the term substitution have been removed. □

3.3 Example: The WebAuth Protocol

WebAuth [SA09] is a *single-sign on protocol* that is designed to allow users to login to multiple websites through a central authentication server, meaning that only one username and password per user is required. It differs from other single-sign on protocols, such as OpenID [FRH$^+$], in that it requires shared keys to be established between the website and the authentication server prior to authentication of users. We prove the correctness of WebAuth by proving three propositions that show what each principal can infer having completed a run. Our analysis reveals a subtlety

concerning the strength of authentication guarantees to the application server, and a requirement on the user that may not be obvious to all users.

In this section we firstly introduce the WebAuth protocol and describe some of the unusual issues that arise when verifying web-based protocols. We then formally define a strand space for the WebAuth protocol, state the assumptions that are required, and present the proofs of correctness.

Three principals participate in a WebAuth session: the *User Agent* (UA) is the web browser that makes requests for the user; the *Application Server* (AS) is the server that the user wishes to access; the *Login Server* (LS) is the server responsible for authenticating the user. These principals communicate via HTTP [LBLM+04] or HTTPS (i.e. HTTP over TLS) [Res00] requests, and pass data to each other by embedding *tokens* in the redirect URLs. For example, when the AS redirects the UA to the LS, the redirect URL will be of the form `https://`*LS*`/?RT=`*rtok*`;ST=`*stok* where *rtok* and *stok* are tokens. The AS and LS also use HTTP cookies to store tokens to allow the user agent to re-authenticate on subsequent requests.

3.3.1 The Protocol

In this paper we prove the correctness of the *initial sign on* mode of WebAuth, which assumes that the user is not already authenticated. The protocol in its most simplified form is as follows:

$$
\begin{array}{lll}
1. & UA \rightarrow AS & : Request \\
2. & AS \rightarrow UA & : RequestToken \hat{} ServiceToken \hat{} LS \\
3. & UA \rightarrow LS & : RequestToken \hat{} ServiceToken \\
4. & LS \rightarrow UA & : LoginForm \hat{} RequestToken \hat{} ServiceToken \\
5. & UA \rightarrow LS & : RequestToken \hat{} ServiceToken \hat{} U \hat{} passwd_{LS}(U) \\
6. & LS \rightarrow UA & : AS \hat{} Request \hat{} ProxyToken \hat{} IdToken \\
7. & UA \rightarrow AS & : Request \hat{} IdToken \\
8. & AS \rightarrow UA & : Response \hat{} AppToken
\end{array}
$$

In the above the user first sends a request to the application server. Since the user is unauthenticated, the application server redirects the user to the login server, who sends the user the login form to complete. The user then completes this form and the username and password are sent back to the login server. Assuming the username and password are correct, the login server issues the user with a token that can be used to authenticate the user to the application server. The user thus passes this token back to the application server and receives the content they requested.

A *RequestToken* encapsulates the original request that the user made. The *ServiceToken* contains configuration information for the LS, enabling it to be stateless. The *ProxyToken* allows a user to authenticate again without supplying her password (i.e. *repeat authentication*), whilst the *IdToken* is a temporary token created by the LS for the AS that details who the user is. The user exchanges this temporary token for an *AppToken* by passing it to the AS.

WebAuth's token encoding is complicated, so we use a simplified version; essentially the same proof would hold for the full protocol. We encode a token by $\{|\mathsf{tag} \hat{} data|\}_{key}$ where $key \in \mathcal{K}_{sym}$ and tag is a tag. The protocol can be described as follows, where $SK(A)$ denotes a symmetric key that is secret to $A \in \mathcal{T}_{names}$ and

Sh_{LS}^{AS} denotes the key shared between AS and LS.

1. $UA \rightarrow AS$: r
2. $AS \rightarrow UA$: $LS \hat{} \{|\mathsf{req} \hat{} r|\}_{Sh_{LS}^{AS}} \hat{} \{|\mathsf{webkdc\_service} \hat{} AS \hat{} Sh_{LS}^{AS}|\}_{SK(LS)}$
3. $UA \rightarrow LS$: $\{|\mathsf{req} \hat{} r|\}_{Sh_{LS}^{AS}} \hat{} \{|\mathsf{webkdc\_service} \hat{} AS \hat{} Sh_{LS}^{AS}|\}_{SK(LS)}$
4. $LS \rightarrow UA$: $LoginForm \hat{} \{|\mathsf{req} \hat{} r|\}_{Sh_{LS}^{AS}} \hat{} \{|\mathsf{webkdc\_service} \hat{} AS \hat{} Sh_{LS}^{AS}|\}_{SK(LS)}$
5. $UA \rightarrow LS$: $U \hat{} passwd_{LS}(U) \hat{} \{|\mathsf{req} \hat{} r|\}_{Sh_{LS}^{AS}}$ $\hat{} \{|\mathsf{webkdc\_service} \hat{} AS \hat{} Sh_{LS}^{AS}|\}_{SK(LS)}$
6. $LS \rightarrow UA$: $AS \hat{} r \hat{} \{|\mathsf{webkdc\_proxy} \hat{} U|\}_{SK(LS)} \hat{} \{|\mathsf{id} \hat{} U|\}_{Sh_{LS}^{AS}}$
7. $UA \rightarrow AS$: $r \hat{} \{|\mathsf{id} \hat{} U|\}_{Sh_{LS}^{AS}}$
8. $AS \rightarrow UA$: $resp \hat{} \{|\mathsf{app} \hat{} U \hat{} Sh_{LS}^{AS}|\}_{SK(AS)}$

WebAuth mandates the use of HTTPS between LS and AS, and between LS and UA, but merely *recommends* that HTTPS is used between UA and AS. Clearly, if HTTPS is not used between UA and AS then there are a number of attacks whereby the intruder intercepts various tokens. For example, the penetrator could intercept the app token and pose as the user in subsequent requests to AS. Thus, we assume that all messages are sent over unilateral TLS, with UA unauthenticated.

When modelling security protocols it is generally assumed that the participants are able to perform checks on the values they receive to ensure adherence to the protocol. For example, a UA may be expected to compare the value of r received in message 6 to the one sent in message 1. However, these checks are not possible if the role is being assumed by a general-purpose web browser. In particular this means that the user will not check if the request and the AS match between messages 1 and 7, or if the request and service tokens match. Therefore, when we formally define how a UA behaves we ensure that it does not check for agreement on any values. Further, the servers are *stateless*. For example, the AS stores no state between the first two messages and the last two; we will therefore model these two exchanges using two distinct strands (and similarly for the LS). Both of these factors contribute to complicating our analysis.

One problem that we do not consider whilst proving the correctness of WebAuth is that the web browser does not check if messages are skipped or reordered. For example, there is nothing to prevent a dishonest application server from sending a message 4 rather than a message 2 since there is no way for the web browser to detect this. In principle, it would be possible to adapt the proofs to model this behaviour, but the proofs would be rather intricate and uninteresting and therefore we do not consider this case.

3.3.2 Strand Space Definition

We now define the strand space corresponding to the protocol. Note that the proof of correctness of WebAuth does not depend on the ordering of the messages, and therefore we omit the sequence number component in the high-level terms. In the following:

- ψ_i denotes channel ends used by the user;
- ϕ_i denotes channel ends used by the login and application servers;

- r_i denotes requests;
- *stok*, *rtok*, *atok*, *ptok*, *idtok* denote service, request, application, proxy and identity tokens respectively.

Further, we assume that the set of atoms, $\mathcal{T}$, includes requests, responses, the token tags, passwords and the login form (denoted by *LoginForm*).

Definition 3.20. A *Web-Auth Strand Space* consists of the union of the images of the following functions.

$$
\begin{aligned}
&AS1[AS, LS, \psi_1, \phi_1, r_1, stok] \mathrel{\widehat{=}} && // \text{ Messages 1, 2}\\
&\quad \langle -(?_{\psi_1},\ AS_{\phi_1},\ r_1,\ {}_{TLS^{C\to S}}),+(AS_{\phi_1},\ ?_{\psi_1},\ \{\!|\mathsf{req}\hat{}\, r_1|\!\}_{Sh^{AS}_{LS}}\hat{}\, stok,\ {}_{TLS^{S\to C}})\rangle\\
&AS2[AS, U, LS, \psi_4, \phi_4, r_2, resp] \mathrel{\widehat{=}} && // \text{ Messages 7, 8}\\
&\quad \langle -(?_{\psi_4},\ AS_{\phi_4},\ r_2\hat{}\, \{\!|\mathsf{id}\hat{}\, U|\!\}_{Sh^{AS}_{LS}},\ {}_{TLS^{C\to S}}),\\
&\quad + (AS_{\phi_4},\ ?_{\psi_4},\ resp\hat{}\, \{\!|\mathsf{app}\hat{}\, U\hat{}\, Sh^{AS}_{LS}|\!\}_{SK(AS)},\ {}_{TLS^{S\to C}})\rangle\\
&LS1[LS, AS, \psi_2, \phi_2, k, r_2] \mathrel{\widehat{=}} && // \text{ Messages 3, 4}\\
&\quad \langle -(?_{\psi_2},\ LS_{\phi_2},\ \{\!|\mathsf{req}\hat{}\, r_2|\!\}_k\hat{}\, \{\!|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, k|\!\}_{SK(LS)},\ {}_{TLS^{C\to S}}),\\
&\quad + (LS_{\phi_2},\ ?_{\psi_2},\\
&\qquad LoginForm\hat{}\, \{\!|\mathsf{req}\hat{}\, r_2|\!\}_k\hat{}\, \{\!|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, k|\!\}_{SK(LS)},\ {}_{TLS^{S\to C}})\rangle\\
&LS2[LS, U, AS, \psi_3, \phi_3, k, r_2] \mathrel{\widehat{=}} && // \text{ Messages 5, 6}\\
&\quad \langle -(?_{\psi_3}, LS_{\phi_3},\\
&\qquad U\hat{}\, passwd_{LS}(U)\hat{}\, \{\!|\mathsf{req}\hat{}\, r_2|\!\}_k\hat{}\, \{\!|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, k|\!\}_{SK(LS)}, {}_{TLS^{C\to S}}),\\
&\quad + (LS_{\phi_3},\ ?_{\psi_3},\ r_2\hat{}\, AS\hat{}\, \{\!|\mathsf{proxy}\hat{}\, U|\!\}_{SK(LS)}\hat{}\, \{\!|\mathsf{id}\hat{}\, U|\!\}_{Sh^{AS}_{LS}},\ {}_{TLS^{S\to C}})\rangle\\
&User[U, AS, AS', LS, \psi_1, \psi_2, \psi_3, \psi_4, \phi_1, \phi_2, \phi_3, \phi_4, r_1, r_2, resp, rtok_1, rtok_2,\\
&\qquad stok_1, stok_2, pt, idtok, atok] \mathrel{\widehat{=}}\\
&\quad \langle +(?_{\psi_1},\ AS_{\phi_1},\ r_1,\ {}_{TLS^{C\to S}}), -(AS_{\phi_1},\ ?_{\psi_1},\ LS\hat{}\, rtok_1\hat{}\, stok_1,\ {}_{TLS^{S\to C}}),\\
&\quad + (?_{\psi_2},\ LS_{\phi_2},\ rtok_1\hat{}\, stok_1,\ {}_{TLS^{C\to S}}),\\
&\quad - (LS_{\phi_2},\ ?_{\psi_2},\ LoginForm\hat{}\, rtok_2\hat{}\, stok_2,\ {}_{TLS^{S\to C}}),\\
&\quad + (?_{\psi_3},\ LS_{\phi_3},\ U\hat{}\, passwd_{LS}(U)\hat{}\, rtok_2\hat{}\, stok_2,\ {}_{TLS^{C\to S}}),\\
&\quad - (LS_{\phi_3},\ ?_{\psi_3},\ AS'\hat{}\, r_2\hat{}\, pt\hat{}\, idtok,\ {}_{TLS^{S\to C}}),\\
&\quad + (?_{\psi_4},\ AS'_{\phi_4},\ r_2\hat{}\, idtok,\ {}_{TLS^{C\to S}}), -(AS'_{\phi_4},\ ?_{\psi_4},\ resp\hat{}\, atok,\ {}_{TLS^{S\to C}})\rangle
\end{aligned}
$$

3.3.3 Verification of WebAuth

Assumptions and Secrecy Lemmas

In order to prove the correctness of WebAuth we require a number of assumptions:

1. Honest application servers are configured with the correct service tokens and keys;
2. The penetrator does not initially possess any term that contains the secret key of a honest agent or a key shared amongst honest agents;
3. Penetrator application servers are configured with the correct service tokens;

4. The penetrator does not initially possess any term that contains the passwords of honest users (i.e. the penetrator has not obtained the user's password in advance).

These are formalised as follows.

Assumption 3.21. 1. If $st \in AS1[AS, LS, \psi_1, \phi_1, r_1, stok]$ then $stok = \{|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, Sh_{LS}^{AS}|\}_{SK(LS)}$;

2. If $A \in \mathcal{T}_{names}^{reg}$ then $SK(A) \notin \mathcal{A}_{\mathcal{P}}^{*}$ and if $B \in \mathcal{T}_{names}^{reg}$ then $Sh_B^A \notin \mathcal{A}_{\mathcal{P}}^{*}$;

3. If $LS \in \mathcal{T}_{names}^{reg}$ and $\{|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, k|\}_{SK(LS)} \in \mathcal{A}_{\mathcal{P}}^{*}$, then $k = Sh_{LS}^{AS}$;

4. If $U, LS \in \mathcal{T}_{names}^{reg}$ then $passwd_{LS}(U) \notin \mathcal{A}_{\mathcal{P}}^{*}$.

Further, we require that the user does not reveal her password except to the appropriate login server; i.e. the user is not tricked into giving her password away to the penetrator. In practice this means that the user, before divulging her password, should verify the LS by ensuring that the domain name matches the expected name; this requirement may not be obvious to all users. This assumption is formalised in the definition of the strand space: in message 6 on a *User* strand, $(?_{\psi_3},\ LS_{\phi_3},\ U\hat{}\, passwd_{LS}(U)\hat{}\, rtok_2\hat{}\, stok_2,\ {}_{TLS}{}^{C\to S})$, the identities of the recipient and of the server in $passwd_{LS}(U)$ are required to be equal.

The Confidentiality Guarantees

We start by proving that shared keys and passwords are confidential, as per Definition 3.14.

Lemma 3.22. Let $\mathcal{B}$ be a bundle from Σ. If $AS \in \mathcal{T}_{names}^{reg}$ and $LS \in \mathcal{T}_{names}^{reg}$ then Sh_{LS}^{AS} is confidential. Further, if $U \in \mathcal{T}_{names}^{reg}$ and $LS \in \mathcal{T}_{names}^{reg}$ then $passwd_{LS}(U)$ is confidential.

Proof. We prove the above result by showing that the relevant terms are safe and then using Lemma 3.18 to obtain confidentiality.

Firstly, given Assumption 3.21 (4) and the fact that $passwd_{LS}(U)$ is always sent confidentially in $\mathcal{B}$ (as ${}_{TLS}{}^{C\to S}$ satisfies $\mathcal{C}$), it immediately follows that $passwd_{LS}(U)$ is safe. Secondly, consider Sh_{LS}^{AS} for $AS, LS \in \mathcal{T}_{names}^{reg}$. By Assumption 3.21 (2), $SK(LS) \notin \mathcal{A}_{\mathcal{P}}^{*}$; further, by the definition of Σ, $SK(LS)$ does not appear as a subterm of any message. Therefore, $SK(LS)$ is a safe key. Assuming $AS \in \mathcal{T}_{names}^{reg}$ it follows by Assumption 3.21 (2) that $Sh_{LS}^{AS} \notin \mathcal{A}_{\mathcal{P}}^{*}$. Further, since Sh_{LS}^{AS} appears only as a subterm of messages encrypted using a safe key (i.e. $SK(LS)$) it follows that Sh_{LS}^{AS} is a safe key, as required. □

The User's Guarantees

We now analyse what the user can deduce having completed a full run of the protocol. The proposition and its proof are illustrated in Figure 3.8.

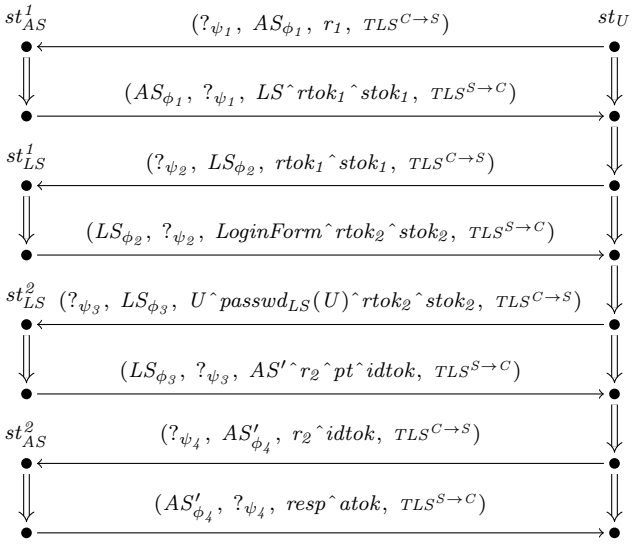


Figure 3.8: A graphical illustration of Proposition 3.23.

Proposition 3.23. Let $\mathcal{B}$ be a bundle from Σ and let

$$st_U \in User[U, AS, AS', LS, \psi_1, \psi_2, \psi_3, \psi_4, \phi_1, \phi_2, \phi_3, \phi_4, r_1, r_2, resp, rtok_1, rtok_2, stok_1, stok_2, pt, idtok, atok]$$

be a regular strand of $\mathcal{B}$-height 8. Then provided $LS \in \mathcal{T}^{reg}_{names}$:

1. If $AS \in \mathcal{T}^{reg}_{names}$ then there exists a strand $st^1_{AS} \in AS1[AS, LS, \psi_1, \phi_1, r_1, stok_1]$ of $\mathcal{B}$-height 2 such that $st_U(1) \to st^1_{AS}(1)$, $st^1_{AS}(2) \to st_U(2)$ and $rtok_1 = \{\!|\mathsf{req}\hat{}r_1|\!\}_{Sh^{AS}_{LS}}$;

2. There exists a strand $st^1_{LS} \in LS1[LS, AS'', \psi_2, \phi_2, k, r'_2]$ of $\mathcal{B}$-height 2 such that $st_U(3) \to st^1_{LS}(1)$ and $st^1_{LS}(2) \to st_U(4)$; further $stok_1 = stok_2 = \{\!|\mathsf{webkdc\_service}\hat{}AS''\hat{}Sh^{AS''}_{LS}|\!\}_{SK(LS)}$ and $rtok_1 = rtok_2 = \{\!|\mathsf{req}\hat{}r'_2|\!\}_{Sh^{AS''}_{LS}}$;

3. There exists a strand $st^2_{LS} \in LS2[LS, U, AS'', \psi_3, \phi_3, k, r'_2]$ of $\mathcal{B}$-height 2 such that $st_U(5) \to st^2_{LS}(1)$, $st^2_{LS}(2) \to st_U(6)$, $r_2 = r'_2$, $AS' = AS''$, $pt = \{\!|\mathsf{proxy}\hat{}U|\!\}_{SK(LS)}$ and $idtok = \{\!|\mathsf{id}\hat{}U|\!\}_{Sh^{AS''}_{LS}}$;

4. If $AS' \in \mathcal{T}^{reg}_{names}$ then there exists a strand $st^2_{AS} \in AS2[AS', U, LS, \psi_4, \phi_4, r_2, resp]$ of $\mathcal{B}$-height 2 such that $st_U(7) \to st^2_{AS}(1)$, $st^2_{AS}(2) \to st_U(8)$ and $atok = \{\!|\mathsf{app}\hat{}U\hat{}Sh^{AS'}_{LS}|\!\}_{SK(LS)}$;

5. If $AS \in \mathcal{T}^{reg}_{names}$ then $r_1 = r_2$ and $AS = AS'$;

6. If $resp \notin \mathcal{A}_{\mathcal{P}}$ and $AS' \in \mathcal{T}^{reg}_{names}$ then $resp$ is confidential in $\mathcal{B}$;

7. The strands st^1_{AS}, st^2_{AS}, st^1_{LS} and st^2_{LS} so defined are unique.

Proof. Let st_U be such a strand; we prove each of the points in turn as follows:

1. Assume that $AS \in \mathcal{T}_{names}^{reg}$. As $\textsc{tls}^{S \to C}$ satisfies $\mathcal{A}$, by the Authentication Rule there must exist a regular node n such that $n \to st_U(2)$, and thus that $msg(n) = msg(st_U(2))$. By inspection this can only be the second node on an $AS1$ strand st_{AS}^1. Consider the node n' such that $n' \to st_{AS}^1(1)$; since $\textsc{tls}^{C \to S}$ satisfies $\mathcal{A}$, the Session Rule can be applied to $st_{AS}^1(1)$ to deduce that n' must be regular and lie on st_U. By inspection this can only be the first node. Hence $msg(st_U(1)) = msg(st_{AS}^1(1))$ and therefore, $st_{AS}^1 \in AS1[AS, LS, \psi_1, \phi_1, r_1, stok_1]$, and thus $rtok_1 = \{|\mathsf{req}\hat{}\, r_1|\}_{Sh_{LS}^{AS}}$.

2. Again, as $\textsc{tls}^{S \to C}$ satisfies $\mathcal{AC}$ and since $LS_{\phi_2} \in \mathcal{I}^{reg}$ it follows, by the Authentication Rule, that there must exist a regular node n such that $n \to st_U(4)$, and thus that $msg(n) = msg(st_U(4))$. By inspection this can only be the second node on a $LS1$ strand st_{AS}^1. Also, by the Session Rule there exists a regular node n' on st_U such that $n' \to st_{LS}^1(1)$. By inspection this can only be $st_U(3)$, and therefore $st_{LS}^1 \in LS1[LS, AS'', \psi_2, \phi_2, k, r_2']$ for some AS'' and k. Therefore, it immediately follows that $rtok_2 = rtok_1 = \{|\mathsf{req}\hat{}\, r_2'|\}_{Sh_{LS}^{AS'}}$, and $stok_2 = stok_1 = \{|\mathsf{webkdc\_service}\hat{}\, Sh_{LS}^{AS'}|\}_{SK(LS)}$.

3. The proof of this case is similar to before.

4. The proof of this case is similar to before.

5. This follows from part 1 and part 3.

6. Note that $\textsc{tls}^{S \to C}$ satisfies $\mathcal{C}$ and therefore, provided $AS' \in \mathcal{T}_{names}^{reg}$, $resp$ is sent confidentially in $st_{AS}^2(2)$. Therefore, it immediately follows that $resp$ occurs safely providing $resp \notin \mathcal{A}_{\mathcal{P}}$, as this is the only high-level term in which it occurs. Thus, by Lemma 3.18, $resp$ is confidential, as required.

7. This follows immediately from Equation 3.1 as disjoint regular strands use disjoint channel ends. □

The Login Server's Guarantees

We now consider the guarantees to the login server. We require a lemma that shows that only correct keys can be embedded in service tokens.

Lemma 3.24. Let $\mathcal{B}$ be a bundle from Σ, $LS \in \mathcal{T}_{names}^{reg}$ and $st_{LS}^2 \in LS2[LS, U, AS, \psi_3, \phi_3, k, r_2]$ be a regular strand of $\mathcal{B}$-height at least 1. Then $k = Sh_{LS}^{AS}$.

Proof. Firstly, note that by Assumption 3.21 (2), $SK(LS) \notin \mathcal{A}_{\mathcal{P}}^*$; further it never appears as a subterm of a message. Hence, it follows trivially that $SK(LS)$ can never appear as the key edge on a E strand meaning that the penetrator cannot use $SK(LS)$ as an encryption key. Therefore, the penetrator can never construct a term of the form $\{|\mathsf{webkdc\_service}\hat{}\, AS\hat{}\, k|\}_{SK(LS)}$. Thus, it follows that terms of this form must either be from $\mathcal{A}_{\mathcal{P}}^*$, and hence of the correct form by Assumption 3.21 (3), or originate from regular AS stands and hence, due to Assumption 3.21 (1), be of the correct form. □

Proposition 3.25. Let $\mathcal{B}$ be a bundle from Σ, $LS \in \mathcal{T}_{names}^{reg}$ and $st_{LS}^2 \in LS2[LS, U, AS', \psi_3, \phi_3, k, r_2]$ be a regular strand of $\mathcal{B}$-height 2. Then:

1. If $AS' \in \mathcal{T}^{reg}_{names}$ then there exists a strand $st^1_{AS} \in AS1[AS', LS, \psi_1, \phi_1, r_2, stok]$ of $\mathcal{B}$-height 2;

2. If $U \in \mathcal{T}^{reg}_{names}$ then there exists a strand (writing $*$ for values that are arbitrary) $st_U \in User[U, AS, *, LS, \psi'_1, \psi_2, \psi_3, *, \phi'_1, \phi_2, \phi_3, *, r_1, *, *, rt, rt, stok_1, stok_1, *, *, *]$ of $\mathcal{B}$-height at least 5, and there exists a strand $st^1_{LS} \in LS1[LS, AS', \psi_2, \phi_2, k, r_2]$, such that $st_U(3) \rightarrow st^1_{LS}(1)$, $st^1_{LS}(2) \rightarrow st_U(4)$, $st_U(5) \rightarrow st^2_{LS}(1)$ and $st^2_{LS}(2) \rightarrow st_U(6)$;

3. If $AS, U \in \mathcal{T}^{reg}_{names}$ then $st_U(1) \rightarrow st^1_{AS}(1)$, $st^1_{AS}(2) \rightarrow st_U(2)$, $\psi_1 = \psi'_1$, $\phi_1 = \phi'_1$, $stok = stok_1$, $AS = AS'$ and $r_1 = r_2$.

Proof. Let st^2_{LS} be such a strand.

1. By Lemma 3.24, $k = Sh^{AS'}_{LS}$; and by Lemma 3.22, $Sh^{AS'}_{LS}$ is confidential. Therefore, the Unsolicited Authentication Test can be applied to $st^2_{LS}(1)$ to deduce that there must exist a regular node $n \prec st^2_{LS}(1)$ such that $\{|\mathsf{req}\hat{}r_2|\}_{Sh^{AS'}_{LS}}$ originates on n. By inspection this can only be the second node on an $AS1$ strand $st^1_{AS} \in AS1[AS', LS, \psi_1, \phi_1, r_2, stok]$.

2. Assuming that $U \in \mathcal{T}^{reg}_{names}$ it follows by Lemma 3.22 that $passwd_{LS}(U)$ is confidential. Therefore, by the Authentication via Confidentiality Rule it follows that there exists a regular node n' such that $n' \rightarrow st^2_{LS}(1)$. By inspection, this can only be the fifth node on a user strand, st_U:

$$msg(st_U(5)) = +(?_{\psi_3},\ LS_{\phi_3},\ U\hat{}passwd_{LS}(U)\hat{}rtok_2\hat{}stok_2,\ {}_{TLS}C{\rightarrow}S).$$

   Therefore $st_U \in User[U, AS, *, LS, \psi'_1, \psi_2, \psi_3, *, \phi'_1, \phi_2, \phi_3, *, r_1, *, *, rtok_1, rtok_2, stok_1, stok_2, *, *, *]$. Further, as $LS \in \mathcal{T}^{reg}_{names}$, by the Authentication Rule there exists a regular node n such that $msg(n) = msg(st_U(4))$. By inspection, this can only be the second node on a $LS1$ strand, st^1_{LS}. Hence, it follows that $\psi_2 \in ends(st_U(4))$ and thus that $?_{\psi_2} \in \mathcal{I}^{reg}$. Therefore, by the Session Rule applied to $st^1_{LS}(1)$ there exists a regular node n' on st_U such that $msg(n') = msg(st^1_{LS}(1))$. Furthermore, the only node of the correct form is $st_U(3)$, and hence it follows that $st^1_{LS} \in LS1[LS, AS', \psi_2, \phi_2, k, r_2]$, and that $stok_1 = stok_2$ and $rtok_1 = rtok_2$.

3. From the previous part, $msg(st_U(2)) = -(AS_{\psi_2},\ ?_{\phi_1},\ LS\hat{}rtok_1\hat{}stok_1)$. Hence, as $AS \in \mathcal{T}^{reg}_{names}$, by the Authentication Rule there exists a regular node n such that $msg(n) = msg(st_U(2))$. By inspection, this can only be the second node on an $AS1$ strand, $st^1_{AS} \in AS1[AS, LS, \psi_1, \phi_1, r'_1, stok_1]$, as required. Further, as $rtok = \{|\mathsf{req}\hat{}r_1|\}_{Sh^{AS}_{LS}}$, $r'_1 = r_1$. □

The Application Server's Guarantees

Lastly, we consider what an application server (in particular, an $AS2$ strand) can deduce having completed a run of the protocol.

Proposition 3.26. Let $\mathcal{B}$ be a bundle from Σ, $AS', LS \in \mathcal{T}^{reg}_{names}$ and $st^2_{AS} \in AS2[AS', U, LS, \psi_4, \phi_4, r_2, resp]$ be a regular strand of $\mathcal{B}$-height 2. Then:

1. There exists a strand $st^2_{LS} \in LS2[LS, U, AS', \psi_2, \phi_2, k, r'_2]$ of $\mathcal{B}$-height 2;
2. If $U \in \mathcal{T}^{reg}_{names}$ then:
   (a) There exists a strand $st_U \in User[U, AS, AS', LS, \psi_1, \psi_2, \psi_3, \psi_4, \phi_1, \phi_2, \phi_3, \phi_4, r_1, r_2, *, rt, rt, stok, stok, pt, idtok, atok]$ of $\mathcal{B}$-height at least 7;
   (b) $r'_2 = r_2$;
   (c) If $AS \in \mathcal{T}^{reg}_{names}$ then $AS = AS'$ and there exists a strand $st^1_{AS} \in AS1[AS, LS, \psi_1, \phi_1, r_1, stok]$ of $\mathcal{B}$-height 2;
   (d) There exists a strand $st^1_{LS} \in LS1[LS, AS', \psi_2, \phi_2, k, r_2]$ of $\mathcal{B}$-height 2;
   (e) $st_U(3) \rightarrow st^1_{LS}(1)$, $st^1_{LS}(2) \rightarrow st_U(4)$, $st_U(5) \rightarrow st^2_{LS}(1)$, $st^2_{LS}(2) \rightarrow st_U(6)$, $st_U(7) \rightarrow st^2_{AS}(1)$; and if $AS \in \mathcal{T}^{reg}_{names}$ then $st_U(1) \rightarrow st^1_{AS}(1)$, $st^1_{AS}(2) \rightarrow st_U(2)$, and $r_1 = r_2$.

Proof. Let st^2_{LS} be such a strand.

1. As $AS', LS \in \mathcal{T}^{reg}_{names}$, it follows by Lemma 3.22 that $Sh^{AS'}_{LS}$ is confidential. Hence, by the Unsolicited Authentication Test applied to $st^2_{AS}(1)$ there must exist a regular node $n \prec st^2_{AS}(1)$ such that $\{\!|\mathsf{id}\hat{}U|\!\}_{Sh^{AS'}_{LS}} \sqsubseteq msg(n)$. By inspection, this can only be the second node on a *LS2* strand, and hence there exists a strand $st^2_{LS} \in LS2[LS, U, AS', \psi_3, \phi_3, k, r'_2]$.
2. This follows immediately from a trivial extension of Proposition 3.25 to a *User* strand of $\mathcal{B}$-height 7 (we need to extend the *User* strand to ensure agreement between the *User* and *LS2* on r_2 and r'_2). □

Item 2c reveals a subtlety of the protocol: the application server has no guarantee that the user wishes to authenticate herself to it. For example, suppose there are two application servers, one dishonest, P, and one honest, AS. Further, suppose the user wishes to access a resource on P; when P redirects the user to the Login Server, rather than P sending his own service and request token, P can send a service token for AS and a request token for a resource r on AS. This means that the user, after successfully authenticating to LS, would be redirected to AS and would inadvertently request r. Clearly, this could be dangerous if r is a request that causes data to be modified or disclosed.

3.4 Summary

In this chapter we started, in Section 3.1, by presenting the high-level strand spaces model, which can model the security guarantees provided by both bilaterally and unilaterally authenticating secure transport protocols. This included defining the basic elements of the high-level strand spaces model, including high-level terms, strands, bundles, etc., as well as defining the various penetrator strands that interact with transport-layer terms.

In Section 3.2 we then defined and proved the correctness of a number of proof rules that could be used to prove the correctness of application-layer protocols in the high-level strand spaces model. These proof rules capture a variety of common protocol interactions, which should make them of significant use to others.

In Section 3.3 we then illustrated how the high-level strand spaces model and, in particular, the proof rules that we have developed can be used by proving the correctness of WebAuth, a web-based single-sign-on protocol. This proof was also interesting in that it showed how web-based protocols can be modelled and formally analysed. Further, this proof revealed an interesting subtly in the guarantees provided to the application server. We are not aware of any other correctness proofs for WebAuth.

Related Work Compared to the high-level strand spaces model defined by Kamil and Lowe in [KL09], the model in this thesis supports a wider variety of transport-layer protocols. In particular, we now are able to model the guarantees provided by unilaterally authenticating secure transport protocols, not just bilateral protocols. Further, the model now supports multiple sessions between the same honest agents, which is something that was not supported by the *session* property of [Kam10]. Lastly, the way in which transport-layer protocols that have sequence numbers are modelled has been altered, in order to make it simpler to work with. In particular, the *stream* property of [Kam10] required that the messages received at a particular name must be a prefix of those sent to it. Whilst this property did capture the intended meaning, it was not very strand-spaces like, as it is a global property. In particular, this means that the act of a penetrator doing a renumber was implicit in the graph, whereas it is now made explicit, via a RN strand.

In addition to our proof of correctness of WebAuth above, in [Hoy12] Hoyland proved the correctness of a variant of OAuth [JH12] using the high-level strand spaces model. This suggests that our model is able to prove the correctness of a variety of different application-layer protocols. We have also developed a prototype tool that is able to automatically prove all of the security guarantees on WebAuth and OAuth. Essentially, the tool works by automatically applying the proof rules of this chapter, along with a few new proof rules. We discuss this further in Section 9.1.

The most similar approach to verifying layered protocols by abstraction is that of Mödersheim and Viganò [MV09a]. They define a model, the *Ideal Channel Model (ICM)*, that abstracts away from how the channels are implemented. They then consider how to model the guarantees given by unilaterally authenticating transport-layer protocols (or *secure pseudonymous channels*). In their model they specify confidential, authentic and secure channels, which roughly correspond to $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{AC}$ respectively. The primary difference between the two approaches is that whilst both formalisms permit analysis of protocols that use bilateral or unilateral transport protocols, ours also allows protocols that do not group messages into sessions to be analysed (i.e. by letting the channel end be ?). Another difference is that they address unauthenticated clients using *pseudonyms* rather than our name and channel end combination, which we use to enable bilateral and unilateral protocols to be considered together more uniformly. We also think that this clarifies the model and makes it clearer what is occurring at the transport layer. Mödersheim and Viganò have developed tool support to enable protocols to be proven correct in their model automatically. Currently, we only have a prototype tool for verifying the correctness of protocols in the high-level strand spaces model, as we discuss further in Section 9.1.

The two other main approaches to modelling layered security protocols, as discussed in Section 1.1, are an extension to the inductive approach [BLP03], and

using a LTL-based model checking approach [ACC07]. Neither of these allows the guarantees provided by unilaterally authenticating secure transport protocols to be modelled although in both cases there appears to be no reason why the model could not be extended to support it. Neither of these models supports grouping messages into sessions or prevents messages from being reordered.

Chapter 4

Soundness of the Abstraction

Analysing layered security protocols by abstracting away from the implementation of the transport layer, as in Chapter 3, introduces a proof obligation that this abstraction is sound. In particular, it requires one to prove that, if there is an attack on the protocol when implemented using a transport layer protocol that satisfies the channel properties, then there is also an attack on the abstracted protocol. In this chapter we begin to prove such a result for the high-level strand spaces model. In particular the outline of the overall proof is as follows:

1. In Section 4.2 and Section 4.3 we define what it means for a high-level bundle to *abstract* a low-level bundle. Informally, this means that they have the same application-layer behaviour. We then prove, in Section 4.4, that if a bundle is *interference-free*, which is a semantic condition, then it can be abstracted.

2. Chapter 5 proves that every bundle $\mathcal{B}$ of a low-level strand space Σ that satisfies our statically-checkable condition, known as *disjoint-layered encryption*, can be transformed to an interference-free low-level bundle $\mathcal{B}'$ of a related strand space Σ_E. Further, $\mathcal{B} \trianglerighteq \mathcal{B}'$ which, informally, means that $\mathcal{B}'$ has *similar* application-layer behaviour.

3. Lastly, Chapter 6 introduces a logic in which protocol-correctness properties can be expressed. If ϕ is such a property, then this section also proves that whenever $\mathcal{B} \trianglerighteq \mathcal{B}'$ and ϕ is not satisfied by $\mathcal{B}$, then ϕ is also not satisfied by $\mathcal{B}'$. This therefore implies that the transformations of Chapter 5 preserve any application-layer incorrectness. Lastly, we also prove that whenever $\hat{\mathcal{B}}'$ abstracts $\mathcal{B}'$ and $\mathcal{B}'$ does not satisfy a property ϕ, then $\hat{\mathcal{B}}'$ also does not.

Together, these results prove that whenever there is an attack against a low-level bundle, there is also an attack against a high-level bundle, thus showing the model is sound.

We begin, in Section 4.1 by defining a number of preliminaries and, in particular, consider how to model arbitrary transport-layer protocols in the low-level strand spaces model. In Section 4.2 and Section 4.3 we formalise the relationship between the low and high-level bundles by defining various relations (one for the regular nodes and two for the penetrator nodes) that relate nodes of low and high-level bundles. Section 4.4 then introduces various definitions in order to define the *independence*

assumption. Finally, in Section 4.5 we prove the main result; namely that for every low-level bundle, if it satisfies the independence assumption, then there exists a corresponding abstracted high-level bundle. This work in this chapter is based on [KL10, Kam10]. Essentially, the work in Sections 4.2 and 4.3 is a straightforward adaptation, but the remainder of the work is new. We give more details on this in Section 4.6.

For the remainder of this thesis, when it is not clear from the context whether we are in the low or high-level strand spaces model, we will denote high-level objects with a hat. For example, $\hat{\mathcal{B}}$ denotes a high-level bundle, whereas $\mathcal{B}$ denotes a low-level bundle.

4.1 Preliminaries

In this section we define several preliminaries that are required for our soundness proof. Firstly, we define several different types of penetrator paths, which are useful for defining low-level versions of high-level strands. Then, we define a method of extracting a subterm that lies at a particular position within a term. Lastly, we describe how generic transport-layer protocols can be represented in the low-level strand spaces model. The notation from this section is used extensively throughout the rest of the thesis and is summarised in Appendix A.

Penetrator Paths

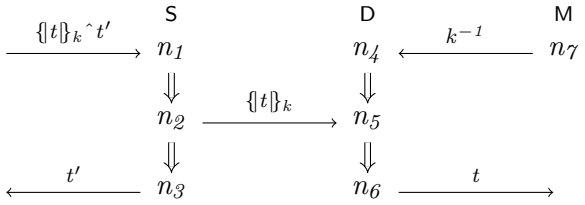
In order to define low-level penetrator behaviours like sending and hijacking, Kamil and Lowe used the notion of *penetrator subpaths*. These allowed them to define how the various transport-layer penetrator strands (such as HJ and FK strands) are translated to the low-level model. Formally, a penetrator subpath is a sequence of penetrator nodes where each consecutive pair of nodes is linked by either $\Rightarrow^+$ or $\rightarrow$. It is then possible to differentiate between several different types of penetrator subpath as follows.

Definition 4.1 (Based on [KL10])**.** A penetrator subpath p in a low-level bundle is:

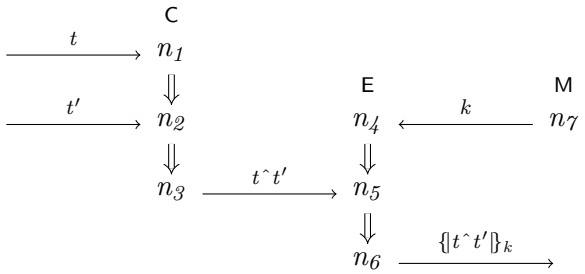
1. *normal* iff whenever a constructive edge sends to a destructive edge, the constructive edge is a KG strand (cf. Definition 2.15);
2. *constructive* iff it only traverses constructive edges;
3. *destructive* iff it only traverses destructive edges.

Extraction Paths

We will frequently wish to check if a given term is *the* application-layer message of a transport-layer message. However, we cannot use the subterm relation to check this. For example, suppose we wished to check if $A \in \mathcal{T}_{names}$ was the application-layer message, but transport-layer packing included A, then using the subterm relation could induce false positives. Hence, we would like the ability to retrieve the term that lies in a particular position. We thus define *extraction paths* which allow us to specify the subterm to extract from a term, given the position of it. Conceptually, they are paths in the abstract syntax tree.



(a) A bundle containing a destructive penetrator path $\langle n_1, n_2, n_5, n_6 \rangle$.



(b) A bundle containing a constructive penetrator path $\langle n_1, n_3, n_5, n_6 \rangle$.

Figure 4.1: Bundles to illustrate the relationship between extraction and penetrator paths.

Definition 4.2. The set of all *extraction paths*, $\mathcal{EP}$, is defined as $(\{1, 2, \mathsf{Decrypt}\})^*$. The partial function $extract : \mathcal{A} \times \mathcal{EP} \to \mathcal{A}$ is defined as follows:

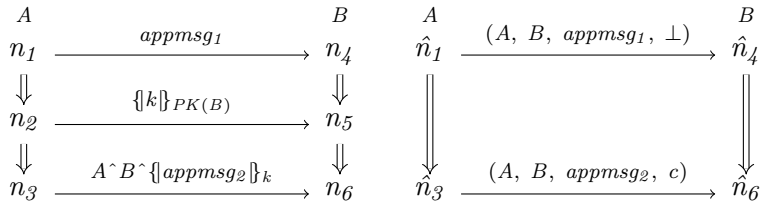
$$\begin{aligned} extract(t, \langle\rangle) &\mathrel{\widehat{=}} t \\ extract(t_1 \hat{} t_2, \langle i \rangle \hat{} p) &\mathrel{\widehat{=}} extract(t_i, p) \\ extract(\{\!|m|\!\}_k, \langle \mathsf{Decrypt} \rangle \hat{} p) &\mathrel{\widehat{=}} extract(m, p). \end{aligned}$$

Given an extraction path es and $x \in \{1, 2, \mathsf{Decrypt}\}$, we write x in es iff there exists an index i such that the i^{th} element is x. We denote $t_1 = extract(t_2, es)$ by $t_1 \sqsubseteq_{es} t_2$, i.e. t_1 is the subterm of t_2 at position es.

For example, if $t = \{\!|t_1|\!\}_k \hat{} t_2$, then $t_1 = extract(t, \langle 1, \mathsf{Decrypt} \rangle)$ (i.e. $t_1 \sqsubseteq_{\langle 1, \mathsf{Decrypt} \rangle} t$) and $t_2 = extract(t, \langle 1 \rangle)$.

It will be helpful, particularly in Chapter 5, to consider what the penetrator is doing in terms of extraction paths. For example, consider Figure 4.1a and the destructive penetrator path $\langle n_1, n_2, n_5, n_6 \rangle$. This penetrator path takes the first component of a concatenation and then decrypts the message. Therefore, this can be represented by the extraction path $\langle 1, \mathsf{Decrypt} \rangle$. Alternatively, consider Figure 4.1b and the constructive penetrator path $\langle n_1, n_3, n_5, n_6 \rangle$. This takes the term t and packages it up as the first component of a concatenation, encrypting the result. This can be represented by the extraction path $\langle \mathsf{Decrypt}, 1 \rangle$ that extracts the original term t from the result term $\{\!|t \hat{} t'|\!\}_k$.

In general, given a destructive or constructive penetrator subpath that does not cross a key-edge, it is possible to obtain a corresponding extraction path. This is formalised below.



(a) The low-level bundle

(b) The corresponding high-level bundle

Figure 4.2: A low-level bundle that sends application-layer messages, one over the unprotected channel and another using a simple transport-layer protocol. The corresponding high-level bundle is also given.

Lemma 4.3. Let p be a constructive or destructive penetrator path in a bundle $\mathcal{B}$ starting at a node n_1 and ending at a node $n_{|p|}$. Providing p does not traverse a key edge or a KG strand there exists an extraction path es such that:

- If p is destructive $extract(msg(n_1), es) = msg(n_{|p|})$; we define $\mathsf{expath}\,(p) \mathrel{\widehat{=}} es$;
- If p is constructive $extract(msg(n_{|p|}), es) = msg(n_1)$; we define $\mathsf{expath}^{\sim}\,(p) \mathrel{\widehat{=}} es$.

Transport Layer Modelling

In this section we define how to extract application-layer information, such as the sender of a transport-layer message, from a low-level strand space including arbitrary secure-transport protocols. We also define some assumptions on the transport-layer protocols and the low-level strand space in order to make sure the above functions are well defined.

Throughout this subsection we use the example of a simple transport-layer protocol that sends an application-layer message m from A to B as follows:

$$\begin{array}{lll} 1. & A \to B & : \{\!|k|\!\}_{PK(B)} \\ 2. & A \to B & : A\hat{}B\hat{}\{\!|m|\!\}_k \end{array}$$

This channel is confidential, as only B can decrypt the message containing they key k, but is not authentic as the penetrator can trivially alter the apparent sender by changing message 2. An example bundle, which is used throughout the following, is given in Figure 4.2.

We start by stating an assumption regarding the partitioning of the set of regular nodes according to the type of transport-layer content they contain (if any). This has to be an assumption, rather than a definition, as it is not always possible to disambiguate between certain types of nodes.

Assumption 4.4 (Based on [KL10])**.** The set of regular nodes $\mathcal{N}^{reg}$ is partitioned between:

- The set of *transport-layer nodes*, denoted $\mathcal{N}_{trpt}$, that contains all regular nodes that send or receive application-layer messages on a channel other than $\bot$;

- The set of *unprotected regular nodes*, denoted $\mathcal{N}_{\perp}$, that contains all regular nodes that send or receive application-layer messages over $\perp$;
- The set of *transport-layer non-message nodes*, denoted $\mathcal{N}_{non\text{-}payload}$, that contains all regular nodes that are related to the transport-layer, but do not send or receive application-layer messages (e.g. handshake nodes).

The set of *application-layer nodes*, denoted $\mathcal{N}_{payload}$, is defined as $\mathcal{N}_{trpt} \cup \mathcal{N}_{\perp}$.

For example, in Figure 4.2, $\{n_1, n_4\} = \mathcal{N}_{\perp}$, $\{n_2, n_5\} = \mathcal{N}_{non\text{-}payload}$ and $\{n_3, n_6\} = \mathcal{N}_{trpt}$.

We now consider how to transform transport-layer messages into high-level terms. Firstly, we will assume the existence of several sets of terms related to the transport-layer terms. We also require the existence of a function that extracts the sequence number of a transport-layer payload term. Lastly, we require the existence of an extraction path that can be used to extract the application-layer message from a transport-layer payload term. These assumptions are formalised as follows. Note that the following excludes $\perp$ as this requires special treatment.

Assumption 4.5 (Based on [KL10])**.** Given a low-level strand space Σ, for each channel $c \in \mathit{Channels} \setminus \{\perp\}$ we assume the existence of:

- A set of *transport-layer payload terms*, $\mathcal{T}^{c}_{payload} \subseteq \mathcal{A}$, that contains all terms that carry an application-layer payload on channel c.
- A set of *transport-layer non-payload terms*, $\mathcal{T}^{c}_{non\text{-}payload} \subseteq \mathcal{A}$, that contains all other terms used by the channel c (e.g. handshake terms).
- An extraction path $\mathsf{expath}^{\mathsf{c}}(c) \neq \langle\rangle$, that extracts the application-layer message from a transport term $t \in \mathcal{T}^{c}_{payload}$.

We also define:

$$\begin{aligned}
\mathcal{T}^{\perp}_{payload} &\mathrel{\widehat{=}} \{msg(n) \mid n \in \mathcal{N}_{\perp}\} \\
\mathcal{T}^{\perp}_{non\text{-}payload} &\mathrel{\widehat{=}} \emptyset \\
\mathcal{T}_{payload} &\mathrel{\widehat{=}} \bigcup_{c \in \mathit{Channels}} \mathcal{T}^{c}_{payload} \\
\mathcal{T}_{non\text{-}payload} &\mathrel{\widehat{=}} \bigcup_{c \in \mathit{Channels}} \mathcal{T}^{c}_{non\text{-}payload}.
\end{aligned}$$

Thus, $\mathcal{T}_{payload}$ consists of all transport-layer terms that contain application-layer messages, whilst $\mathcal{T}_{non\text{-}payload}$ consists of all transport-layer terms that do not contain application-layer messages.

Further, we assume that for all channels $c \in \mathit{Channels}$:

$$\begin{aligned}
\mathcal{T}^{c}_{payload} &= \{msg(n) \mid n \in \mathcal{N}^{c}_{trpt}\} \\
\mathcal{T}^{c}_{non\text{-}payload} &= \{msg(n) \mid n \in \mathcal{N}^{c}_{non\text{-}payload}\}.
\end{aligned}$$

For example, considering Figure 4.2 and the example channel c, $\mathcal{T}^c_{payload}$ contains all terms of the form $A \hat{} B \hat{} \{\!|appmsg_2|\!\}_k$, $\mathcal{T}^c_{non\text{-}payload}$ contains all terms of the form $\{\!|k|\!\}_{PK(B)}$ and $\mathsf{expath^c}\,(c) = \langle 2, 2, \mathsf{Decrypt}\rangle$ (assuming that $\hat{}$ associates to the right).

Note that the existence of $\mathsf{expath^c}\,(c)$ implicitly assumes that the application-layer message is a subterm of the transport-layer payload term. Further, it requires that all application-layer messages on a given channel are packaged in the same way. These conditions are satisfied by every secure transport protocol that we have considered[1].

We now make an assumption that requires the existence of functions to extract the sender and recipient of high-level terms in a particular bundle.

Assumption 4.6 (Based on [KL10])**.** For each low-level bundle $\mathcal{B}$ of Σ we assume the existence of:

- $sender^c_{\mathcal{B}} : \mathcal{T}^c_{payload} \to \mathcal{I}$ that gives the sender of a transport-layer payload term on channel c;
- $recipient^c_{\mathcal{B}} : \mathcal{T}^c_{payload} \to \mathcal{I}$ that gives the recipient of a transport-layer payload term on channel c;
- $seqno^c_{\mathcal{B}} : \mathcal{T}^c_{payload} \to \mathcal{S}$ that gives the sequence number (possibly $\_$) of the transport-layer payload term.

Further, for each regular node $n \in \mathcal{N}_{trpt}$, if n is positive, $sender_{\mathcal{B}}(msg(n)) \in \mathcal{I}^{reg}$ and, if n is negative, $recipient_{\mathcal{B}}(msg(n)) \in \mathcal{I}^{reg}$.

For example, using our running example, in Figure 4.2 $sender_{\mathcal{B}}(msg(n_3)) = A$ and $recipient_{\mathcal{B}}(msg(n_3)) = B$.

Observe that we cannot define *sender* and *recipient* over the strand space since it is possible for the sender or recipient of a particular transport-layer term to be different in different bundles. For example, if the transport-layer protocol in use was TLS then the sender and recipient of a given transport-layer term are determined by the encryption key used to protect the application-layer message. However, as it is possible for the same encryption key to be derived in two different bundles by different participants it follows that *sender* and *recipient* are bundle-specific.

The same also applies to *seqno*. In TLS, each end of a channel records the number of messages that have been sent or received. The sequence number is then included not as part of the plaintext, but instead it is incorporated into the hash that is sent with each message. Regular agents then verify that the messages are received in the correct order by including the sequence number that they expect in the data they hash to verify the message. Hence, the sequence number is a bundle-specific property, since it is dependent on what terms have been sent before and is therefore dependent on the structure of the bundle.

Using the above functions we can define several functions that extract components from arbitrary transport-layer payload terms. These will later be used to define a mapping between transport-layer terms and the high-level terms that they represent. In order to define these functions it will be necessary to know which channel a

[1]It would be relatively straightforward to support transport-layer protocols that had multiple application-layer extraction paths, providing the extraction path that was in use could be determined by a value within the application-layer message.

particular transport-layer payload term was sent on. To ensure such a function is well-defined we clearly need to ensure that no two channels share any transport-layer payload terms. We formalise this assumption as follows.

Definition 4.7 (Based on [KL10])**.** Channel c_1 has *disjoint transport-layer payload messages* with a channel c_2 iff $\mathcal{T}^{c_1}_{payload} \cap \mathcal{T}^{c_2}_{payload} = \emptyset$.

Assumption 4.8 (Based on [KL10])**.** For all distinct channels $c_1 \neq \bot$ and $c_2 \neq \bot$, c_1 has disjoint transport-layer payload messages with c_2.

Given the above assumption we can now define the following functions.

Definition 4.9 (Based on [KL10])**.** Let Σ be a low-level strand space. The following functions are defined over all $\mathcal{T}_{payload}$:

$$\begin{aligned} chan &: \mathcal{T}_{payload} \to (Channels \setminus \{\bot\}) \\ chan(t) &\mathrel{\widehat{=}} c \text{ where } t \in \mathcal{T}^{c}_{payload} \\ appmsg &: \mathcal{T}_{payload} \to \mathcal{T}_{app} \\ appmsg(t) &\mathrel{\widehat{=}} extract(t, \mathsf{expath}^{\mathsf{c}}(chan(t))) \end{aligned}$$

Thus *chan* gives the channel of a transport-layer term (assuming Assumption 4.8); and *appmsg* gives the application-layer message contained in a transport-layer term.

For each $\mathcal{B}$ of Σ we define $sender_{\mathcal{B}}, recipient_{\mathcal{B}} : \mathcal{T}_{payload} \to \mathcal{I}$, which give the claimed sender and intended recipient end points of a transport layer-term, respectively. Further, we define $seqno_{\mathcal{B}} : \mathcal{T}_{payload} \to \mathcal{S}$, which gives the claimed sequence number (possibly _) for a given message. Formally, these functions are defined by:

$$\begin{aligned} sender_{\mathcal{B}}(t) &\mathrel{\widehat{=}} sender^{chan(t)}_{\mathcal{B}}(t) \\ recipient_{\mathcal{B}}(t) &\mathrel{\widehat{=}} recipient^{chan(t)}_{\mathcal{B}}(t) \\ seqno_{\mathcal{B}}(t) &\mathrel{\widehat{=}} seqno^{chan(t)}_{\mathcal{B}}(t). \end{aligned}$$

When the bundle is clear from the context we simply write $sender(t)$ and $recipient(t)$. We lift *chan*, *sender*, *recipient* and *seqno* to be defined on $\mathcal{N}_{trpt}$ in the obvious way (e.g. $chan(n) = chan(msg(n))$. We lift *appmsg* to be defined on $\mathcal{N}_{payload}$ by defining $appmsg(n)$ to be $msg(n)$ if $n \in \mathcal{N}_{\bot}$ and $appmsg(msg(n))$ otherwise (i.e. if $n \in \mathcal{N}_{trpt}$).

For example, in the case of Figure 4.2, $sender(msg(n_3)) = A_?$, $recipient(msg(n_3)) = B_?$, $seqno(msg(n_3)) = \_$, $appmsg(msg(n_3)) = appmsg_2$ and $chan(msg(n_3)) = c$.

We now consider what conditions are required to ensure that the above functions are well-defined. Recall that in the high-level strand spaces model (in particular in Assumption 3.3) we disallowed the penetrator from transforming a message where, for example, the sending channel end was ? into one where the sending channel end was not ?. In order to ensure that there are no penetrator behaviours that would be prohibited at the high-level we lift this assumption to an assumption on the low-level bundles by insisting that either all terms sent on c have sending channel end ?, or none do. Since we require, in Equation 3.1, that distinct regular strands use disjoint channel ends we will also require that the functions *sender* and *recipient* are defined in such a way that ensures this. Lastly, since we require in Assumption 3.5 that the sequence numbers are actually used sequentially, we also need to assume that this holds in the low-level strand space. We lift these assumptions as follows.

Assumption 4.10. For all low-level bundles $\mathcal{B}$, channels $c \in \mathit{Channels} \setminus \{\bot\}$, and terms $t_1, t_2 \in \mathcal{T}^c_{payload}$ sent on c:

1. $name(sender_{\mathcal{B}}(t_1)) = ?$ iff $name(sender_{\mathcal{B}}(t_2)) = ?$;
2. $end(sender_{\mathcal{B}}(t_1)) = ?$ iff $end(sender_{\mathcal{B}}(t_2)) = ?$;
3. $name(recipient_{\mathcal{B}}(t_1)) = ?$ iff $name(recipient_{\mathcal{B}}(t_2)) = ?$;
4. $end(recipient_{\mathcal{B}}(t_1)) = ?$ iff $end(recipient_{\mathcal{B}}(t_2)) = ?$;
5. $seqno_{\mathcal{B}}(t_1) = \_$ iff $seqno_{\mathcal{B}}(t_2) = \_$;
6. If $seqno_{\mathcal{B}}(t_1) \neq \_$ then $end(sender_{\mathcal{B}}(t_1)) \neq ?$ and $end(recipient_{\mathcal{B}}(t_1)) \neq ?$ (and similarly for t_2).

Further, for all low-level bundles $\mathcal{B}$ and low-level strands st and st' in $\mathcal{B}$, if $st \neq st'$ then $ends'_{\mathcal{B}}(st) \cap ends'_{\mathcal{B}}(st') = \emptyset$ (cf. Equation 3.1), where $ends'_{\mathcal{B}}$ is the low-level version of $ends$ and is formally defined as:

$$\begin{aligned} ends'_{\mathcal{B}}(st) \mathrel{\widehat{=}} &\{end(sender_{\mathcal{B}}(n)) \mid n \in \mathcal{N}_{trpt} \wedge sign(n) = + \wedge n \text{ is on } st\} \\ &\cup \{end(recipient_{\mathcal{B}}(n)) \mid n \in \mathcal{N}_{trpt} \wedge sign(n) = - \wedge n \text{ is on } st\}. \end{aligned}$$

Lastly, for all low-level bundles $\mathcal{B}$ and for all regular strands st in Σ, the function $th^{st}_{\mathcal{B}} : \mathcal{C} \times \mathcal{C} \to \mathcal{S}^*$, which returns the list of sequence numbers sent between two channel ends on a strand, is defined by:

$$\begin{aligned} th^{st}_{\mathcal{B}}(\psi, \phi) \mathrel{\widehat{=}} \langle &seqno_{\mathcal{B}}(msg(st, i)) \mid i \in \langle 1..\rangle, i \leq |st|, (st, i) \in \mathcal{N}_{payload}, \\ &end(sender_{\mathcal{B}}(msg(st, i))) = \psi, end(recipient_{\mathcal{B}}(msg(st, i))) = \phi\rangle. \end{aligned}$$

For all regular strands st, and all $\psi, \phi \in \mathcal{C}$, either $th^{st}_{\mathcal{B}}(\psi, \phi) \leq \langle 1..\rangle$, or $th^{st}_{\mathcal{B}}(\psi, \phi) \in \{\_\}^*$.

4.2 Relating Regular Nodes

The first step in the proof is to define an abstraction function, $\hat{\alpha}_{\mathcal{B}}$, that translates low-level terms from $\mathcal{B}$ into the corresponding high-level terms. For example, given the low-level term $A \hat{} B \hat{} \{|m|\}_{PK(B)}$ above, $\hat{\alpha}_{\mathcal{B}}$ returns $(A,\ B,\ \_,\ m,\ c)$.

Definition 4.11 (Based on [KL10])**.** Let $\mathcal{B}$ be a low-level bundle. The *term mapping function* $\hat{\alpha}_{\mathcal{B}} : \mathcal{A} \to \hat{\mathcal{A}}$ that maps low-level terms to the corresponding high-level terms is defined as follows:

1. for $m \in \mathcal{T}_{payload}$:

   $$\hat{\alpha}_{\mathcal{B}}(m) \mathrel{\widehat{=}} (sender_{\mathcal{B}}(m),\ recipient_{\mathcal{B}}(m),\ seqno_{\mathcal{B}}(m),\ appmsg(m),\ chan(m));$$

2. for $m \in \mathcal{A} \setminus \mathcal{T}_{payload}$, $\hat{\alpha}_{\mathcal{B}}(m) = (?_?,\ ?_?,\ \_,\ m,\ \bot)$;
3. if m is a directed term then $\hat{\alpha}_{\mathcal{B}}(m)$ has the same direction as m.

The bundle is omitted when it is clear from the context.

Using the above we specify a class of functions that, given a low-level node containing an application-layer message, return the corresponding high-level node. Furthermore, such functions are defined so as to preserve the bundle structure.

Definition 4.12 (From [KL10])**.** A *regular node map* is a partial function $\hat{\phi}$ that maps regular nodes of a low-level bundle $\mathcal{B}$ onto those of a high-level bundle $\hat{\mathcal{B}}$ and such that:

1. $dom(\hat{\phi}) = \mathcal{N}_{payload}$;
2. $\hat{\alpha}_{\mathcal{B}}(msg(n)) = msg(\hat{\phi}(n))$;
3. $\hat{\phi}$ is injective and surjective onto the regular nodes of $\hat{\mathcal{B}}$;
4. for $n, n' \in dom(\phi)$, $n \Rightarrow^{+}_{\mathcal{B}} n'$ iff $\hat{\phi}(n) \Rightarrow^{+}_{\hat{\mathcal{B}}} \hat{\phi}(n')$.

For example, the regular node map between the low and high-level bundles of Figure 4.2 is given by the function $\hat{\phi}$ where $\hat{\phi}(n_1) = \hat{n}_1$, $\hat{\phi}(n_3) = \hat{n}_3$, $\hat{\phi}(n_4) = \hat{n}_4$, $\hat{\phi}(n_6) = \hat{n}_6$.

4.3 Relating Penetrator Nodes

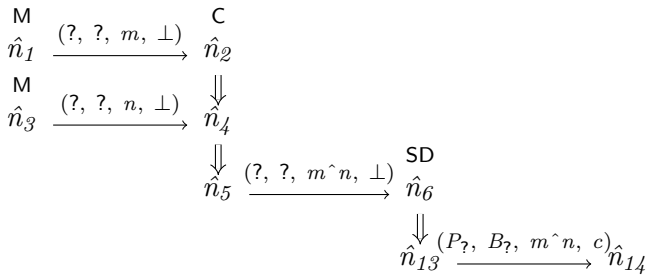
In this section we define how the penetrator nodes of the low-level bundle are related to the penetrator nodes of the high-level bundle. This is done by defining two different maps, the first of which maps non-application layer nodes to application-layer penetrator nodes. This definition is similar to Definition 4.12.

Definition 4.13 (From [KL10])**.** An *application-layer penetrator node map* is a partial injective function $\hat{\beta}_1$ from penetrator nodes of $\mathcal{B}$ to those of $\hat{\mathcal{B}}$, whose domain is a subset of $\Sigma \setminus \mathcal{N}_{payload}$, and such that if $n \in dom(\hat{\beta}_1)$ then:

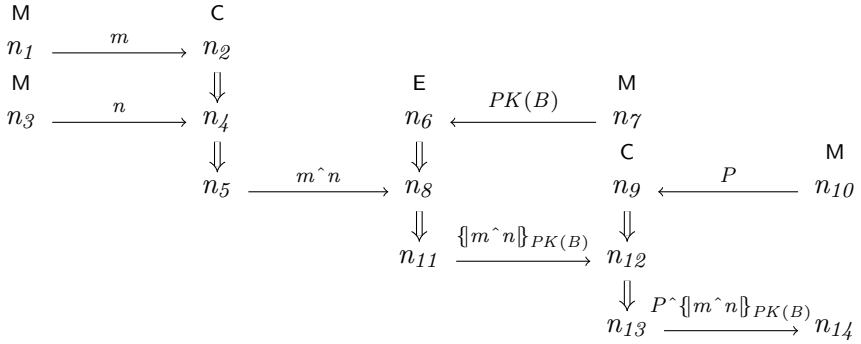
1. $\hat{\alpha}_{\mathcal{B}}(msg(n)) = msg(\hat{\beta}_1(n)) = (?_?, ?_?, \_, msg(n), \bot)$;
2. $\hat{\beta}_1(n)$ is on the same type of strand as n (e.g. if n is on a low-level E strand then $\hat{\beta}_1(n)$ is on a high-level E strand);
3. $\hat{\beta}_1$ respects the strand structure: for $n, n' \in dom(\hat{\beta}_1)$, $n \Rightarrow^{+} n'$ iff $\hat{\beta}_1(n) \Rightarrow^{+} \hat{\beta}_1(n')$.

For example, considering the bundle shown in Figure 4.3b, a valid application-layer penetrator node map would be the map that maps n_1, n_2, n_3, n_4 and n_5 to $\hat{n}_1$, $\hat{n}_2$, $\hat{n}_3$, $\hat{n}_4$ and $\hat{n}_5$ respectively.

Relating transport-layer penetrator nodes requires more thought since some of the transport-layer high-level penetrator strands (i.e. HJ, FK, SD, RV, LN and RN strands) do not correspond directly to single low-level penetrator strands, but instead to a collection of them. For example, consider the transport-layer protocol (which we will use as a running example for the remainder of this section) that encodes a high-level term $t = (A_?, B_?, \_, m, c)$ by $A \hat{} \{\!|m|\!\}_{PK(B)}$. Using this protocol, the path from n_6 to n_{13} in Figure 4.3b is the low-level representation of a high-level SD strand that sends the message $m \hat{} n$ to an honest agent B. We will define a mapping



(a) The corresponding high-level bundle, including the high-level SD strand.



(b) A bundle containing a low-level Send penetrator subpath.

Figure 4.3: An illustration of how the low-level penetrator subpaths and high-level penetrator strands relate.

from such low-level penetrator subpaths to the corresponding high-level penetrator strands.

Note that on a single penetrator path it is possible for the penetrator to change both the apparent sender and the sequence number associated with a message (e.g. Figure 4.5h). Therefore, for technical reasons, that are explained in Remark 4.22, we require a strand that does both re-numbering and hijacking at the same time. Further, we will also require a high-level penetrator strand that simply copies the application-layer message. This is again required for technical reasons, which are discussed in Remark 4.20. We define these new penetrator strands as follows.

Definition 4.14. The transmit and hijack-renumbering penetrator strands are defined as:

TX Transmit: $\langle -(A_\psi,\ B_\phi,\ i,\ m,\ c), +(A_\psi,\ B_\phi,\ i,\ m,\ c)\rangle$;

HJRN Hijack-Renumbering: $\langle -(S_\psi,\ R_\phi,\ i,\ m,\ c), +(S'_{\psi'},\ R'_{\phi'},\ i',\ m,\ c)\rangle$ providing $i' \neq i$, and either $S_\psi \neq S'_{\psi'}$ or $R_\phi \neq R'_{\phi'}$.

Note that by Assumption 4.10 in any such HJRN strand, $i \neq \_$, $i' \neq \_$, $S = ? \Leftrightarrow S' = ?$, $R = ? \Leftrightarrow R' = ?$, $\psi = ? \Leftrightarrow \psi' = ?$ and $\phi = ? \Leftrightarrow \phi' = ?$, as required.

However, note that both TX and HJRN strands are unnecessary, in the sense that they can be replaced by a direct message sending edge or combinations of HJ and RN strands, respectively. Therefore, when proving a result in the high-level strand spaces model for a particular bundle $\hat{\mathcal{B}}$ we can instead consider the equivalent bundle $\hat{\mathcal{B}}'$ that contains no HJRN strands. Note, however, that the above is only true if the channel conditions, such as $\mathcal{AC}$, permit HJRN strands iff they permit equivalent combinations of HJ and RN strands. We formalise this assumption as follows. Note that we also assume that all channels permit TX strands. This will be required in the proof of Proposition 4.31.

Assumption 4.15. All channels c permit TX strands, and permit HJRN strands iff they permit an equivalent combination of HJ and RN strands.

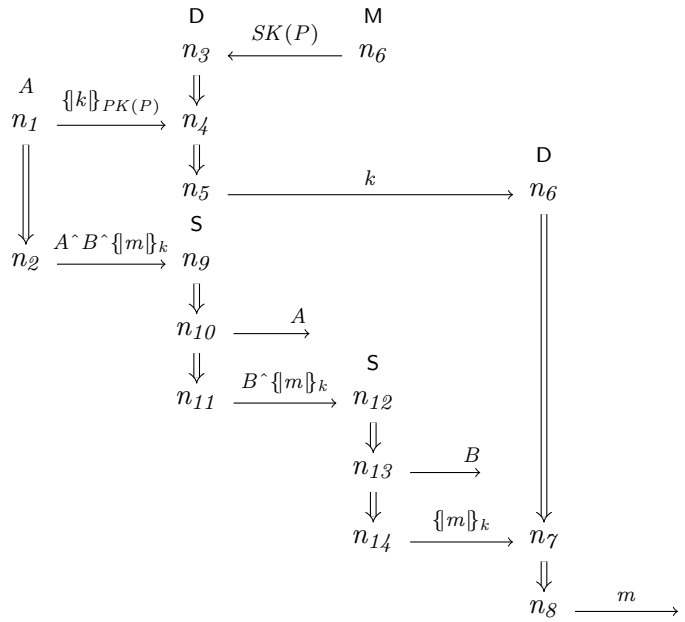
Lemma 4.16. Let $\hat{\mathcal{B}}$ be a high-level bundle. Then $\hat{\mathcal{B}}$ is equivalent to a bundle that contains no TX or HJRN strands.

Proof. Clearly, any TX strands can be removed by replacing them with direct message transmission edges, noting that the bundle conditions will still be satisfied. Any HJRN strand is equivalent to a combination of a HJ and a RN strand, in some order. Further, since the channel permits HJRN strands, it follows by Assumption 4.15 strand that both the RN and HJ strand are permitted. Hence, the bundle conditions are satisfied. □

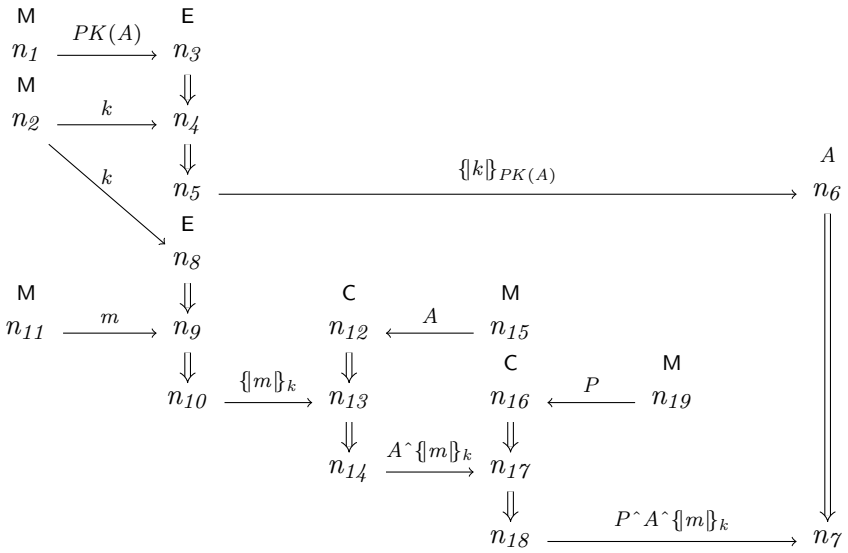
In Proposition 6.22 we use this lemma to justify only considering bundles in the high-level strand space that don't contain HJRN or TX strands.

In order to define the various transport-layer penetrator strands, we will need to identify when a penetrator path is sending or receiving an application-layer message. We can formalise such a definition by using extraction paths, as follows.

Definition 4.17. We say that a destructive penetrator path p_d starting at a node n *extracts the application message from* n, written p_d app-extracts n, iff



(a) A bundle containing a path that extracts the application-message from n_1.



(b) A bundle containing a path that packages the application-message of n_1.

Figure 4.4: A low-level bundle in which the penetrator sends and receives application-layer messages using a simple transport-layer protocol.

$\mathsf{expath}\,(p_d) = \mathsf{expath}^{\mathsf{c}}\,(chan(n))$. A constructive penetrator path p_c ending at a node n *packages the application message of* n, denoted p_c app-packages n, iff $\mathsf{expath}^{\sim}\,(p_c) = \mathsf{expath}^{\mathsf{c}}\,(chan(n))$.

For example, in Figure 4.4a, $\langle n_2, n_9, n_{11}, n_{12}, n_{14}, n_7, n_8\rangle$ app-extracts n_2 whilst in Figure 4.4b, $\langle n_{11}, n_9, n_{10}, n_{13}, n_{14}, n_{17}, n_{18}\rangle$ app-packages n_7.

Consider the low-level penetrator subpaths that correspond to the normal high-level penetrator strands, i.e. HJ, RN, TX, HJRN strands. All of these penetrator strands do not alter, or even inspect the enclosed application-layer message. Therefore, a low-level penetrator subpath that corresponds to one of these strands should, again, not extract the application-layer message from the transport-layer packaging. We formalise this as follows.

Definition 4.18. A normal penetrator path p *transports the application-layer message* iff $p(1), p(|p|) \in \mathcal{N}_{trpt}$, $chan(p(1)) = chan(p(|p|))$, and there exists p_d, and p_c such that $p = p_d \hat{} p_c$, p_d is destructive, p_c is constructive and $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c) \leq \mathsf{expath}^{\mathsf{c}}\,(chan(p(1)))$.

The above definition ensures that the penetrator can extract the application-layer message, but cannot divide it further.

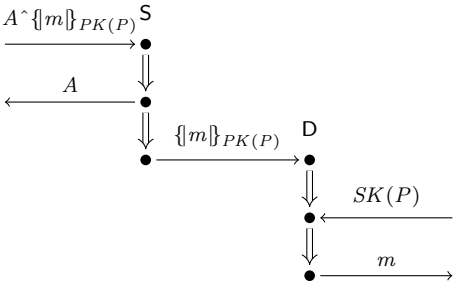
We now define how low-level penetrator subpaths correspond to high-level penetrator strands.

Definition 4.19 (Based on [KL10])**.** Let p be a penetrator subpath in a low-level bundle $\mathcal{B}$, starting at a node n and ending at a node n'. p is a *transport-layer penetrator subpath* iff it is of one of the following forms: (these subpaths are illustrated in Figure 4.5):

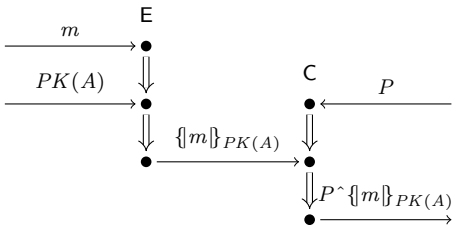
Receive
1. n is a positive regular node and n' is a negative penetrator node;
2. p is destructive;
3. $n \in \mathcal{N}_{trpt}$;
4. $sender(msg(n)) \in \mathcal{I}^{reg}$;
5. $recipient(msg(n)) \in \mathcal{I}^{pen}$;
6. p app-extracts n.

Learn
1. n is a positive regular node and n' is a negative penetrator node;
2. p is destructive;
3. $n \in \mathcal{N}_{trpt}$;
4. $sender(msg(n)) \in \mathcal{I}^{reg}$;
5. $recipient(msg(n)) \in \mathcal{I}^{reg}$;
6. p app-extracts n.

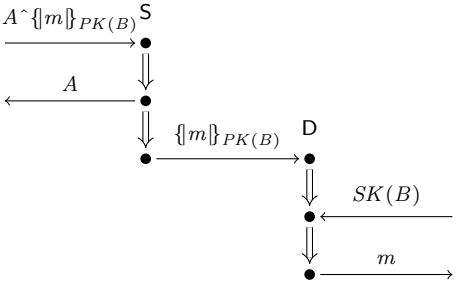
Fake
1. n is a positive penetrator node and n' is a negative regular node;
2. p is constructive;
3. $n' \in \mathcal{N}_{trpt}$;
4. $sender(msg(n')) \in \mathcal{I}^{reg}$;



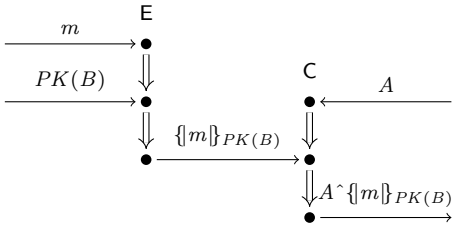
(a) A Receive subpath for $A \in \mathcal{T}^{reg}_{names}$, $P \in \mathcal{T}^{pen}_{names}$.



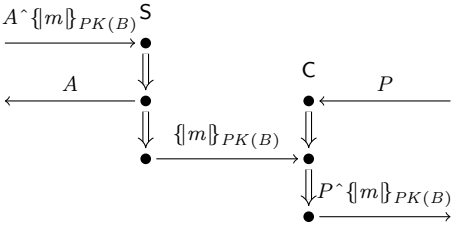
(b) A Send subpath for $A \in \mathcal{T}^{reg}_{names}$, $P \in \mathcal{T}^{pen}_{names}$.



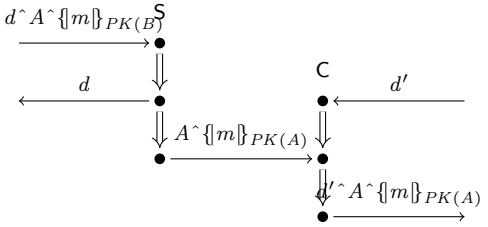
(c) A Learn subpath for $A, B \in \mathcal{T}^{reg}_{names}$, assuming that the penetrator obtains $SK(B)$ somehow.



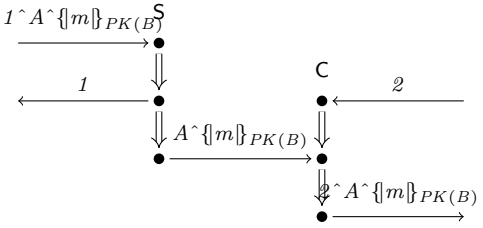
(d) A Fake subpath for $A, B \in \mathcal{T}^{reg}_{names}$.



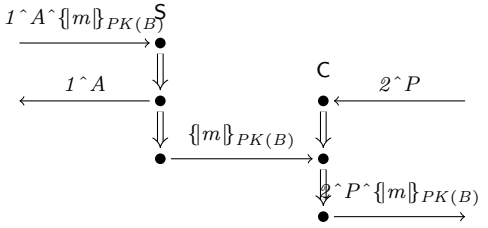
(e) A Hijack subpath for $A \in \mathcal{T}^{reg}_{names}$, $P \in \mathcal{T}^{pen}_{names}$.



(f) A Transmit subpath, assuming that the running example has been augmented with nonces.



(g) A Renumber subpath, assuming that the running example has been augmented with sequence numbers.



(h) A Hijack-Renumber subpath, assuming that the running example has been augmented with sequence numbers.

Figure 4.5: Various penetrator subpaths from Definition 4.19 using the running example of a secure transport protocol.

5. $recipient(msg(n')) \in \mathcal{I}^{reg}$;
6. p app-packages n'.

Send
1. n is a positive penetrator node and n' is a negative regular node;
2. p is constructive;
3. $n' \in \mathcal{N}_{trpt}$;
4. $sender(msg(n')) \in \mathcal{I}^{pen}$;
5. $recipient(msg(n')) \in \mathcal{I}^{reg}$;
6. p app-packages n'.

Hijack
1. n is a positive regular node and n' is a negative regular node;
2. p is normal;
3. $n, n' \in \mathcal{N}_{trpt}$;
4. $seqno(msg(n)) = seqno(msg(n'))$;
5. $appmsg(n') = appmsg(n)$;
6. $chan(msg(n')) = chan(msg(n))$;
7. $sender(msg(n)) \neq sender(msg(n'))$ or $recipient(msg(n)) \neq recipient(msg(n'))$;
8. p transports the application-layer message.

Renumber
1. n is a positive regular node and n' is a negative regular node;
2. p is normal;
3. $n, n' \in \mathcal{N}_{trpt}$;
4. $appmsg(n) = appmsg(n')$;
5. $chan(msg(n)) = chan(msg(n'))$;
6. $sender(msg(n)) = sender(msg(n'))$;
7. $recipient(msg(n)) = recipient(msg(n'))$;
8. $\_ \neq seqno(msg(n)) \neq seqno(msg(n')) \neq \_$;
9. p transports the application-layer message.

Hijack-Renumber
1. n is a positive regular node and n' is a negative regular node;
2. p is normal;
3. $n, n' \in \mathcal{N}_{trpt}$;
4. $appmsg(n') = appmsg(n)$;
5. $chan(msg(n')) = chan(msg(n))$;
6. $sender(msg(n)) \neq sender(msg(n'))$ or $recipient(msg(n)) \neq recipient(msg(n'))$;
7. $\_ \neq seqno(msg(n)) \neq seqno(msg(n')) \neq \_$;
8. p transports the application-layer message.

Transmit
1. n is a positive regular node and n' is a negative regular node;

2. p is normal;
3. $n, n' \in \mathcal{N}_{trpt}$;
4. $appmsg(n) = appmsg(n')$;
5. $chan(msg(n)) = chan(msg(n'))$;
6. $seqno(msg(n)) = seqno(msg(n'))$;
7. $sender(msg(n)) = sender(msg(n'))$;
8. $recipient(msg(n)) = recipient(msg(n'))$;
9. p transports the application-layer message.

Remark 4.20. As an example of why the Transmit subpath is needed consider the transport layer protocol that encodes the high-level term (A, B, m, c) as $d \hat{} A \hat{} \{|m|\}_{PK(B)}$ where d is a nonce. Clearly, this can be transformed by the penetrator to $d' \hat{} A \hat{} \{|m|\}_{PK(B)}$ where $d \neq d'$. However, observe that the corresponding high-level term is unchanged and therefore, since we do not wish to remove penetrator subpaths entirely from the high-level bundle, there needs to be a penetrator subpath that reflects this.

We can now define the second map involving penetrator nodes, which maps nodes on penetrator paths to nodes on high-level penetrator strands. Note that the map cannot be a function since a subsequence of a penetrator subpath could be part of two different penetrator subpaths. For example, consider the penetrator subpath in Figure 4.3b; if the penetrator wanted to send the message from two different penetrator identities then he could create another C strand that would receive from the E strand and therefore the first few nodes of the penetrator subpaths are shared.

Definition 4.21 (Based on [KL10])**.** A *transport-layer penetrator node map* is a relation $\hat{\beta}_2$ between the penetrator nodes of $\mathcal{B}$ and $\hat{\mathcal{B}}$ that:

1. Maps the first and last penetrator nodes of Send, Receive, Learn, Fake, Hijack, Renumber, Hijack-Renumber and Transmit subpaths to the first and last nodes of high-level SD, RV, LN, FK, HJ, RN, HJRN and TX strands respectively, and relates no other nodes; and
2. $\hat{\alpha}_{\mathcal{B}}(msg(n)) = msg(\hat{\beta}_2(n))$ for $(n, \hat{n}) \in \hat{\beta}_2$.

For example, again considering the bundle from Figure 4.3b, a suitable transport-layer penetrator node map would map n_6 and n_{13} to $\hat{n}_6$ and $\hat{n}_{13}$ respectively.

Remark 4.22. The above definition illustrates the need for HJRN strands. For example, consider a Hijack-Renumber subpath which, if we did not have HJRN strands, would need to be mapped to a HJ strand followed by a RN strand, or vice-versa. The above definition would insist that there exists an intermediate low-level transport-layer node that contained the original transport term, but with only one of the hijacking or the renumbering done. However, such a node does not necessarily exist as in a normal bundle the penetrator would do both transformations without reconstructing an intermediate transport-layer term.

For example, consider the transport-layer protocol that represents (A, B, i, m, c) as $A \hat{} B \hat{} \{|\{|m|\}_{PK(B)} \hat{} i|\}_{SK(A)}$. Clearly there exists a penetrator path that consists of a Hijack subpath (changing the sending identity to one

from $\mathcal{I}^{pen}$) followed by a Renumber subpath that includes an explicit intermediate node. However, note that the above path would be equivalent to one that obtains the term $\{|m|\}_{PK(B)}$ using S and D strands and then uses C and E strands to construct a term $P \hat{} B \hat{} \{|\{|m|\}_{PK(B)} \hat{} i'|\}_{SK(P)}$. Hence, there exists no intermediate node: the hijack and renumber happen simultaneously.

Using the above definitions in conjunction with those from Section 4.2 it is now possible to define a node map for a bundle $\mathcal{B}$ that maps the regular and penetrator nodes to nodes of a high-level bundle. In particular, the map explains how each high-level node and strand corresponds to low-level nodes and strands.

Definition 4.23 (From [KL10])**.** A *node map* is a partial relation $\hat{\psi}$ between the nodes of $\mathcal{B}$ and $\hat{\mathcal{B}}$, such that:

1. $\hat{\psi}$ is of the form $\hat{\phi} \cup \hat{\beta}_1 \cup \hat{\beta}_2$, where $\hat{\phi}$, $\hat{\beta}_1$ and $\hat{\beta}_2$ are as in Definitions 4.12, 4.13, 4.21 respectively; and
2. $\hat{\psi}$ is surjective onto the nodes of $\hat{\mathcal{B}}$; and
3. $\hat{\psi}$ respects the strand structure:
   (a) if $n, n' \in dom(\hat{\psi})$ and $n \rightarrow_{\mathcal{B}} n'$ then $\exists \hat{n}, \hat{n}' \cdot n \; \hat{\psi} \; \hat{n} \wedge n' \; \hat{\psi} \; \hat{n}' \wedge \hat{n} \rightarrow_{\hat{\mathcal{B}}} \hat{n}'$; and
   (b) if $\hat{n} \rightarrow_{\hat{\mathcal{B}}} \hat{n}'$, then $\exists n, n' \cdot n \; \hat{\psi} \; \hat{n} \wedge n' \; \hat{\psi} \; \hat{n}' \wedge n \rightarrow_{\mathcal{B}} n'$.

For example, in Figure 4.3b, the node map is equivalent to the regular node map since it contains penetrator nodes. Equally, in Figure 4.3 the node map is equivalent to $\hat{\beta}_1 \cup \hat{\beta}_2$ since there are no regular nodes.

Note that in some cases there may be multiple, equally valid, node maps from a low-level bundle. For example, a Receive subpath followed by a Send subpath could be mapped to a RV and SD strand. However, providing the receive and send are both on the same channel this could be viewed as a Hijack subpath, and thus mapped to a HJ strand.

4.4 Interference Freedom and Abstract Correctness

In this section we consider the conditions under which we can safely abstract away from the transport-layer behaviour. In particular, we develop a condition that holds whenever a term t is constructed by the penetrator in a way that can be mirrored in a high-level bundle. Further, we describe a bundle condition, *interference-freedom*, that is true whenever the bundle can be safely *abstracted*, in the following sense.

Definition 4.24 (Based on [KL10])**.** A high-level bundle $\hat{\mathcal{B}}$ *abstracts* a low-level bundle $\mathcal{B}$ iff there exists a node map $\hat{\psi}$ between the nodes of $\mathcal{B}$ and $\hat{\mathcal{B}}$. A low-level bundle $\mathcal{B}$ is *abstractable* iff there is a high-level bundle $\hat{\mathcal{B}}$ that abstracts $\mathcal{B}$. A high-level strand space $\hat{\Sigma}$ *abstracts* a low-level strand space Σ iff every low-level bundle $\mathcal{B}$ of Σ is abstractable to a bundle of $\hat{\Sigma}$.

In Section 4.4.1 we consider under what conditions we can safely abstract a low-level bundle. In particular, we define what it means for a bundle to be interference-free. In order to define this we also give an axillary definition that decides when a

penetrator node can be safely abstracted; we term such nodes *abstractly constructible* (in the sense that the message on the node can be constructed after abstracting away from the transport protocol). In Section 4.4.2 we briefly consider how the restrictions on high-level transport-layer penetrator strands, such as $\mathcal{AC}$, impact the bundles that we can safely abstract. Lastly, in Section 4.5 we prove our main result and show that any interference-free bundle is abstractable, in the above sense.

4.4.1 Interference Freedom

When proving that a low-level bundle is abstractable we will need to ensure that application-layer messages constructed by the penetrator are constructed only from terms that he can obtain in the high-level model, otherwise the bundle cannot be abstracted. For example, suppose the penetrator takes a nonce sent by an honest agent in the key establishment phase of the transport-layer protocol and replays it within an application-layer message. Clearly, this case has to be disallowed since the node providing the nonce will not exist in the abstracted bundle.

In order to define interference-freedom we will require the following auxiliary definition. Informally, a negative low-level node n is *abstractly constructible* precisely when $msg(n)$ is constructed only from:

1. Terms in the penetrator's initial knowledge, obtained from M strands;
2. Terms sent by honest agents on $\perp$;
3. Terms sent by honest agents on non-$\perp$ channels that are unpacked by the penetrator.

Thus, if a negative node is abstractly constructible it follows that all the nodes that are required to construct the message will be in the abstracted bundle.

Note that in all subsequent definitions and lemmas we assume, without loss of generality, that all bundles are normal in order to simplify the definitions and proofs. In the following an *initial* penetrator subpath is one that either starts at an initial penetrator node (i.e. at a M strand), or starts at a regular node.

In order to define abstractly constructible, we firstly need to define what it means for a penetrator path to contribute to the message of a node n'. In particular, a penetrator path that merely provides a key to decrypt a transport-layer message whose application-layer content is later used in constructing $msg(n')$ should not be considered as contributing to $msg(n')$.

Definition 4.25. A penetrator subpath p *directly-contributes* iff, for every node n on p that lies on the key edge of a D strand, there exists a destructive penetrator subpath p_d ending at n such that:

1. p_d does not traverse a key edge or contain a KG strand; and
2. $p_d(1) \in \mathcal{N}_\perp$; or $p_d(1) \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p_d(1))) \leq \mathsf{expath}(p_d)$.

Thus, the above definition identifies penetrator paths that only provide a key to decrypt a transport-layer message. Using the above, we can now define what it means for penetrator paths and nodes to be abstractly constructible.

Definition 4.26. Let $\mathcal{B}$ be a normal low-level bundle. An initial penetrator subpath p is *abstractly constructible* iff either:

1. p starts at a M strand; or
2. p starts at a regular node $n_1 \in \mathcal{N}_\perp$; or
3. p starts at a regular node $n_1 \in \mathcal{N}_{trpt}$ and there exists a destructive penetrator path $p_d \leq p$ such that p_d $\mathsf{app\text{-}extracts}$ n_1.

A penetrator node $n_2 \in \mathcal{B}$ is *abstractly constructible* iff every initial penetrator path p that ends at n_2 and directly-contributes is abstractly constructible. A regular node $n_2 \in \mathcal{B}$ is *abstractly constructible* iff $n_2 \in \mathcal{N}_\perp$.

The above definition intuitively states that if a node is abstractly constructible then every penetrator path that reaches it either starts at an initial penetrator node, or starts with a message receiving path. In particular, if a node is abstractly constructible we should be able to safely abstract it, as it does not depend in any way on the transport-layer: it depends only on application-layer messages.

As an example, consider Figure 4.6a which uses the running example as a transport-layer protocol and sends the application-layer message $(P_?,\ B_?,\ U,\ c)$ to n_{13}. In this figure, n_9 is abstractly constructible because the only penetrator path to n_9 starts at a M strand. However, n_5 is not abstractly constructible since $n_1 \in \mathcal{N}_{trpt}$ and thus the penetrator subpath $\langle n_1, n_2, n_3, n_5 \rangle$, is not abstractly constructible. Intuitively, this is because the penetrator is taking a value U from the transport-layer packaging, rather than unpacking the message contents.

We prove a simple consequence of the abstractly constructible definition, which is easier to apply.

Lemma 4.27. Let $\mathcal{B}$ be a normal bundle. If a penetrator node $n_2 \in \mathcal{B}$ is *abstractly constructible* then either:

1. There exists a destructive penetrator path p ending at n_2 and starting at a regular transport-layer node $n_1 \in \mathcal{N}_{trpt}$ such that p $\mathsf{app\text{-}extracts}$ n_1; or
2. There exists a regular node $n_1 \in \mathcal{N}_\perp$ such that $n_1 \rightarrow n_2$; or
3. n_2 is a negative node on a penetrator strand such that there exists an abstractly constructible node n_1 where $n_1 \rightarrow n_2$; or
4. n_2 is a positive node on a penetrator strand such that every negative node on the strand is abstractly constructible.

Proof. Let n_2 be abstractly constructible and suppose that cases (1) and (2) do not apply. If n_2 is a negative node then, by Definition 4.26, it follows that every directly-contributing penetrator path $p \hat{} \langle n_1 \rangle \hat{} \langle n_2 \rangle$ ending at n_2, is abstractly constructible. Thus, every directly-contributing penetrator path ending at n_1 is also abstractly constructible and hence n_1 is abstractly constructible, meaning that n_2 satisfies clause (3).

Otherwise, n_2 is a positive node. Therefore, let $n_1 \Rightarrow^+ n_2$ be a negative node on n_2's strand, if one exists. Note that since n_2 is abstractly constructible every

penetrator path of the form $p^\frown\langle n_1\rangle^\frown\langle n_2\rangle$ must be abstractly constructible. Thus, every penetrator path that ends at n_1 must also be abstractly constructible, and therefore n_1 is abstractly constructible. Hence n_2 satisfies clause (4). □

We now define what it means for a bundle to be *interference-free*. The intention is that this property is true precisely when a low-level bundle can be safely abstracted, in the sense that high-level terms are not constructed from transport-layer terms. (Note that this definition differs from that given in [KL10] in order to simplify it and also support regular nodes that send messages over $\perp$.)

Definition 4.28. Let $\mathcal{B}$ be a normal low-level bundle. A negative regular application-layer node $n_2 \in \mathcal{N}_{payload}$ is *interference-free* iff either:

1. $n_2 \in \mathcal{N}_\perp$ and there exists an abstractly constructible node n_1 such that $n_1 \rightarrow n_2$; or
2. $n_2 \in \mathcal{N}_{trpt}$ and there exists a penetrator subpath p ending at n_2 such that either:
   (a) p is normal, p starts at a regular node $n_1 \in \mathcal{N}_{trpt}$, $appmsg(n_1) = appmsg(n_2)$, $chan(n_1) = chan(n_2)$ and p transports the application-layer message; or
   (b) p is constructive and starts at an abstractly constructible node n_1 such that p app-packages n_2.

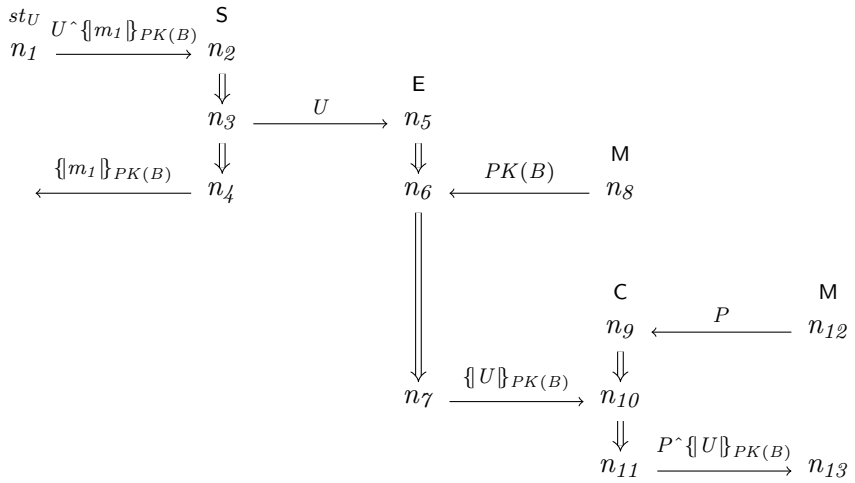
A bundle $\mathcal{B}$ is *interference-free* iff every negative regular application-layer node is interference-free.

Thus, a negative regular node n_2 is interference-free providing the penetrator has constructed any messages that are sent to n_2 in a way that can be safely abstracted. In particular, this means that the penetrator must either:

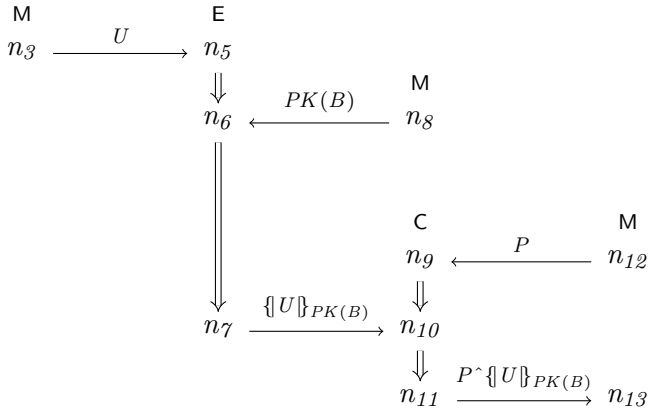
- Hijack, renumber, or simply alter the packaging of an existing message (case (2a)); or
- Send a message that has been properly constructed (i.e. is generated at an abstractly constructible node) via a Send or a Fake subpath (case (2b)), or directly on the $\perp$ channel.

The above definition restricts several kinds of attacks. In particular, it disallows penetrator paths where the penetrator transforms a transport-layer term from one channel into one for another channel, without fully decrypting the application layer message (a *multi-channel* attack). It also prevents the penetrator from sending transport-layer terms as application-layer messages (a *multi-layer* attack — e.g. Figure 4.6a). Clearly, we have to prohibit both of these cases: otherwise the bundle cannot be safely abstracted as there would be attacks that depend on the implementation of the transport-layer protocol. See Figure 4.6 for an example.

Whilst the above definition is a semantic condition, we define a statically-checkable condition in Chapter 5 that implies interference freedom.



(a) n_{13} is not interference-free as there is no penetrator subpath that satisfies condition (1) or (2) of Definition 4.28. In particular, the penetrator path that sends the application-layer message (i.e. $\langle n_3, n_5, n_7, n_{10}, n_{11}, n_{13} \rangle$) starts at a non-abstractly constructible node n_3. This is because it is taking a term, U, from a non-application layer source (that will not be in the high-level bundle) and attempting to send it as an application-layer message.



(b) This bundle is interference-free as n_3 is now abstractly constructible (as it lies on a M strand).

Figure 4.6: Two bundles that illustrate abstractly constructible and interference-freedom.

4.4.2 Abstract Correctness

The channel properties that were defined in Section 3.1.2, such as $\mathcal{AC}$, require that certain high-level strands are not contained in the high-level strand space. Therefore, when abstracting low-level bundles we must take care to ensure that we only abstract bundles that do not contain any prohibited penetrator behaviours. For example, if the transport protocol was bilateral TLS then the high-level strand space contains no high-level LN, FK, HJ or RN strands on the bilateral TLS channel. Therefore, we can only abstract low-level bundles that contain no Learn, Fake, Hijack, Renumber or Hijack-Renumber subpaths.

Unfortunately, this is too strong. Recall that a RV strand followed by a SD strand is actually equivalent to a HJ strand where the recipient of the first message and the sender of the second message are penetrator identities. Hence, since the penetrator is always allowed to send and receive messages, it follows that some Hijack subpaths will be present for even the most secure transport-layer protocols. As a result, the high-level bundles that we abstract to will contain some prohibited penetrator strands, but only if they are equivalent to combinations of permitted strands. This is addressed in Proposition 4.31, but we formalise the type of penetrator subpath that is problematic as follows.

Definition 4.29. A Hijack or Hijack-Renumber subpath p is an *innocuous penetrator subpath* iff $recipient(p(1)) \in \mathcal{I}^{pen}$ and $sender(p(|p|)) \in \mathcal{I}^{pen}$. An innocuous penetrator subpath p is *expanded* iff it is the concatenation of Receive and Send subpaths.

We now define a property of low-level bundles that is true iff the bundle contains only penetrator behaviours that are permitted by the channel definitions. Whilst this could be considered as a property of strand spaces, we define it as a property of bundles as in Chapter 5 we will have to consider strand spaces in which only some of the bundles satisfy this property.

Definition 4.30. A channel $c \in Channels \setminus \{\bot\}$ is *abstractly correct* in a low-level bundle $\mathcal{B}$ iff every penetrator subpath of c in $\mathcal{B}$ either corresponds to a penetrator strand allowed by definition of c, or is an innocuous subpath. A bundle $\mathcal{B}$ is *abstractly correct* iff every channel $c \in Channels \setminus \{\bot\}$ is abstractly correct in $\mathcal{B}$. A strand space Σ is *abstractly correct* iff every bundle of Σ is abstractly correct.

Observe that equivalence of bundles does not not necessarily imply that both are abstractly correct. For example, consider a very simple bundle that contains two regular transport-layer nodes and a direct message edge between them. Clearly such a bundle is trivially abstractly correct. However, the transport protocol in question may be fundamentally insecure, and the bundle may be equivalent to one in which the penetrator concatenates a Learn subpath and a Fake subpath. This is of no consequence in this section, but does affect our proofs in Chapter 5.

4.5 Soundness of The Abstraction

In this section we finally prove that any interference-free low-level bundle is abstractable. Note that the resulting high-level bundle will contain TX and HJRN strands. We deal with this in Proposition 6.22. This proof is based on the proof given in [KL10].

Proposition 4.31. Let $\mathcal{B}$ be a normal, abstractly correct, interference-free bundle such that every innocuous penetrator subpath is expanded. Then $\mathcal{B}$ is abstractable by a high-level bundle $\hat{\mathcal{B}}$.

Proof. Let $\mathcal{B}$ be a normal, abstractly-correct, interference-free bundle in Σ such that every innocuous penetrator subpath is expanded. We construct a high-level bundle $\hat{\mathcal{B}}$ that abstracts $\mathcal{B}$. Firstly, the regular strands of $\hat{\mathcal{B}}$ are constructed from those of $\mathcal{B}$ according to Definition 4.12.

In order to make $\hat{\mathcal{B}}$ a bundle we need to construct sufficient penetrator strands such that there are no negative regular nodes with no incoming edges (henceforth *lonely* nodes). We follow the approach of Kamil and Lowe and show that for each negative regular application layer node $n_2 \in \mathcal{N}_{payload}$ in $\mathcal{B}$ there exists a penetrator path p that ends at n_2 and that maps to a high-level penetrator path that ends at $\hat{\phi}(n_2)$ in $\hat{\mathcal{B}}$.

Consider the negative node n_2. As $\mathcal{B}$ is interference-free there are three cases to consider, following Definition 4.28:

Case (1) $n_2 \in \mathcal{N}_\perp$ and hence, there exists a positive abstractly constructible node n_1 such that $n_1 \rightarrow n_2$.

Case (2a) $n_2 \in \mathcal{N}_{trpt}$ and there exists a normal penetrator path p, starting at a regular node $n_1 \in \mathcal{N}_{trpt}$ and ending at n_2 such that $chan(n_1) = chan(n_2)$ and p transports the application-layer message. If p is an innocuous penetrator subpath, it follows by the assumptions of this lemma that p must be expanded, and therefore, we choose to consider this path in Case (2b) instead. Otherwise, by Assumption 4.10 it follows that:

- $name(sender(msg(n_1))) = ?$ iff $name(sender(msg(n_2))) = ?$;
- $end(sender(msg(n_1))) = ?$ iff $end(sender(msg(n_2))) = ?$;
- $name(recipient(msg(n_1))) = ?$ iff $name(recipient(msg(n_2))) = ?$;
- $end(recipient(msg(n_1))) = ?$ iff $end(recipient(msg(n_2))) = ?$;
- $seqno(msg(n_1)) = \_$ iff $seqno(msg(n_2)) = \_$.

Therefore, according to Definition 4.19, p must either be a Transmit, Hijack, Renumber or Hijack-Renumber subpath and hence, by Definition 4.21, it can be mapped to a TX, HJ, RN or HJRN strand that ends at $\hat{\phi}(n_2)$.

Case (2b) There exists a constructive penetrator subpath p, starting at a positive abstractly constructible node n_1, and ending at n_2, such that p app-packages n_2. Hence, this subpath fits the definition of a Send or Fake subpath (noting that $recipient(msg(n_2)) \in \mathcal{I}^{reg}$ as n_2 is a negative regular node) and thus gets mapped to a SD or FK strand that ends at $\hat{\phi}(n_2)$.

We now show that the positive abstractly constructible node n_1 from cases (1) and (2b) above maps to a non-lonely node. If n_1 is regular then by Definition 4.26 it follows that $n_1 \in \mathcal{N}_\perp$ and hence gets mapped to $\hat{\phi}(n_1)$ in $\hat{\mathcal{B}}$. Otherwise, n_1 is a penetrator node and thus there are the following cases of Lemma 4.27 to consider.

Case (1) There exists a destructive penetrator path p_d ending at n_1 and starting at a regular application-layer node $n_0 \in \mathcal{N}_{trpt}$ such that p_d app-extracts n_0. Therefore, as $sender(msg(n_0)) \in \mathcal{I}^{reg}$ (since n_1 is regular), p_d fits the definition of a Learn or Receive subpath and thus can be mapped to either a LN or RV strand that ends at $\hat{\phi}(n_1)$.

Case (2) There exists a regular node $n_0 \in \mathcal{N}_{\perp}$ such that $n_0 \rightarrow n_1$. Hence, in the high-level bundle this is simply mapped to a transmission edge from $\hat{\phi}(n_0)$ to $\hat{\phi}(n_1)$.

Case (3) This case cannot apply as n_1 is positive.

Case (4) n_1 is a positive node on a penetrator strand st such that every negative node on the strand is abstractly constructible. Therefore, according to Definition 4.13, this penetrator strand will be mapped to one of the same type. Clearly, we now must ensure that the negative nodes on st (if any) are not lonely. This can be proven by inductively applying the argument of this and the previous paragraph to each negative node on this strand.

Hence, it follows that there are no regular or penetrator lonely nodes and therefore, $\hat{\mathcal{B}}$ satisfies the bundle conditions. Further, since $\mathcal{B}$ is abstractly correct it follows that if $\hat{\mathcal{B}}$ contains a penetrator strand p that is prohibited by the channel definitions, then p must be the abstraction of an innocuous penetrator subpath. However, since all innocuous penetrator subpaths are expanded, we have not abstracted any such penetrator subpaths. Hence, $\hat{\mathcal{B}}$ does not violate any channel conditions. Lastly, note that Assumption 3.5, Assumption 3.3 and Equation 3.1 hold by Assumption 4.10 and the definition of $\hat{\alpha}$. Thus, $\mathcal{B}$ is abstractable. □

4.6 Summary

In this chapter we have proven the soundness of the high-level strand spaces model presented in Section 3.1. In particular, we have shown that every bundle that satisfies our independence assumption, defined in Definition 4.28, can be abstracted to a high-level bundle.

In order to prove this result we firstly defined a mapping $\hat{\alpha}$ that converts low-level transport terms into the corresponding high-level terms. Using this we then defined a function $\hat{\phi}$ that relates the low-level regular nodes of a bundle to the corresponding high-level regular nodes. Further, $\hat{\phi}$ abstracts away from the details of how the transport-layer is implemented by not including, for instance, handshake nodes. We then defined two further maps, $\hat{\beta}_1$ and $\hat{\beta}_2$, that relate the low-level penetrator nodes to high-level penetrator nodes. These three maps are all combined into a single map, $\hat{\psi}$, that maps the nodes of a low-level bundle to the corresponding nodes of a high-level bundle. Using this we define what it means for a high-level bundle to abstract a low-level bundle.

We then defined our main semantic condition. In order to do this we firstly defined what it means for a node to be abstractly constructible, which is true whenever a node's message is built from only application-layer values. Intuitively, this means

that the node can be safely abstracted. Using this, we then defined our main semantic assumption, interference freedom, before proving that whenever a bundle is interference free, it is abstractable.

As noted in the introduction, the work in this chapter is based on that in [KL10, Kam10]. In particular, Section 4.2 and Section 4.3 are substantially based on [KL10, Kam10], whilst the remaining sections have been entirely reworked. Compared to the work of Kamil and Lowe, the proof in this chapter adds support for our expanded model, which includes both RN strands and support for unilaterally authenticating secure transport protocols. It also adds support for application-layer protocols that send some messages over $\perp$. Further, the independence assumption (i.e. as presented in Section 4.4.1) has been entirely reformulated in order to increase its applicability, and to clarify a number of hidden assumptions. We have also modified the way the transport-layer protocols are defined in the low-level strand spaces model. In particular, we have defined what extraction paths are, and have redefined many of the properties in terms of extraction paths. This has made a number of the definition more precise. Lastly, we have made explicit a number of hidden assumptions. Most notably, we have made explicitly clear, via the definition of abstract correctness, what it means for a low-level transport-layer protocol to be *correct*.

We discuss related work at the end of Chapter 7, in Section 7.4.

Chapter 5

Disjoint Encryption

In the previous chapter we proved that any interference-free bundle can be safely abstracted. One problem with the definition of interference freedom is that it is *semantic* and therefore it is difficult to prove that all bundles modelling the protocol satisfy it. What would be preferable is a *statically-checkable* condition, based on the transport and application-layer messages, that could decide if a given bundle is interference-free. In this chapter we address this problem by introducing a new statically-checkable condition based on disjoint encryption [GT00]. We then prove that if a strand space satisfies our condition, then any bundle of the strand space can be transformed to an equivalent bundle that is interference-free.

One of the main challenges that we have to address is the presence of paths in the low-level bundle that transform some value that is part of the transport-layer packaging or handshake (say a nonce, or a name) and send it as part of an application-layer message. We term such penetrator paths *crossing-paths*. These paths prevent us from safely abstracting such bundles as the paths are not abstractly constructible. However, under the assumptions that we define in this chapter, it turns out that any crossing-paths that do exist are actually superfluous, in that an equivalent bundle exists that contains no crossing-path.

The outline of the proof in this chapter is as follows. Firstly, we assume that the low-level strand space satisfies a number of (statically checkable) assumptions relating to various disjointness conditions. Then, we detail the construction of a new strand space that is related to the previous strand space, but in which the penetrator has been given a larger set of initial knowledge (this set will not contain any values of relevance to the application layer). Intuitively, this is required because the penetrator may, for instance, transfer transport-layer nonces into the application-layer. Therefore, as we cannot model such a transfer in the high-level bundle, we instead assume that the penetrator knows such values initially. We then prove that any bundle in this new strand space can be transformed to an equivalent bundle that contains no crossing-paths. Then, we prove the main result and show how to transform a crossing-path-free bundle into an equivalent one that is interference-free, which we do as a number of separate, but composeable, bundle transformations. This allows us to prove the high-level strand spaces model is sound, since any bundle that contains an attack is equivalent to an abstractable bundle that will also contain the attack. We formalise and prove this in Chapter 6.

We begin in Section 5.1 by defining a few preliminaries; in particular we define

several sets of encryptions (for the disjointness condition) and a type of penetrator path that are used by the penetrator to send application-layer messages. We also introduce a new, stronger, form of bundle equivalence that preserves the satisfaction of application-layer correctness properties. Then, in Section 5.2 we define nine different assumptions that are required to prove our main result. In Section 5.3 we consider crossing-paths and show how we can remove them. This requires the proof of several technical results. In Section 5.4 we prove the main result of this chapter and prove that any low-level bundle can be transformed into an abstractable low-level bundle in such a way that it preserves any attacks.

5.1 Preliminaries

In this section we briefly outline a few extra definitions and results that will be needed in the following subsections. In Section 5.1.1 we further split up the set of terms that occur in transport-layer messages. This then allows us to define various sets of encryptions that occur in particular contexts, which we use when defining our main assumptions. In Section 5.1.2 we define a new type of penetrator path, a *penetrator message-construction* path that helps to identify which penetrator nodes are manipulating application-layer messages. We then show in Section 5.1.3 that bundle equivalence does not necessarily preserve the correctness of application-layer correctness properties, and introduce a stronger type of bundle equivalence that does preserve such correctness properties.

5.1.1 Encryption Sets

In the following sections we will need to know several sets of terms.

Definition 5.1. The set of application-layer messages sent over the unprotected channel, $\perp$, is defined by $\mathcal{T}_{app}^{\perp} \mathrel{\widehat{=}} \{msg(n) \mid n \in \mathcal{N}_{\perp}\}$.

The following sets of terms are defined for each channel $c \in \mathit{Channels} \setminus \{\perp\}$.

- The set of *application-layer terms*, $\mathcal{T}_{app}^{c} \subseteq \mathcal{A}$ defined as:

$$\mathcal{T}_{app}^{c} \mathrel{\widehat{=}} \{extract(t, \mathsf{expath}^{\mathsf{c}}(c)) \mid t \in \mathcal{T}_{payload}^{c}\}.$$

- The (subterm closed) set of *transport-layer non-message terms*, $\mathcal{T}_{non\text{-}msg}^{c}$, that contains all non-application layer terms that appear in transport terms, is defined as:

$$\begin{aligned}\mathcal{T}_{non\text{-}msg}^{c} \mathrel{\widehat{=}} \{&extract(t, es) \mid t \in \mathcal{T}_{payload}^{c} \wedge es \in \mathcal{EP} \\ &\wedge \mathsf{expath}^{\mathsf{c}}(c) \not\leq es \wedge es \not\leq \mathsf{expath}^{\mathsf{c}}(c)\} \\ \cup \{&s \mid t \in \mathcal{T}_{non\text{-}payload}^{c} \wedge s \sqsubseteq t\}.\end{aligned}$$

The second line of the definition of $\mathcal{T}_{non\text{-}msg}^{c}$ extracts all subterms of a transport-layer message that do not contain the application-layer message (i.e. $es \not\leq \mathsf{expath}^{\mathsf{c}}(c)$), or occur within the application-layer message (i.e. $\mathsf{expath}^{\mathsf{c}}(c) \not\leq es$).

- The set of *application-layer ingredients*, $\mathcal{A}_{app}$, is defined by[1]:

$$\mathcal{A}_{app} \mathrel{\widehat{=}} X \cup \{k, k^{-1} \mid k \in X \cap \mathcal{K}\}$$
$$\text{where } X \mathrel{\widehat{=}} \{t' \mid t \in \mathcal{T}^c_{app}, c \in \mathit{Channels}, t' \ \mathsf{ingredient}\ t\}.$$

Recalling the running example used in Section 4.1, in the case of Figure 4.2, $\mathcal{T}^c_{app} = \{\mathit{appmsg_2}\}$, $\mathcal{T}^c_{non\text{-}msg} = \{A, B, \{\!|k|\!\}_{PK(B)}, k\}$, $\mathcal{T}^{\perp}_{app} = \{\mathit{appmsg_1}\}$ and $\mathcal{A}_{app} = \{\mathit{appmsg_1}, \mathit{appmsg_2}\}$, assuming $\mathit{appmsg_1}$ and $\mathit{appmsg_2}$ are atomic.

A central part of our disjointness condition will be that certain encryptions cannot be shared amongst different layers. In order to define such a condition we define several sets of encryptions, as follows.

Definition 5.2. The set of all encryptions, denoted by $\mathcal{E}$, is defined as: $\{\{\!|m|\!\}_k \mid \{\!|m|\!\}_k \in \mathcal{A}\}$. The set of *application-layer encryptions for* $\perp$, denoted $\mathcal{E}^{\perp}_{app}$, is defined as $\mathcal{E} \cap \{s \mid t \in \mathcal{T}^{\perp}_{app}, s \sqsubseteq t\}$.

The following are all defined for each channel $c \in \mathit{Channels} \setminus \{\perp\}$.

- The set of *application-layer encryptions*, denoted by $\mathcal{E}^c_{app}$, of all encryptions that appear in an application-layer message:

$$\{e \mid t \in \mathcal{T}^c_{app}, e \sqsubseteq t, e \in \mathcal{E}\}.$$

- The set of *transport-layer message encryptions*, denoted by $\mathcal{E}^c_{trpt}$, of all encryptions that enclose an application-layer message:

$$\{e \mid t \in \mathcal{T}^c_{payload}, e \in \mathcal{E}, es \in \mathcal{EP}, es \leq \mathsf{expath^c}\,(c)\,, e \sqsubseteq_{es} t\}.$$

- The set of *transport-layer non-message encryptions* for a channel c, denoted $\mathcal{E}^c_{trpt\text{-}non\text{-}msg}$:

$$\mathcal{T}^c_{non\text{-}msg} \cap \mathcal{E}.$$

The unions of $\mathcal{E}^c_{trpt}$ and $\mathcal{E}^c_{trpt\text{-}non\text{-}msg}$ over $\mathit{Channels} \setminus \{\perp\}$ are written as $\mathcal{E}_{trpt}$ and $\mathcal{E}_{trpt\text{-}non\text{-}msg}$ respectively. The closure of $\mathcal{E}^c_{app}$ over *Channels* is written as $\mathcal{E}_{app}$.

We now state a simple result that shows that the above encryption sets are total, in that every encryption that occurs on a node is contained within one of the above sets.

Lemma 5.3. Let $\mathcal{B}$ be a bundle and n a regular node in $\mathcal{B}$. Then every encryption $\{\!|m|\!\}_k \sqsubseteq \mathit{msg}(n)$ is a member of at least one of $\mathcal{E}_{app}$, $\mathcal{E}_{trpt}$ and $\mathcal{E}_{trpt\text{-}non\text{-}msg}$.

Proof. Recall Assumption 4.4 states that regular nodes are partitioned between $\mathcal{N}_{\perp}$, $\mathcal{N}_{trpt}$ and $\mathcal{N}_{non\text{-}payload}$. The result then immediately follows from Assumption 4.5 and the definitions of $\mathcal{E}_{app}$, $\mathcal{E}_{trpt}$ and $\mathcal{E}_{trpt\text{-}non\text{-}msg}$. □

[1] In the definition of $\mathcal{A}_{app}$, note it is important that complex keys are symmetric. If this was not the case then $\mathcal{A}_{app}$ would erroneously not include the ingredients required to produce inverse keys. This definition could be adapted if non-symmetric complex keys were desired.

5.1.2 Message Sending

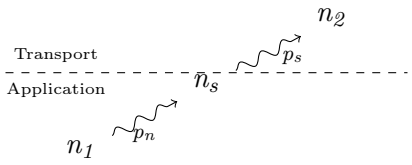
In the following sections we will need to identify which penetrator nodes are manipulating application-layer messages. We term such nodes *application-layer* penetrator nodes. Unfortunately, it is difficult to tell, for instance, if a D or an E strand is manipulating a transport-layer encryption, or an application-layer encryption. We first consider how to identify the nodes that construct application-layer messages. In particular, we will to identify all penetrator paths that manipulate terms that are eventually incorporated into application-layer messages. The following definition identifies when a constructive penetrator path constructs a message and then sends it over a transport channel. For ease we consider only normal bundles.

Note that the following definition is related to Proposition 4.31, as in the proof of this proposition we identified nodes that were manipulating application-layer messages. However, Proposition 4.31 assumed that the bundle was interference-free, so here we require a different method of identifying such nodes.

Definition 5.4. Let $\mathcal{B}$ be a normal low-level bundle. A penetrator path p that starts at a positive regular node n_1 and ends at a negative regular node $n_2 \in \mathcal{N}_{payload}$ is a *penetrator message-construction* path iff either:

1. $n_2 \in \mathcal{N}_{\perp}$ and p is directly contributing; or
2. $n_2 \in \mathcal{N}_{trpt}$, $p = p_n \hat{} p_s$ such that p_n is directly contributing and $\mathsf{expath}^{\mathsf{c}}(chan(n_2)) = \mathsf{expath}^{\sim}(p_s)$.

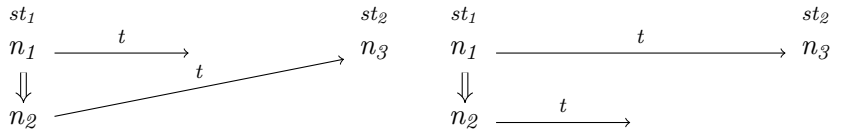
We can visualise the latter type of penetrator message-construction path as follows:



where p_s sends the application-layer message of n_s using a Send or Fake subpath, whilst p constructs the application-layer message, and $p = p_n \hat{} p_s$. Diagrams in this section will be drawn with: penetrator paths drawn as wiggly lines, constructive penetrator paths sloping up, destructive penetrator paths sloping down and a delineation between nodes that are dealing with application-layer content, and those that are dealing with transport-layer content (if such a line can be drawn).

5.1.3 Bundle Correctness Properties

In Chapter 6 we will prove that the high-level strand spaces model is sound, by showing that whenever a low-level bundle does not satisfy a correctness property, a high-level bundle that also does not satisfy the property exists. We will prove this by showing that we can transform (via a number of bundle transformations that we develop in this section) the low-level bundle to a related low-level bundle that is abstractable, but also does not satisfy the correctness property. Then, we prove that if a low-level bundle is abstractable and does not satisfy the correctness property, then its abstraction also does not. Clearly, in order for the above proof to work we



(a) Bundle 1 (b) Bundle 2

Figure 5.1: A bundle illustrating the problem with bundle equivalence.

need the bundle transformations that we develop to preserve the incorrectness of the correctness property. Unfortunately, the standard definition of bundle equivalence is not strong enough to do this.

For example, consider trying to specify that a certain node causally precedes, i.e. using $\preceq$, another node. Since bundle equivalence only requires the regular behaviour to be the same, this relation is not necessarily preserved by bundle equivalence. For example, Figure 5.1 gives two bundles that are equivalent, but such that the $\preceq$ relation is different. Further, the transformations we define will change the $\preceq$ relation: they will remove some penetrator paths that cannot be safely abstracted. Thankfully, in our correctness properties we will only want to write $\preceq$ as a conclusion of an implication, and therefore it will only occur *positively*. Thus, if suffices to ensure that the transformed $\preceq$ relation relates fewer nodes. This will ensure that if the original bundle did not satisfy the correctness property, then the transformed bundle also will not, as required.

We therefore introduce a relation on bundles $\trianglerighteq$ that holds when the bundles are equivalent and the resulting $\preceq$ relation is a restriction of the original relation. We then ensure that all of the transformations in this section output bundles that are related to the original bundle using $\trianglerighteq$.

Definition 5.5. A low-level bundle $\mathcal{B}$ *can be reduced to* $\mathcal{B}'$, denoted $\mathcal{B} \trianglerighteq \mathcal{B}'$, iff $\mathcal{B}$ and $\mathcal{B}'$ are equivalent and $\preceq_{\mathcal{B}'}$ relates no more regular nodes than $\preceq_{\mathcal{B}}$, i.e:

$$\left(\mathcal{N}_{\mathcal{B}}^{reg} \times \mathcal{N}_{\mathcal{B}}^{reg}\right) \cap \preceq_{\mathcal{B}} \supseteq \left(\mathcal{N}_{\mathcal{B}'}^{reg} \times \mathcal{N}_{\mathcal{B}'}^{reg}\right) \cap \preceq_{\mathcal{B}'}.$$

We now prove that any bundle can be reduced to a normal bundle, analogously to Lemma 2.17. This lemma is used instead of Lemma 2.17 throughout the following sections to ensure all bundles satisfy the same correctness properties.

Lemma 5.6. Let $\mathcal{B}$ be a low-level bundle. Then there exists a normal bundle $\mathcal{B}'$ such that $\mathcal{B} \trianglerighteq \mathcal{B}'$.

Proof. The proof of Lemma 2.17 in [GT02] proceeds by removing redundancies, as illustrated in Figure 2.2b, to create an equivalent bundle $\mathcal{B}'$. This transformation ensures that if $n_1 \preceq_{\mathcal{B}'} n_2$ then $n_1 \preceq_{\mathcal{B}} n_2$, as it does not add paths between regular nodes; it merely removes them. Therefore, $\mathcal{B} \trianglerighteq \mathcal{B}'$. □

5.2 Formulating the Assumption

In this section we consider what *statically-checkable* assumptions are necessary in order to prove our main result of this chapter, i.e. that every bundle can be made

interference-free. In particular, we consider what behaviours we need to prohibit in order to safely abstract the bundles of a given strand space. We intersperse the definition of our assumptions with the reasons for their necessity.

The first condition requires that different transport protocols do not share any encryptions that enclose application-layer messages. This ensures that a message for one transport protocol cannot be partially deconstructed and then altered to appear as a message for another transport protocol. In particular, this ensures that any penetrator path starting at a regular transport-layer node on a channel c_1 and ending at a regular transport-layer node on a channel c_2 must traverse a node n such that the application-layer message is unencrypted on n. This is necessary as HJ strands do not allow the channel to be changed, meaning that there is no high-level strand that could represent the the opposite of the above behaviour.

Definition 5.7. Two transport-layer channels $c_1, c_2 \in \mathit{Channels} \setminus \{\bot\}$ satisfy *disjoint message encryption* iff $\mathcal{E}_{trpt}^{c_1} \cap \mathcal{E}_{trpt}^{c_2} = \emptyset$. A set of channels $C \subseteq \mathit{Channels} \setminus \{\bot\}$ satisfies *disjoint message encryption* iff each distinct pair of channels from C satisfies disjoint message encryption.

Using the above definition we can easily state our first assumption.

1. $\mathit{Channels} \setminus \{\bot\}$ satisfies disjoint message encryption.

The next few conditions impose further disjointness conditions on various types of encryptions. Recall that the set of all encryptions is given by $\mathcal{E}_{app} \cup \mathcal{E}_{trpt} \cup \mathcal{E}_{trpt\text{-}non\text{-}msg}$ (cf. Definition 5.1); the next few conditions concern the overlaps that are allowed between these sets. In particular, our second condition ensures that the penetrator cannot send transport-layer messages as application-layer messages. The third condition requires that the penetrator cannot send encryptions that contain application-layer payloads in the transport-layer packaging or as part of the handshake. Clearly if a bundle were to exist that exhibited either of these behaviours, then it would not be possible to make the bundle interference-free as there is no high-level strand to represent such behaviour.

2. No application-layer encryption can also be used as a transport-layer encryption, i.e. $\mathcal{E}_{app} \cap \mathcal{E}_{trpt} = \emptyset$.

3. No non-message transport encryption is also a transport-layer encryption, i.e. $\mathcal{E}_{trpt} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg} = \emptyset$.

Note that we do not require that $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg} = \emptyset$ as there may be values, such as public-key certificates, that can be shared safely between the layers. Instead we require that, for each encryption in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, the inverse key must be public.

4. If a term $\{\!|m|\!\}_k$ is an application-layer encryption and a non-message transport encryption then k^{-1} must be public, i.e. $\forall \{\!|m|\!\}_k \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg} \cdot k^{-1} \in \mathcal{A}_{\mathcal{P}}$.

To see why a condition like this is necessary consider the bundle in Figure 5.2, where $\{\!|t|\!\}_k \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$. In this figure, nodes within dotted boxes are

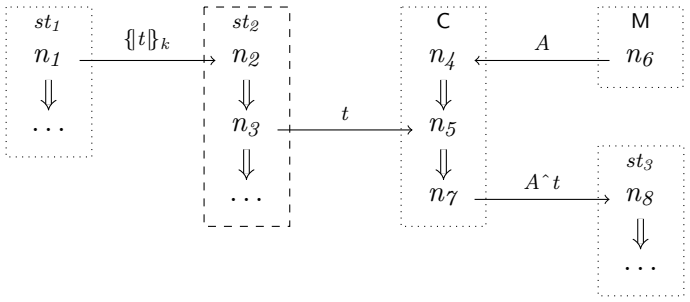


Figure 5.2: A bundle that does not satisfy Assumption 5.9 (4).

application-layer nodes whilst nodes within dashed boxes are handshake nodes (i.e. transport non-message nodes). Observe that the bundle is not interference-free as n_5 is not abstractly constructible (as the penetrator obtains t from a handshake node). However, under the above assumption we can easily transform the bundle to add a path between n_1 and n_5 via a D strand, with the key coming from a M strand (as $k^{-1} \in \mathcal{A}_{\mathcal{P}}$). This condition will be required in the following section when *crossing-paths* are considered.

The alternatives to this are not clear. Whatever transformation we make has to ensure the path between n_1 and n_5 is independent of the transport-layer. Further, it may well be the case that the only paths that can decrypt terms of the form $\{\!|m|\!\}_k$ are in the transport-layer. Therefore, it would appear that in order to remove such paths we need such inverse keys to be public.

The fifth condition ensures that the penetrator cannot possess any transport-layer encryptions in his initial knowledge. This is required as it prevents the penetrator from creating transport-layer messages for which he does not know the application-layer message: clearly there is no high-level strand that permits this.

5. No transport-layer encryptions are initially deducible by the penetrator, i.e. $\mathcal{A}^*_{\mathcal{P}} \cap \mathcal{E}_{trpt} = \emptyset$.

The sixth condition ensures that the penetrator cannot possess non-public application-layer terms surrounded by non-application-layer encryptions. If the penetrator initially knew a non-public term that contained an application-layer term surrounded by some transport-layer encryptions, then there could exist a penetrator path on which he extracts the application-layer term by decrypting using transport-layer keys. However, such a path would not be valid in the high-level bundle and thus we cannot safely abstract the bundle.

6. Non-public application-layer terms must only appear in the penetrator's initial knowledge surrounded by application-layer encryptions, i.e. if $t \in \mathcal{A}_{app} \setminus \mathcal{A}_{\mathcal{P}}$, then for all $s \in \mathcal{A}_{\mathcal{P}}$, if es is an extraction path such that $t \sqsubseteq_{es} s$ then for each $es' < es$ such that $\{\!|m|\!\}_k \sqsubseteq_{es'} s$, $\{\!|m|\!\}_k \in \mathcal{E}_{app}$.

The seventh condition disallows bundles in which an application-layer secret (e.g. a key) originates at both the application-layer and the transport-layer. Such occurrences are likely to be coincidences (and are certainly indicative of poor design) and

may result in the bundle being unable to be abstracted. For instance, suppose a node n originates a value t at both the application-layer and the transport-layer. Further, assume that the application-layer value is unavailable to the penetrator (as it is sent over a confidential channel, or similar) but that the transport-layer value can be obtained by the penetrator. Therefore, the low-level bundle may contain a penetrator path from n to another application-layer node along which the penetrator uses t. However, such a path is not abstractable since it requires a particular configuration of the transport-layer protocol.

7. Atomic or encrypted application-layer values originate only at the application-layer. That is, if a non-public application-layer term $t \in \mathcal{A}_{app} \setminus \mathcal{A}_{\mathcal{P}}$ originates at a regular node n and t is an atom or an encryption, then either $n \in \mathcal{N}_{\perp}$, or $n \in \mathcal{N}_{trpt}$ and for every extraction path es such that $t \sqsubseteq_{es} msg(n)$, $\mathsf{expath}^{\mathsf{c}}(c) \leq es$ (i.e. t originates *only* at the application-layer at n).

The eighth condition disallows bundles in which a transport-layer encryption is also an application-layer ingredient (cf. Definition 2.1). Clearly any such bundle is not abstractable, since the transport-layer encryptions are not available in the abstracted bundle, and thus the key cannot be generated.

8. No transport-layer encryption can be an application-layer ingredient. That is, $\mathcal{A}_{app} \cap \mathcal{E}_{trpt} = \emptyset$.

Note that the above condition actually subsumes the first part of Assumption 5.9 (2); we keep these as separate conditions for clarity.

Lastly, we require that the strand space is abstractly correct. This is required as without this condition we are unable to abstract any bundles as there would be transport-layer behaviours that have no accompanying representation in the high-level bundles. In particular, recall that Proposition 4.31 required a bundle to be abstractly correct in order for it to be abstractable.

9. Σ is abstractly correct.

Definition 5.8. A strand space Σ satisfies *layered-disjoint encryption* iff it satisfies conditions (1)–(9) above.

Assumption 5.9. Σ satisfies layer-disjoint encryption.

Unfortunately, TLS [DR08] does not satisfy the above condition. This is discussed further in Chapter 7.

We now briefly outline how these properties can be checked using static analysis. Assumption 5.9 (1)–(4) and (8) all require various sets of encryptions to be disjoint. The most obvious approach to verify this would be to firstly prove that the sets are disjoint for a single protocol (e.g. for each c, $\mathcal{E}^c_{app} \cap \mathcal{E}^c_{trpt} = \emptyset$), and then that sets for different protocols are disjoint. The latter can be proven correct by using a standard protocol composition result, such as [GT00, DDMR07, CD08, ACG+08, CC10]. The former could be addressed by considering a symbolic definition of the application and transport-layer protocols and considering where encryption keys can be used. For many protocols, including TLS, this should be largely straightforward (since, for

example, TLS encryption keys are not obtainable by the application layer), although it would almost certainly require additional assumptions regarding the secure storage of keys.

Assumption 5.9 (7) could be easily verified given a symbolic definition of the regular agents that included details of where terms originate (i.e. which terms are fresh). Assumption 5.9 (5) and Assumption 5.9 (6) are assumptions on the penetrator's initial knowledge and thus could be verified by considering a symbolic definition of $\mathcal{A}_{\mathcal{P}}$. Lastly, Assumption 5.9 (9) can be verified by, firstly, proving that the transport-layer protocols are abstractly correct in isolation, via a standard technique (e.g. ProVerif [Bla01], Scyther [Cre08a] etc.). Then, the composition of the protocols can be proven to be abstractly correct by applying one of the standard protocol composition results, as per above.

5.3 Crossing-Paths

As discussed in the introduction, there is nothing to prevent the penetrator from taking terms from a transport-layer message and then moving them into the application layer. We term such penetrator paths *crossing-paths* and the corresponding terms *crossing terms*. Clearly, we cannot safely abstract such bundles as the transport-layer sections of such penetrator paths will not appear in the high-level bundle and therefore there is nowhere to obtain the crossing term from.

Note that we do not mind if the penetrator moves values from the application layer to the transport layer (which could also be considered as a crossing path). This is because such crossovers do not affect our ability to abstract the bundle as they affect transport-layer behaviour, rather than application-layer behaviour. Further, as we assume that the bundle is abstractly correct, we are already assuming that the application-layer protocol does nothing to break the transport-layer protocol.

In this section we prove (Lemma 5.24) that we are able to remove crossing-paths from a bundle $\mathcal{B}$ and produce a bundle $\mathcal{B}'$, of a related strand space, such that $\mathcal{B} \trianglerighteq \mathcal{B}'$. This proof will be used in later sections in order to help make bundles interference-free. We begin by formally defining what a crossing-path is.

Definition 5.10. Let $\mathcal{B}$ be a normal low-level bundle and p be a penetrator message-construction subpath that starts at a positive regular node n_1 and finishes at a negative regular node $n_2 \in \mathcal{N}_{payload}$. Let p_d be the longest destructive prefix of p that does not traverse a key-edge. p is a *crossing-path* iff either:

1. $n_1 \in \mathcal{N}_{trpt}$ and, letting $c = chan(n_1)$, $\mathsf{expath}(p_d) \not\leq \mathsf{expath}^{\mathsf{c}}(c)$, and $\mathsf{expath}^{\mathsf{c}}(c) \not\leq \mathsf{expath}(p_d)$; or
2. $n_1 \in \mathcal{N}_{non\text{-}payload}$.

A bundle $\mathcal{B}$ is *crossing-path-free* iff it contains no crossing paths. The *crossing point* is defined as the last node on p_d and the *crossing term* is defined as $msg(n)$ where n is the crossing point.

The first clause in the above captures the case where the penetrator takes a term that originated within a transport-layer message, but not from within the application-layer content or enclosing the application-layer content, and then uses

it in constructing the application-layer content of another message. The second clause captures the case where the penetrator takes a term obtained from a handshake node and uses it to construct the application-layer content of another message. Both of these take terms from a non-application-layer source and move it into the application layer, and are therefore crossing-paths.

In order to abstract our bundles we need to remove any crossing-paths, as such paths will not be abstractly constructible. Our basic approach is to replace the crossing edge with a M strand that simply originates the required term; the resulting bundle would clearly have no crossing-paths and therefore could be safely abstracted. A difficulty with this approach is that the unique origination assumptions that held on the original low-level bundle would no longer hold in the transformed bundle. This means that if a low-level bundle does not satisfy a given property, then the high-level bundle may satisfy it (as the high-level bundle may not satisfy the unique origination assumptions and thus the property holds vacuously). Therefore, we remove crossing-paths, taking care not to break any unique origination assumptions.

In this section we firstly, in Section 5.3.1, prove several small lemmas that follow almost immediately from the assumptions of the previous section. Then, in Section 5.3.2, we define a new, related, strand space in which the penetrator's initial knowledge has been updated to include various crossing terms that are not relevant to the application-layer. This is done in order to remove several types of crossing-paths. By enlarging the set of terms that the penetrator initially knows we could be allowing him to develop more attacks that would not have been possible initially. Therefore, in Section 5.3.3 we define a class of *abstract correctness preserving* bundle transformations that ensure false attacks against the transport-layer protocol are not introduced. We also show that the types of transformations that we perform are abstract correctness preserving. Lastly, in Section 5.3.4 we consider how to actually remove crossing-paths.

5.3.1 Preliminaries

We now prove that if a regular node has a transport-layer encryption as a subterm of its message, then it must be a transport-layer node. Further, we show that the encryption must be a transport-layer encryption of the channel of the transport-layer node.

Lemma 5.11. If n is a regular node such that for some $e \in \mathcal{E}_{trpt}$, $e \sqsubseteq msg(n)$ then $n \in \mathcal{N}_{trpt}$ and $e \in \mathcal{E}_{trpt}^{chan(n)}$.

Proof. Let n and e be as stated. If $n \in \mathcal{N}_{non\text{-}payload}$ then it immediately follows that $e \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$ which violates Assumption 5.9 (3). Similarly, if $n \in \mathcal{N}_{\perp}$ then, as $msg(n) \in \mathcal{T}_{app}$, $e \in \mathcal{E}_{app}$ which violates Assumption 5.9 (2). Hence, as the regular nodes are partitioned, $n \in \mathcal{N}_{trpt}$.

Let es be an extraction path such that $e \sqsubseteq_{es} msg(n)$. If $\mathsf{expath^c}\,(chan(n)) < es$ then it follows that $e \in \mathcal{E}_{app}$, violating Assumption 5.9 (2). Thus $\mathsf{expath^c}\,(chan(n)) \not< es$. Alternatively, if $es \not< \mathsf{expath^c}\,(chan(n))$ then it follows that $e \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$ (by Definition 5.1 and Definition 5.2), violating Assumption 5.9 (3). Therefore, $es < \mathsf{expath^c}\,(chan(n))$ and hence, by definition of $\mathcal{E}_{trpt}$, $e \in \mathcal{E}_{trpt}^{chan(n)}$. □

Next we show that whenever there is a path between two regular transport-layer nodes such that the same transport-layer encryption is a subterm of all the messages on the path, then neither the channel nor application-layer message can be altered. This follows from the fact that distinct channels use distinct transport-layer encryptions (i.e. Assumption 5.9 (1)).

Lemma 5.12. Let $n_1, n_2 \in \mathcal{N}_{trpt}$. If there exists $e \in \mathcal{E}_{trpt}$ and a penetrator path p from n_1 to n_2 such that for every node n on p, $e \sqsubseteq n$, then $appmsg(n_1) = appmsg(n_2)$ and $chan(n_1) = chan(n_2)$.

Proof. Let n_1, n_2, e and p be as stated in the lemma. As $e \in \mathcal{E}_{trpt}$ it follows, by Lemma 5.11, that $e \in \mathcal{E}_{trpt}^{chan(n_1)}$ and $e \in \mathcal{E}_{trpt}^{chan(n_2)}$. Therefore, by Assumption 5.9 (1) $chan(n_1) = chan(n_2)$.

By the definition of $\mathcal{E}_{trpt}$ it follows that there exists extraction paths es_1 and es_2 such that $es_1 < \mathsf{expath^c}(c)$, $es_2 < \mathsf{expath^c}(c)$ and:

$$e \sqsubseteq_{es_1} msg(n_1) \quad \text{and} \quad e \sqsubseteq_{es_2} msg(n_2).$$

If $es_1 = es_2$ then it immediately follows that $appmsg(n_1) = appmsg(n_2)$, as required. If $es_1 < es_2$ (the other case is symmetric to this) then define $xs_1 \neq xs_2$ such that:

$$es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} xs_1 = \mathsf{expath^c}(c) = es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} xs_2. \tag{5.1}$$

Suppose, for a contradiction, that $\mathsf{Decrypt}$ in xs_1, and let ys be the longest extraction path such that $ys \hat{} \langle \mathsf{Decrypt} \rangle \leq xs_1$. Thus, as $es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys < \mathsf{expath^c}(c)$, by definition of $\mathcal{E}_{trpt}$:

$$extract(msg(n), es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys) = extract(e, \langle \mathsf{Decrypt} \rangle \hat{} ys) \in \mathcal{E}_{trpt}. \tag{5.2}$$

Let i be the (1-based) index of the last $\mathsf{Decrypt}$ on $\mathsf{expath^c}(c)$ and note that, by definition of ys, $i = |es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys \hat{} \langle \mathsf{Decrypt} \rangle|$. However, as $es_1 < es_2$, it follows that there exists an index $i' > i$ at which the final $\mathsf{Decrypt}$ appears in $es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys \hat{} \langle \mathsf{Decrypt} \rangle$. Thus, $es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys \hat{} \langle \mathsf{Decrypt} \rangle \not\leq \mathsf{expath^c}(c)$. Further:

- If $\mathsf{expath^c}(c) < es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys \hat{} \langle \mathsf{Decrypt} \rangle$ then $extract(e, \langle \mathsf{Decrypt} \rangle \hat{} ys) \in \mathcal{E}_{app}$. Hence, by Equation 5.2, $extract(msg(n_2), es_2 \hat{} \langle Decrypt \rangle \hat{} ys) = extract(e, \langle \mathsf{Decrypt} \rangle \hat{} ys) \in \mathcal{E}_{trpt} \cap \mathcal{E}_{app}$, contradicting Assumption 5.9 (2).
- If $\mathsf{expath^c}(c) \not\leq es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} ys \hat{} \langle \mathsf{Decrypt} \rangle$ then:

  $$extract(msg(n_2), es_2 \hat{} \langle Decrypt \rangle \hat{} ys) = extract(e, \langle \mathsf{Decrypt} \rangle \hat{} ys) \in \mathcal{E}_{trpt\text{-}non\text{-}msg}.$$

  Hence, by Equation 5.2, $extract(e, \langle \mathsf{Decrypt} \rangle \hat{} ys) \in \mathcal{E}_{trpt} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, contradicting Assumption 5.9 (3).

Hence, as we derive a contradiction in both of the above cases, $\mathsf{Decrypt}$ does not appear in xs_1. Further, as by Equation 5.1 xs_2 is a suffix of xs_1, it follows that $\mathsf{Decrypt}$ does not appear in xs_2. Thus, both xs_1 and xs_2 must be equal to the longest $\mathsf{Decrypt}$-free suffix of $\mathsf{expath^c}(c)$ and hence $xs_1 = xs_2$. Moreover, $es_1 = es_2$ and therefore $appmsg(n_1) = appmsg(n_2)$, as required. □

5.3.2 Enlarging the Strand Space

When removing crossing-paths from a bundle we may need to add extra values to the penetrator's initial knowledge. For instance, consider a bundle containing a crossing-path where the penetrator takes a handshake nonce and then uses it in an application-layer message. This bundle will be transformed to an equivalent bundle in which the crossing-path has been replaced by a M strand.

The reason why we allow the above transformation is that the penetrator was taking a value that originated only at the transport-layer, and was moving it into the application layer. From the application-layer's point of view, this behaviour is indistinguishable from the penetrator simply originating the value on a M strand.

We now consider how to enlarge the set of public terms in such a way as to allow transformations like those discussed above. We do this by defining a new strand space, based on the existing strand space, that differs only in the penetrator's initial knowledge (cf. Definition 2.10). Further, we restrict the penetrator's behaviour so that the transport-layer behaviour is identical (but the application-layer behaviour is enlarged, as discussed above). Clearly, such a transformation has the potential to introduce false attacks against the application-layer protocol by giving the penetrator encryption keys that he could not otherwise obtain. However, we avoid doing this by only adding values that are not in $\mathcal{A}_{app}$ (Definition 5.1) which contains all terms of interest to the application-layer.

This still has some repercussions on the application-layer proof. In general, in order to prove the required application-layer result we will need an assumption on the terms in $\mathcal{A}_{\mathcal{P}}$ (e.g. Assumption 3.21). Since we add terms to $\mathcal{A}_{\mathcal{P}}$, the application-layer correctness proof is only allowed to assume that certain terms are *not* in $\mathcal{A}_{\mathcal{P}}$, rather than specifying exactly what is allowed (i.e. it can only assume that $\mathcal{A}_{\mathcal{P}} \cap X = \emptyset$, not that $\mathcal{A}_{\mathcal{P}} \subseteq X$). Further, the assumption may only consider terms in $\mathcal{A}_{app}$ (i.e. $X \subseteq \mathcal{A}_{app}$), as terms that are not in $\mathcal{A}_{app}$ may be added to the penetrator's initial knowledge by the transformation. In practice this is not a restriction, since terms that are not in $\mathcal{A}_{app}$ cannot possibly affect the security of the application layer. Note that the WebAuth assumptions in Assumption 3.21 are formulated in such a way.

In order to define this transformation we firstly define a set that contains all terms that may appear at crossing points and may need transforming as above.

Definition 5.13. The set of *transport extractable components*, denoted $\mathcal{T}_X$ is defined as the set of all terms that are not in $\mathcal{A}_{app}$, but are in the following set:

$$\begin{aligned}&\{\mathit{extract}(t, es) \mid t \in \mathcal{T}_{\mathit{payload}}, es \in \mathcal{EP}, \mathsf{expath}^{\mathsf{c}}(\mathit{chan}(t)) \not\leq es,\\ &\qquad\qquad es \not\leq \mathsf{expath}^{\mathsf{c}}(\mathit{chan}(t))\}\\ &\cup \{s \mid t \in \mathcal{T}_{\mathit{non\text{-}payload}}, s \sqsubseteq t\}\end{aligned}$$

Recall that $\mathcal{T}_{payload}$ gives the set of all terms that carry an application-layer payload. Thus, the first clause in the above definition extracts all the transport-layer packaging of a payload carrying-message, whilst the second clause gives all the terms that the penetrator could possibly extract from other transport-layer messages. The following lemma proves that this set is sufficiently large for our purposes by proving that any crossing term that is not in $\mathcal{A}_{app}$ must be in $\mathcal{T}_X$.

Lemma 5.14. Let $\mathcal{B}$ be a normal low-level bundle. If p is a crossing-path starting at a node n_1, t the crossing term on p and $t \notin \mathcal{A}_{app}$, then $t \in \mathcal{T}_X$.

Proof. Let $\mathcal{B}$, p, t be as per the lemma. Further, let n_1 be the first node of p, n_c be the crossing point (i.e. $t = msg(n_c)$ is the crossing term), and p_d be the destructive path starting at n_1 and ending at n_c. By the definition of a crossing-path it follows that either $n_1 \in \mathcal{N}_{trpt}$ or $n_1 \in \mathcal{N}_{non\text{-}payload}$. In the latter case it immediately follows that $t \sqsubseteq msg(n_1) \in \mathcal{T}_{non\text{-}payload}$ (as p_d is destructive) and hence that $t \in \mathcal{T}_X$. Otherwise, $n_1 \in \mathcal{N}_{trpt}$ and thus, by Definition 5.10, $\mathsf{expath}\,(p_d) \not\leq \mathsf{expath}^{\mathsf{c}}\,(chan(n_1))$ and $\mathsf{expath}^{\mathsf{c}}\,(chan(n_1)) \not\leq \mathsf{expath}\,(p_d)$. Therefore, as $msg(n_1) \in \mathcal{T}_{payload}$, $t \in \mathcal{T}_X$. □

Using the above it is now possible to define the enlarged strand space. Clearly, if we were to allow the penetrator to use the terms from $\mathcal{T}_X$ arbitrarily in the enlarged strand space, then there would exist bundles in which the penetrator could break otherwise secure transport protocols (for example, if $\mathcal{T}_X$ contained transport-layer keys that should be secret).

Definition 5.15. Let Σ be an abstractly correct low-level strand space. The *enlarged strand space of* Σ, denoted Σ_E is defined to be as Σ but with the set of public terms enlarged to be:

$$\mathcal{A}_{\mathcal{P}}^{\Sigma_E} = \mathcal{A}_{\mathcal{P}}^{\Sigma} \cup \mathcal{T}_X.$$

In order to prove anything useful in the above strand space we will need Assumption 5.9 to hold. Most of the assumptions hold trivially in Σ_E as the regular behaviour of agents has not changed. However, note that not all bundles in Σ_E will necessarily be abstractly correct as the penetrator may now know values that could be used to compromise transport-layer protocols. However, in the following we can restrict our attention to the subset of bundles that are abstractly correct.

Lemma 5.16. If Σ is a low-level strand space satisfying layer-disjoint encryption then Σ_E also satisfies layer-disjoint encryption, with the exception of Definition 5.8 (9).

Proof. Let Σ and Σ_E be as per the lemma. From the definition of layer-disjoint encryption it is clear that Definition 5.8 (1), (2), (3), (4) and (8) trivially hold in Σ_E. Further, it can be seen from the definition of $\mathcal{T}_X$ that no $\{|m|\}_k \in \mathcal{E}_{trpt}$ can be in $\mathcal{T}_X$ and thus Definition 5.8 (5) holds. Also, Definition 5.8 (6) and (7) are monotonic with respect to $\mathcal{T}_{app}$ and thus trivially hold in Σ_E. Hence, Σ_E satisfies layer-disjoint encryption with the exception of Definition 5.8 (9). □

Given the above result, for the remainder of this section we make exclusive use of the enlarged strand space, Σ_E, rather than Σ.

5.3.3 Abstract Correctness Preserving Transformations

Later in this section we will develop a number of bundle transformations that take a bundle from Σ_E and return another bundle. We then prove that the resulting bundles satisfy certain properties. However, in these proofs we will need to assume that the bundle is abstractly correct. Therefore, when transforming bundles we need to be careful not to go outside of the set of abstractly correct bundles. In order to ensure this, we define a class of transformations, known as *abstract correctness preserving* transformations, that preserve the abstract correctness of bundles.

Definition 5.17. A bundle transformation f is *abstract correctness preserving* (henceforth ACP) iff whenever $\mathcal{B}$ is an abstractly correct bundle, $f(\mathcal{B})$ is abstractly correct.

Observe that the composition of a series of ACP bundle transformations is itself ACP.

In order to prove that the transformations we use in this section are ACP, we firstly need an additional assumption that the channel restrictions are *sensible*. For example, recall that a RV strand followed by a SD strand is equivalent to a HJ strand when the channel types and application-layer messages match. We require that if a channel prohibits certain HJ strands, then it also prohibits the equivalent RV, SD strand combination. This is merely a well-formedness condition on the channel and is satisfied by all existing definitions (e.g. [DL08, KL09]).

Assumption 5.18. A low-level strand space Σ contains only *sensible* channels iff: the strand space contains a Receive, Send subpath pair $\langle -(A_\psi,\ P_\phi,\ i,\ m,\ c),\ +(?,\ ?,\ \_,\ m,\ \bot)\rangle$ and $\langle -(?,\ ?,\ \_,\ m,\ \bot), +(P'_{\psi'},\ B_{\phi'},\ i,\ m,\ c)\rangle$ iff it contains the equivalent Hijack, Renumber or Hijack-Renumber subpath $\langle -(A_\psi,\ P_\phi,\ i,\ m,\ c),\ +(P'_{\psi'},\ B_{\phi'},\ i',\ m,\ c)\rangle$.

Using the above we can now prove a simple lemma that shows a sufficiently large class of transformations are ACP.

Lemma 5.19. If a transformation is one of the following then it is ACP:

1. The addition of an M strand connected only to a message-construction path;
2. The concatenation of adjacent Receive and Send subpaths to form an equivalent Hijack, Renumber or Hijack-Renumber subpath;
3. The addition of redundant S and C strand pairs.

Proof. In order to show that the transformation is ACP it suffices to show that it preserves abstract correctness. (1) follows immediately from the observation that introducing an M strand connected only to a message-construction path means no new penetrator subpaths of the form of Definition 4.19 are introduced. Hence, no new transport-layer subpaths are introduced and thus trivially abstract correctness is preserved.

(2) is ACP since whenever adjacent Receive and Send subpath pairs are in the strand space, by Assumption 5.18, the corresponding Hijack, Renumber or Hijack-Renumber subpath is allowed.

(3) is ACP since the transformation will, at worst, create a Receive, Send subpath pair where a Hijack, Renumber or Hijack-Renumber subpath existed previously. Thus, by Assumption 5.18, the resulting subpaths are in the strand space iff the original subpaths were. □

In this section we only consider transformations of the above form, and thus all bundle transformations are ACP. Further, note that the composition of an arbitrary chain of bundle transformations is ACP.

5.3.4 Removing Crossing-Paths

We now prove the main result from this section and show how to remove individual crossing-paths from a bundle (in the enlarged strand space Σ_E). In order to do this we consider several sub-cases, each of which removes a certain type of crossing-path whilst not adding any new crossing-paths.

The first case that we consider concerns crossing-paths where the crossing term is non-public and does not originate in the application-layer. Such cases correspond to the penetrator passing a transport-layer term, like a nonce, into the application-layer. Intuitively, such behaviours are harmless in that the application-layer behaviour is not affected by the fact that the value originated in the transport-layer behaviour. We can remove such a crossing-path by creating a new M strand, since any such crossing term is in $\mathcal{A}_{\mathcal{P}}^{\Sigma_E}$.

Lemma 5.20. Let $\mathcal{B}$ be a normal low-level bundle and p be a crossing-path such that the crossing term $t \notin \mathcal{A}_{\mathcal{P}}$ and $t \notin \mathcal{A}_{app}$. Then there exists a $\mathcal{B}'$ in Σ_E such that $\mathcal{B} \trianglerighteq \mathcal{B}'$, in which p has been removed, and such that the transformation is ACP.

Proof. Let $\mathcal{B}$, p and t be as per the lemma. We construct the bundle $\mathcal{B}'$ as follows. Let p_d be the destructive penetrator path leading to the crossing point on p.

As $t \notin \mathcal{A}_{app}$, Lemma 5.14 can be applied to deduce that $t \in \mathcal{T}_X$. Hence, by definition of Σ_E, $t \in \mathcal{A}_{\mathcal{P}}^{\Sigma_E}$ and thus a new M strand $\langle +t \rangle$ can be introduced in place of p_d. Note that the resulting bundle is well formed (i.e. it satisfies the bundle conditions), still normal and $\mathcal{B} \trianglerighteq \mathcal{B}'$ as we have not added any penetrator paths. Further, by Lemma 5.19, such a transformation is ACP. □

Before considering the next case we prove a technical lemma. Informally, the lemma states that there is only one possible way of extracting the application-layer message from a given encryption. That is, if there are two extraction paths that are prefixes of the channel extraction path and that lead to the same encrypted subterm in two different transport-layer messages, then the extraction paths must be identical.

Lemma 5.21. Let $c \in \mathit{Channels} \setminus \{\bot\}$, $t_1, t_2 \in \mathcal{T}^c_{\mathit{payload}}$ and $es_1, es_2 \leq \mathsf{expath}^{\mathsf{c}}(c)$. If $\mathit{extract}(t_1, es_1) = \mathit{extract}(t_2, es_2)$ is an encryption, then $es_1 = es_2$.

Proof. Let c, t_1, t_2, es_1 and es_2 be as per the lemma. Further, let $t = \mathit{extract}(t_1, es_1) = \mathit{extract}(t_2, es_2)$. Suppose for a contradiction that $es_1 < es_2$ (the case for $es_2 < es_1$ is entirely symmetric). Since t is an encryption it follows that there exists es_1' such that $es_2 = es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1'$. Let es' be such that $\mathsf{expath}^{\mathsf{c}}(c) = es_2 \hat{} es'$. If $es_2 = \mathsf{expath}^{\mathsf{c}}(c)$ then $\mathit{extract}(t_2, es_2) = t \in \mathcal{E}_{app}$ but $\mathit{extract}(t_1, es_1) = t \in \mathcal{E}_{trpt}$. Hence, $t \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt}$, contradicting Assumption 5.9 (2). Otherwise, $es_2 < \mathsf{expath}^{\mathsf{c}}(c)$ and thus, as t is an encryption, es' starts with $\mathsf{Decrypt}$. Consider the last $\mathsf{Decrypt}$ in es', i.e. let es'' and es''' be extraction paths such that $es' = es'' \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es'''$ and $\mathsf{Decrypt} \not\sqsubset es'''$. In summary:

$$\mathsf{expath}^{\mathsf{c}}(c) = \underbrace{es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1'}_{es_2} \hat{} \underbrace{es'' \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es'''}_{es'}$$

where es''' is $\mathsf{Decrypt}$-free.

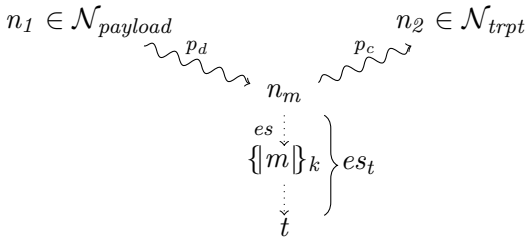


Figure 5.3: An illustration of Lemma 5.22. In the figure above dotted arrows indicate extraction paths, whilst wiggly lines arrows indicate, as usual, penetrator paths.

Let $t' = extract(t_1, es_2 \hat{} es'')$ (note that this is a valid extraction path for t_1 since it is a prefix of $\mathsf{expath}^{\mathsf{c}}(c)$). As $es_2 \hat{} es'' \leq \mathsf{expath}^{\mathsf{c}}(c)$ it follows, by definition of $\mathcal{E}_{trpt}$, that $t' \in \mathcal{E}_{trpt}$. We now prove, for a contradiction, that $t' \notin \mathcal{E}_{trpt}$. Thus, observe that:

$$\begin{aligned} t' &= extract(t_1, es_2 \hat{} es'') \\ &= extract(t_1, es_1 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1' \hat{} es'') \\ &= extract(t, \langle \mathsf{Decrypt} \rangle \hat{} es_1' \hat{} es'') \\ &= extract(t_2, es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1' \hat{} es''). \end{aligned}$$

Further, as $\mathsf{expath}^{\mathsf{c}}(c) = es_2 \hat{} es'$ it follows that $es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1' \hat{} es'' \not\leq \mathsf{expath}^{\mathsf{c}}(c)$. Thus, considering the copy of t' at $es_2 \hat{} \langle \mathsf{Decrypt} \rangle \hat{} es_1' \hat{} es''$ in t_2, it follows that t' cannot be in $\mathcal{E}_{trpt}$. However, as $t' \in \mathcal{E}_{trpt}$, this contradicts Assumption 5.9 (2). Thus, as we derive a contradiction in all cases, $es_1 = es_2$. □

We now consider the most complex case of the proof. This case considers a crossing-path where the crossing term originally originated in the application layer. Thus, the penetrator has taken the value from within the application layer, then moved it to the transport layer, then moved it back to the application layer (the last of these paths is the crossing-path). In this particular case, we show that whenever the penetrator takes a value from within the application layer (from a regular node) and sends it into a non-application-layer context then there must exist a node such that any enclosing encryptions are in $\mathcal{E}_{trpt\text{-}non\text{-}msg} \cap \mathcal{E}_{app}$. The utility of this lemma follows from the fact that, due to Assumption 5.9 (4), the inverse key to any such encryption must be public (e.g. cryptographic certificates). Therefore, we can arrange for the penetrator to remove these encryptions to obtain the crossing term, and so eliminate the crossing-path.

Intuitively, this result follows from the fact that any enclosing encryption would have to be a transport-layer message encryption or an application-layer encryption at the first node, but could only be a transport-layer message encryption or a transport-layer non-message encryption at the second node. Due to the disjointness conditions imposed by our assumptions above, the only possible case is that the encryption is a transport-layer encryption from both nodes' point of view. Therefore, the proof essentially shows that it is impossible to view a single transport-layer message as containing two different application-layer messages.

Lemma 5.22. Let $\mathcal{B}$ be a normal low-level bundle, $p = p_d \hat{} p_c$ be a penetrator path that does not traverse a key-edge or a KG starting at $n_1 \in \mathcal{N}_{payload}$ and ending at $n_2 \in \mathcal{N}_{trpt}$, such that p_d is destructive, p_c is constructive and n_m is the middle node (i.e. $n_m = p_c(1)$). Further, let es_t be an extraction path to a term t inside $msg(n_m)$ such that $t \sqsubseteq_{es_t} msg(n_m)$, t is inside the application layer at n_1 and t lies outside of the application layer at n_2. That is, either $n_1 \in \mathcal{N}_\perp$ or $\mathsf{expath}^{\mathsf{c}}(chan(n_1)) \leq \mathsf{expath}(p_d) \hat{} es_t$, but $\mathsf{expath}^{\mathsf{c}}(chan(n_2)) \not\leq \mathsf{expath}^{\sim}(p_c) \hat{} es_t$. Then, any encryption that encloses t in $msg(n_m)$ is in $\mathcal{E}_{trpt\text{-}non\text{-}msg} \cap \mathcal{E}_{app}$.

Proof. Let $\mathcal{B}$, p, p_d, p_c, n_1, n_2, n_m, t and es_t be as stated above. Further let m, k and es be an extraction path such that $es < es_t$ and $extract(msg(n_m), es) = \{\!|m|\!\}_k$, as illustrated in Figure 5.3. We need to prove that $\{\!|m|\!\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg} \cap \mathcal{E}_{app}$.

Consider n_2; by assumption $\mathsf{expath}^{\mathsf{c}}(chan(n_2)) \not\leq \mathsf{expath}^{\sim}(p_c) \hat{} es_t$ and hence $\mathsf{expath}^{\mathsf{c}}(chan(n_2)) \not\leq \mathsf{expath}^{\sim}(p_c) \hat{} es$. Therefore, in order to prove that $\{\!|m|\!\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$ it suffices to show that $\mathsf{expath}^{\sim}(p_c) \hat{} es \not\leq \mathsf{expath}^{\mathsf{c}}(chan(n_2))$ (cf. Definition 5.2). Assume, for a contradiction, that $\mathsf{expath}^{\sim}(p_c) \hat{} es \leq \mathsf{expath}^{\mathsf{c}}(chan(n_2))$. It thus follows that $\{\!|m|\!\}_k \in \mathcal{E}_{trpt}$ and therefore, by Lemma 5.11, $n_1 \in \mathcal{N}_{trpt}$. Hence, Lemma 5.12 can be applied to deduce that $appmsg(n_1) = appmsg(n_2)$ and $chan(n_1) = chan(n_2)$. Let $c = chan(n_1)$. As $\mathsf{expath}(p_d) \hat{} es \leq \mathsf{expath}^{\mathsf{c}}(c)$ and:

$$\begin{aligned} &extract(msg(n_1), \mathsf{expath}(p_d) \hat{} es) \\ &= extract(msg(n_2), \mathsf{expath}^{\sim}(p_c) \hat{} es) \\ &= \{\!|m|\!\}_k \end{aligned}$$

it follows, by Lemma 5.21 applied to $msg(n_1)$, $msg(n_2)$, c, $\mathsf{expath}(p_d) \hat{} es$ and $\mathsf{expath}^{\sim}(p_c) \hat{} es$, that $\mathsf{expath}(p_d) \hat{} es = \mathsf{expath}^{\sim}(p_c) \hat{} es$. Hence, $\mathsf{expath}(p_d) = \mathsf{expath}^{\sim}(p_c)$ and therefore, since $\mathsf{expath}^{\mathsf{c}}(c) \leq \mathsf{expath}(p_d) \hat{} es_t$ (by assumption of the lemma), it follows that $\mathsf{expath}^{\mathsf{c}}(c) \leq \mathsf{expath}^{\sim}(p_c) \hat{} es_t$, contradicting an assumption of the lemma.

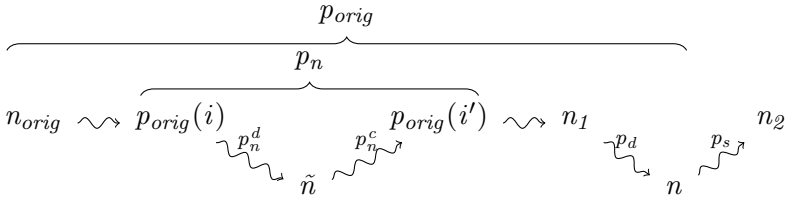
Thus, as we derive a contradiction in all cases, it must be the case that $\mathsf{expath}^{\sim}(p_c) \hat{} es \not\leq \mathsf{expath}^{\mathsf{c}}(chan(n_2))$ and thus $\{\!|m|\!\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$.

Further, observe that any encryption enclosing t in $msg(n_1)$ must either be in $\mathcal{E}_{trpt}$ or $\mathcal{E}_{app}$. However, as $\mathcal{E}_{trpt} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg} = \emptyset$, by Assumption 5.9 (3), it follows that $\{\!|m|\!\}_k \in \mathcal{E}_{app}$. Thus, $\{\!|m|\!\}_k \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as required. Further, this holds for any encryption that encloses t in $msg(n_1)$. □

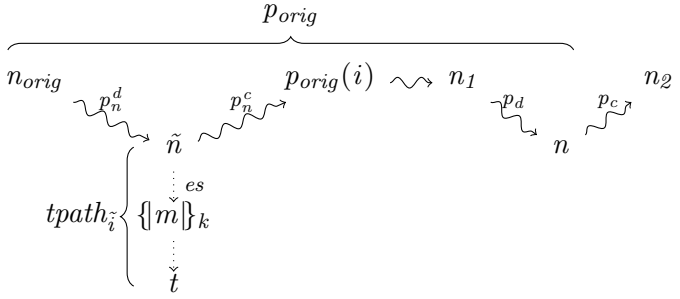
Using the above lemmas it is now possible to prove that it is always possible to remove a crossing-path to yield an equivalent bundle. The following proof considers only non-concatenations (i.e. only atoms and encryptions) as crossing terms. We lift this assumption in Lemma 5.24.

Lemma 5.23. Let $\mathcal{B}$ be a normal low-level bundle that contains a crossing-path, but contains no crossing-paths such that the crossing term is a concatenation. Then, there exists a normal bundle $\mathcal{B}'$ in Σ_E in which the number of crossing-paths in the bundle strictly decreases and such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and the transformation is ACP.

Proof. Let $\mathcal{B}$ be as per the lemma; we construct the bundle $\mathcal{B}'$ as follows. Let p be a crossing-path in $\mathcal{B}$ such that there does not exist a crossing-path p' where $p'(1) \preceq p(1)$ (i.e. p is a first crossing-path, according to $\preceq$). Let p_d be the longest



(a) The penetrator paths for Case (1).



(b) The penetrator paths for Case (2).

Figure 5.4: The penetrator paths involved in Lemma 5.23.

destructive prefix of p that does not traverse a key-edge. Let n_1 be the first node of p_d, n_2 be the last node of p, n be the crossing point and $t = msg(n)$ be the crossing term, as illustrated in the right-hand part of Figure 5.4). Clearly, if $t \in \mathcal{A}_{\mathcal{P}}$ then the crossing-path can simply be removed by the addition of a new M strand $\langle +t \rangle$ replacing p_d. Otherwise, $t \notin \mathcal{A}_{\mathcal{P}}$. If $t \notin \mathcal{A}_{app}$ then Lemma 5.20 can be applied to construct $\mathcal{B}'$.

Otherwise, $t \in \mathcal{A}_{app}$. Let $n_{orig} \preceq n_1$, be the node that originates the instance of t that ultimately ends up at this crossing-point. Further, let p_{orig} be a path (nb. not necessarily a penetrator path), starting at n_{orig} and ending at n, such that $t \sqsubseteq msg(n')$ for every node n' on p_{orig}. Note that by definition, p_{orig} cannot traverse any key-edges or KG strands (although n_{orig} could be the positive node on a KG strand). Further, for each i, let $tpath_i$ be the extraction path needed to extract the copy of t that we are interested in from $p_{orig}(i)$ (e.g. $tpath_{|p|} = \langle\rangle$). We now prove that there exists a node $\tilde{n}$, at index $\tilde{i}$ on p_{orig}, that contains t and such that any encryption that encloses t in $msg(\tilde{n})$ is in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$. Formally:

$$\begin{aligned} t \sqsubseteq_{tpath_{\tilde{i}}} msg(\tilde{n}) \wedge \forall es, m, k \cdot es < tpath_{\tilde{i}} \\ \wedge \{\!|m|\!\}_k \sqsubseteq_{es} msg(\tilde{n}) \implies \{\!|m|\!\}_k \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}. \end{aligned} \tag{5.3}$$

There are two cases to consider.

Case 1 Suppose an index i exists such that $p_{orig}(i)$ is regular and t occurs inside the application-layer payload, i.e. either $p_{orig}(i) \in \mathcal{N}_\perp$ or $p_{orig}(i) \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \leq tpath_i$. Let i be the largest such index and $i' > i$ be the index of the first regular node on p_{orig} after $p_{orig}(i)$. Note that such an i' must exist as n_1 is such a node and $p_{orig}(i) \neq n_1$ as, otherwise, n_1 would not be the start of a crossing-path. Then, by definition of i, it follows that

t does not occur inside the application-layer payload at i', so $p_{orig}(i') \notin \mathcal{N}_{\perp}$, and either:

1. $p_{orig}(i') \in \mathcal{N}_{non\text{-}payload}$; or
2. $p_{orig}(i') \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i'))) \not\leq tpath_{i'}$.

Consider the penetrator path p_n between $p_{orig}(i)$ and $p_{orig}(i')$: as $\mathcal{B}$ is normal we can divide p_n around the node that divides p_n into a destructive and a constructive subpath, as illustrated in Figure 5.4a. Let $\tilde{n}$ be this node; it remains show that any encryption enclosing t in $\tilde{n}$ is in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as per Equation 5.3.

If (1) applies then any encryption surrounding t at $p_{orig}(i')$ is in $\mathcal{E}_{trpt\text{-}non\text{-}msg}$, by definition of $\mathcal{E}_{trpt\text{-}non\text{-}msg}$. Further, since t lies within the application-layer at $p_{orig}(i)$, any encryption enclosing t must either be in $\mathcal{E}_{app}$ or $\mathcal{E}_{trpt}$. However, the latter case would contradict Assumption 5.9 (3) and thus, every encryption enclosing t at $p_{orig}(i)$ must be in $\mathcal{E}_{app}$. Hence, every encryption enclosing t at $msg(\tilde{n})$ (which are a subset of the intersection of the encryptions enclosing t at $p_{orig}(i)$ and $p_{orig}(i')$) must be in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as required.

Otherwise, if (2) applies then we can apply Lemma 5.22 to deduce that any encryption enclosing t in $msg(\tilde{n})$ is in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as required.

Case 2 If the above case does not apply then for every regular node n on p_{orig}, t does not occur inside the application-layer payload. Therefore, for each index i of a regular node on p_{orig}, $n \notin \mathcal{N}_{\perp}$ and either $n \in \mathcal{N}_{non\text{-}payload}$, or $n \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(n)) \not\leq tpath_i$. Suppose, for a contradiction, that n_{orig} is regular. Note that, as $t \in \mathcal{A}_{app}$ it follows, by Assumption 5.9 (7), that $n_{orig} \in \mathcal{N}_{trpt}$ and for every extraction path es such that $t \sqsubseteq_{es} msg(n_{orig})$, $\mathsf{expath}^{\mathsf{c}}(chan(n_{orig})) \leq es$. Hence, $\mathsf{expath}^{\mathsf{c}}(chan(n_{orig})) \leq tpath_1$, contradicting the above. Therefore, it follows that n_{orig} is a penetrator node and is thus the positive node on either a M, E or KG strand (t is not a concatenation by assumption). We perform a case analysis on the type of n_{orig}.

Firstly, if n_{orig} lies on either a KG or E strand then it necessarily follows that $t = msg(n_{orig})$ since t originates on n_{orig}. Thus, the crossing path can be trivially removed by replacing p_d with a connection from n_{orig}.

Otherwise, n_{orig} lies on a M strand. Since $t \in \mathcal{A}_{app} \backslash \mathcal{A}_{\mathcal{P}}$, by Assumption 5.9 (6), t is surrounded only by encryptions from $\mathcal{E}_{app}$ in n_{orig}. Let i be the index of the first regular node on p_{orig} (note that such a node exists as n_1 is one) and p be the normal penetrator path leading to $p_{orig}(i)$. Further, we divide p into a destructive path p_n^d and a constructive path p_n^c. This is illustrated in Figure 5.4b. Let $\tilde{n}$ be $p_n^c(1)$ and let $\tilde{i}$ be the index of $p_n^c(1)$ on p_{orig}. Further, let es, m and k be such that $es < tpath_{\tilde{i}}$ and $extract(es, msg(\tilde{n})) = \{|m|\}_k$. Note that $\{|m|\}_k \in \mathcal{E}_{app}$ as every encryption enclosing t in n_{orig} is in $\mathcal{E}_{app}$. It remains to show that $\{|m|\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as required by Equation 5.3.

If $p_{orig}(i) \in \mathcal{N}_{non\text{-}payload}$ it immediately follows, by definition of $\mathcal{E}_{trpt\text{-}non\text{-}msg}$ that $\{|m|\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$, as required. Otherwise, $p_{orig}(i) \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \not\leq tpath_i$. As $es < tpath_{\tilde{i}}$, it follows by definition of $tpath_i$ that $\mathsf{expath}^{\sim}(p_n^c)^\frown es < tpath_i$. Thus, it must be the case that

$\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \not\leq \mathsf{expath}^{\sim}(p_n^c)\hat{}es$. Further, if $\mathsf{expath}^{\sim}(p_n^c)\hat{}es \leq \mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i)))$ then $\{|m|\}_k \in \mathcal{E}_{trpt}$, which as $\{|m|\}_k \in \mathcal{E}_{app}$, contradicts Assumption 5.9 (2). Hence, $\mathsf{expath}^{\sim}(p_n^c)\hat{}es \not\leq \mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i)))$ and therefore $\{|m|\}_k \in \mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$.

Therefore, by Equation 5.3, there exists a node $\tilde{n}$ such that any encryption enclosing t in $msg(\tilde{n})$ is in $\mathcal{E}_{app} \cap \mathcal{E}_{trpt\text{-}non\text{-}msg}$. Therefore, by Assumption 5.9 (4), the inverse key for any encryption enclosing t in $\tilde{n}$ is in $\mathcal{A}_{\mathcal{P}}$. Hence, a new penetrator path can be constructed that starts at $\tilde{n}$ and consists of appropriate S and D strands (with keys coming from M strands) so as to extract t. The last node on this new path can then be connected to n, thus removing the crossing-path that did exist. Further, the new path is not a crossing-path, $\mathcal{B} \trianglerighteq \mathcal{B}'$ as there are no new causal predecessors of n, and by Lemma 5.19, the transformation is ACP. □

Lemma 5.24. Let $\mathcal{B}$ be a normal low-level bundle. Then there exists a bundle $\mathcal{B}'$ in Σ_E that contains no crossing-paths such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and the transformation is ACP.

Proof. Let $\mathcal{B}$ be a normal bundle. If $\mathcal{B}$ contains no crossing-paths such that the crossing term is a concatenation then we let $\mathcal{B}''$ be equal to $\mathcal{B}$. Otherwise, we construct a new bundle $\mathcal{B}''$ which is formed by adding extra S and C strand pairs to each such crossing-path to leave only crossing-paths such that the crossing term is not a concatenation. Note that $\mathcal{B}''$ will still be normal and that the transformation is ACP by Lemma 5.19 (1).

Since $\mathcal{B}''$ contains no crossing-paths with concatenations, Lemma 5.23 can be applied multiple times to remove each crossing-path to yield a bundle $\mathcal{B}'$ such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and which is reachable via an ACP transformation. □

5.4 Interference Freedom

In this section we prove our main result: that we can transform any bundle of Σ into an equivalent interference-free bundle of Σ_E. We begin in Section 5.4.1 by proving that the penetrator has only a few behaviours available. We then provide our first bundle transformation in Section 5.4.2 and show how to make nodes abstractly constructible. Using this we then show in Section 5.4.3 how to make nodes interference-free. Lastly, we show how to combine the above results to take an arbitrary bundle of Σ and transform it into an equivalent abstractable bundle.

5.4.1 Restricting the Penetrator Paths

We now consider what behaviours the penetrator has available to him, given the restrictions imposed on him by Assumption 5.9. In particular, we prove that the bundle is almost interference-free, but may contain transport-layer messages that are not fully received, but have had all of their encryptions removed (i.e. they are of the form $\ldots\hat{}appmsg\hat{}\ldots$). In particular, we show that each negative regular application-layer node either:

1. Receives messages on the $\perp$ channel; or

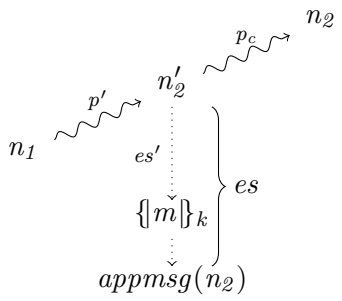


Figure 5.5: An illustration of Lemma 5.25.

2. Receives messages on a transport channel from a penetrator node that contains the application-layer message unencrypted (i.e. there is a node from which the application-layer message can be extracted without removing any transport-layer encryptions); or

3. Receives messages from another regular agent along a Hijack, Renumber, Hijack-Renumber or a Transmit strand.

We formalise these conditions as follows.

Lemma 5.25. Let $\mathcal{B}$ be a normal low-level bundle and $n_2 \in \mathcal{N}_{payload}$ a negative regular node in $\mathcal{B}$. Then there must exist a penetrator path p starting at a node n_1 and ending at n_2 such that either:

1. $n_2 \in \mathcal{N}_\perp$ and $n_1 \rightarrow n_2$; or

2. $n_2 \in \mathcal{N}_{trpt}$, p is constructive and there exists an extraction path es, not containing Decrypt, such that $es \hat{}\, \mathsf{expath}^\sim(p) = \mathsf{expath}^\mathsf{c}(c)$; or

3. $n_1, n_2 \in \mathcal{N}_{trpt}$, $appmsg(n_1) = appmsg(n_2)$, $chan(n_1) = chan(n_2)$ and p transports the application-layer message.

Proof. Let $\mathcal{B}$ and n_2 be as per the lemma and suppose that (1) and (2) do not hold. Then, it must be the case that $n_2 \in \mathcal{N}_{trpt}$. Let p_c be the longest constructive penetrator path ending at n_2 such that p_c does not traverse a key-edge and $\mathsf{expath}^\sim(p_c)$ is a suffix of $\mathsf{expath}^\mathsf{c}(c)$. Also, let n_2' be the start node of p_c and es be such that $es \hat{}\, \mathsf{expath}^\sim(p_c) = \mathsf{expath}^\mathsf{c}(c)$, as illustrated in Figure 5.5. Clearly, as (2) does not hold (with p_c in the place of p), Decrypt in es and hence there exists an extraction path $es' < es$ such that $extract(es', msg(n_2')) = \{\!|m|\!\}_k$. Further, as $es' < es$, $\mathsf{expath}^\sim(p_c) \hat{}\, es' < \mathsf{expath}^\mathsf{c}(c)$; so by definition of $\mathcal{E}_{trpt}$, $\{\!|m|\!\}_k \in \mathcal{E}_{trpt}^{chan(n_2)}$.

We show that there is a destructive path p' starting at a regular or initial node, ending at n_2', such that $\{\!|m|\!\}_k \sqsubseteq msg(n)$ for every node n on p'. Suppose otherwise. It therefore follows that there must exist a node n such that $msg(n) = m$ and there is a constructive path p between n and n_2' that does not traverse a key-edge. Hence, $\mathsf{expath}^\sim(p \hat{}\, p_c) = \langle \mathsf{Decrypt} \rangle \hat{}\, es' \hat{}\, \mathsf{expath}^\sim(p_c)$ and is therefore a suffix of $\mathsf{expath}^\mathsf{c}(c)$. However, this contradicts the maximality of p_c. Thus, there exists a penetrator path p', ending at n_2' such that for every node n on p', $\{\!|m|\!\}_k \sqsubseteq msg(n)$. Further, p' must

be destructive as otherwise p_c could be extended to a longer path, contradicting its maximality. Lemma 5.21 can then be applied (with $t_1 = msg(n_1)$, $t_2 = msg(n_2)$, $es_1 = \mathsf{expath}^{\sim}(p')$ and $es_2 = \mathsf{expath}(p_c)$) to deduce that $\mathsf{expath}^{\sim}(p') = \mathsf{expath}(p_c)$.

Consider the starting node n_1 of $p'^{\frown}p_c$; by Assumption 5.9 (5) it cannot lie on a M strand. Hence, n_1 must be a regular node and therefore, by Lemma 5.11 it follows that $n_1 \in \mathcal{N}_{trpt}$. Thus, by Lemma 5.12, $appmsg(n_1) = appmsg(n_2)$ and $chan(n_1) = chan(n_2)$, as required. Further, note that $\mathsf{expath}(p_c) < \mathsf{expath}^{\mathsf{c}}(chan(n))$ as otherwise a node n would exist such that $msg(n) = m$, contradicting the above paragraph. Hence, p transports the application-layer message, as required. □

5.4.2 Making Nodes Abstractly Constructible

In this section we consider how to transform a bundle in order to make a given penetrator node abstractly-constructible (cf. Definition 4.26). In order to define this transformation, we begin by proving several lemmas that show how the $\mathsf{ingredient}$ relation is preserved by penetrator paths. This will be used to show that the penetrator only uses application-layer ingredients to construct application-layer messages.

Lemma 5.26. If p is a constructive penetrator path in a normal bundle $\mathcal{B}$ then $msg(p(1))$ $\mathsf{ingredient}$ $msg(p(|p|))$.

Proof. Since p can only contain KG, C, and E strands, the required result immediately follows from the definition of $\mathsf{ingredient}$. □

A weaker result can be proven for destructive penetrator paths. In particular, we now prove that if a destructive penetrator path ends with a message in $\mathcal{A}_{app}$, then there must be some subterm of the first message that is in $\mathcal{A}_{app}$. This result is complicated by the presence of key-edges. However, thanks to the assumptions, it follows that if an encryption $\{\!|m|\!\}_k$ contains an application-layer ingredient inside m, then it can only be using encryption keys that are application-layer ingredients, thus setting up an inductive argument.

Lemma 5.27. Let p be a directly-contributing destructive penetrator path in a normal crossing-path-free bundle $\mathcal{B}$ such that: p starts at a penetrator node; $msg(p(|p|)) \in \mathcal{A}_{app}$; and, for all nodes n on p that are not on M strands, $msg(n) \notin \mathcal{A}_{\mathcal{P}}$. Then there exists $t \in \mathcal{A}_{app}$ such that $t \sqsubseteq msg(p(1))$.

Proof. Let $\mathcal{B}$ and p be as per the lemma. Consider the sequence of nodes n_i on p such that $p(|p|)$ is the last node, and the remaining nodes are the positive nodes that send via a key-edge to a node on a D strand. We prove, by a backwards induction, that for each i, $msg(n_i) \in \mathcal{A}_{app}$. Note that this immediately implies the required result, since the destructive penetrator path between $p(1)$ and n_1 does not traverse any key edges, and thus $msg(p(1)) \sqsupseteq msg(n_1) \in \mathcal{A}_{app}$, as required.

The base case of the induction follows immediately from the assumption that $msg(p(|p|)) \in \mathcal{A}_{app}$. For the inductive case, suppose that that $msg(n_{i+1}) \in \mathcal{A}_{app}$ and let p_d be the destructive penetrator path between n_i and n_{i+1}. By definition of n_i, it follows there must be a key edge must be from $n_i = p_d(1)$ to $p_d(2)$, where $p_d(2)$ must be the first node on a D strand $\langle -k^{-1}, -\{\!|m|\!\}_k, +m\rangle$. Further, by the inductive hypothesis and the fact that the path between $p_d(2)$ and $p_d(|p_d|)$ does not traverse a key-edge, it follows that $m \sqsupseteq msg(n_{i+1}) \in \mathcal{A}_{app}$. We prove that

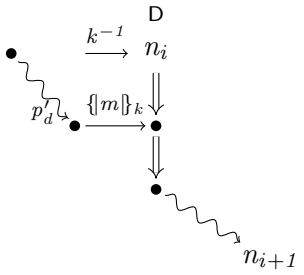


Figure 5.6: An illustration of Lemma 5.27.

this implies $\{\!|m|\!\}_k \in \mathcal{E}_{app}$, which therefore implies that $msg(n_i) = k^{-1} \in \mathcal{A}_{app}$, as required.

Let p'_d be the penetrator path that does not traverse any key-edges and leads to the second node on the D strand (i.e. p'_d is the source of $\{\!|m|\!\}_k$). This is illustrated in Figure 5.6. We prove that $\{\!|m|\!\}_k \in \mathcal{E}_{app}$ by a case analysis on $p'_d(1)$.

1. $p'_d(1)$ is a penetrator node and thus must lie on a M strand (it cannot be a E strand as p is normal). Since p'_d is destructive, $msg(n_{i+1}) \sqsubseteq msg(p'_d(1))$. Thus, since by assumption $msg(n_{i+1}) \notin \mathcal{A}_{\mathcal{P}}$, by Assumption 5.9 (6), every encryption that encloses $msg(n_{i+1})$ in $msg(p'_d(1))$ is in $\mathcal{E}_{app}$. In particular, $\{\!|m|\!\}_k \in \mathcal{E}_{app}$, as required.

2. $p'_d(1)$ is a regular node. Since p is directly-contributing and $p_d(1)$ lies on the key-edge of a D strand, it follows that $p'_d \hat{} p_n$, where p_n is the suffix of p starting at the positive node on the D strand, is also directly-contributing. Since $p'_d \hat{} p_n$ is directly-contributing either $p'_d(1) \in \mathcal{N}_\perp$, or $p'_d(1) \in \mathcal{N}_{trpt}$. If $p'_d(1) \in \mathcal{N}_\perp$ it immediately follows that $\{\!|m|\!\}_k \in \mathcal{E}_{app}$ (since it is a subterm of $msg(p'_d(1)) \in \mathcal{T}_{app}$). Otherwise, $p'_d(1) \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p'_d(1))) \leq \mathsf{expath}(p'_d)$. Therefore, $\{\!|m|\!\}_k \sqsubseteq appmsg(p'_d(1))$ and hence $\{\!|m|\!\}_k \in \mathcal{E}_{app}$, as required.

Hence, $\{\!|m|\!\}_k \in \mathcal{E}_{app}$ and thus, as discussed above, $msg(n_i) \in \mathcal{A}_{app}$, as required. □

We now prove a technical lemma that states, informally, that if a normal penetrator path is directly-contributing (cf. Definition 4.25), then the message on the starting node of the penetrator path is in $\mathcal{A}_{app}$. This, in some sense, proves that the definition of $\mathcal{A}_{app}$ is sufficiently general to include all terms that the penetrator uses, as ingredients, to construct application-layer messages. This lemma will be used when proving Lemma 5.30 in order to show that even if a node is not abstractly constructible, the penetrator must only be using terms from $\mathcal{A}_{app}$.

In the following lemma, we consider the *turning points* of a normal penetrator path p. These are the series of nodes n_i^t on p such that the penetrator path to the next turning point is purely constructive or destructive. Unless a turning point is the first node on a penetrator path, note that they will be negative. For example, if the bundle contains no KG strands then there are at most three turning points: the first and last nodes on p, and the node that divides p into destructive and constructive paths, if such a node exists. Figure 5.7 illustrates the turning points on a normal penetrator path.

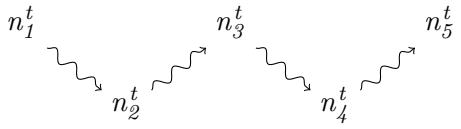


Figure 5.7: An illustration of the turning points on a normal penetrator path.

Lemma 5.28. Let $\mathcal{B}$ be a normal crossing-path-free low-level bundle and p be a directly-contributing normal penetrator path such that $p(1) \in \mathcal{N}_{trpt}$, $msg(p(|p|)) \in \mathcal{T}_{app}$, and for all nodes n on p that are not on M strands, $msg(n) \notin \mathcal{A}_{\mathcal{P}}$. Further, let $p_d \leq p$ be the maximal destructive penetrator path that does not traverse a key-edge such that $\mathsf{expath}\,(p_d) \leq \mathsf{expath}^{\mathsf{c}}\,(chan(p(1)))$. Then, $msg(p_d(|p_d|)) \in \mathcal{A}_{app}$.

Proof. Let $\mathcal{B}$, p and p_d be as per the lemma. Consider the node n such that $p_d(|p_d|) \rightarrow n$. By assumption, either: (1) n is the negative key node on a D strand, (2) $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\mathsf{c}}\,(chan(p(1)))$, or (3) n is a negative node on a constructive strand. If case (1) applies, it therefore follows that $t = msg(n)$ is an atom, and therefore that $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\mathsf{c}}\,(chan(p(1)))$. Hence, this reduces to case (2). If case (2) applies, then $t = appmsg(p(1)) \in \mathcal{T}_{app}$ and thus $t \in \mathcal{A}_{app}$, as required.

Otherwise, case (3) must apply and therefore n is a negative node on a constructive strand. We prove, by a backwards induction on the turning points n_i^t of p, that for all $i > 1$, $msg(n_i^t) \in \mathcal{A}_{app}$. This implies the required result since the node n is the second turning point. The base case of the induction is trivial, since by assumption $msg(p(|p|)) \in \mathcal{T}_{app} \subseteq \mathcal{A}_{app}$.

For the inductive case, consider the i^{th} turning point (for $i > 1$) n_i^t and assume that $msg(n_{i+1}^t) \in \mathcal{A}_{app}$. If the penetrator path between n_i^t and n_{i+1}^t is constructive, then the result immediately follows by Lemma 5.26. Otherwise, consider the destructive penetrator path p'_d between n_i^t and n_{i+1}^t (i.e. n_{i+1}^t is the first node on a constructive strand). n_i^t cannot be a regular node, since p is a penetrator path and thus only the first and last nodes on p can be regular (recall $i > 1$). Hence, n_i^t is a penetrator node. By definition of normality, it follows that n_i^t must lie on the key-edge of a D strand. Since the path to n_{i+1}^t is destructive, by Lemma 5.27 applied to n_i^t, $msg(n_i^t) \in \mathcal{A}_{app}$ (since $msg(n_i^t)$ is atomic), as required. □

Before proving the main result of this section we firstly state, for clarity, what it means for a penetrator path *not* to be abstractly-constructible.

Lemma 5.29. Let $\mathcal{B}$ be a normal low-level bundle. If a penetrator path p is not abstractly-constructible then either:

1. p starts at a regular node $n_1 \in \mathcal{N}_{non\text{-}payload}$; or
2. p starts at a regular node $n_1 \in \mathcal{N}_{trpt}$ but there exists no destructive penetrator subpath $p_d \leq p$ such that p_d $\mathsf{app\text{-}extracts}$ n_1.

Proof. This follows immediately from Definition 4.26. □

We now show how we can transform a non-abstractly-constructible node into an abstractly-constructible node, using an ACP transformation. Note in the following

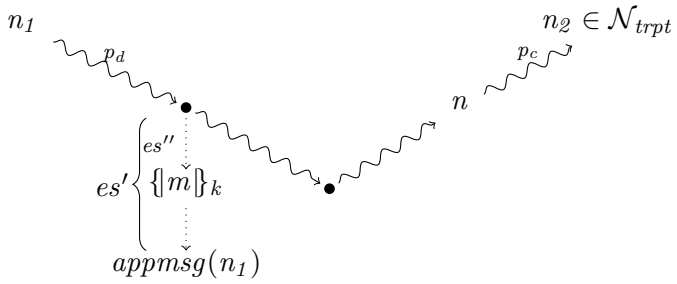


Figure 5.8: An illustration of Lemma 5.30.

we assume that the bundle is crossing-path-free. Under this assumption, it turns out that the only reason a node might not be abstractly-constructible is if the penetrator has not fully unpacked a transport-layer message, but has removed all the encryptions. Thus, we can make the node abstractly-constructible simply by adding extra S and C strands.

Lemma 5.30. Let $\mathcal{B}$ be a normal crossing-path-free low-level bundle and p_s be a message-construction penetrator path ending at a node $n_2 \in \mathcal{N}_{payload}$ that has a suffix p_c such that $\mathsf{expath}^{\sim}(p_c) = \mathsf{expath}^{\mathsf{c}}(chan(n_2))$. Then, there exists an application-layer equivalent crossing-path-free normal bundle $\mathcal{B}'$ such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and that can be reached by an ACP transformation in which $n \mathrel{\widehat{=}} p_c(1)$ is abstractly-constructible.

Proof. Let $\mathcal{B}$, n, n_2, p_s and p_c be as per the lemma. We show how to transform each non-abstractly-constructible directly-contributing penetrator path p that ends at n. Hence, let p be such a penetrator path.

Firstly, we simplify the bundle. If there exists a negative node n on p such that $msg(n) \in \mathcal{A}_{\mathcal{P}}$, but that the sender of n is not a M strand, then it follows that a new M strand $\langle +msg(n) \rangle$ can be created and connected to n. This results in a crossing-path-free normal bundle $\mathcal{B}'$ such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and that is obtainable via a ACP transformation, as per Lemma 5.19. Further, the resulting bundle is simpler, in that the length of p has strictly decreased. This means that this lemma can be safely applied recursively to ensure n is abstractly-constructible.

Otherwise, for all $i > 1$, $msg(n_i^t) \notin \mathcal{A}_{\mathcal{P}}$. Further, since p is not abstractly-constructible, by Lemma 5.29, either p starts at a node in $\mathcal{N}_{non\text{-}payload}$, or p starts at a node in $\mathcal{N}_{trpt}$ but there does not exist a destructive prefix p' of p such that p' $\mathsf{app\text{-}extracts}$ $p(1)$. However, if the first case applies then $p(1) \in \mathcal{N}_{non\text{-}payload}$ and thus p_s is a crossing-path, contradicting the assumption that $\mathcal{B}$ is crossing-path-free.

Hence, p starts at a node $n_1 \in \mathcal{N}_{trpt}$ but there exists no destructive penetrator subpath $p' \leq p$ such that p' $\mathsf{app\text{-}extracts}$ n_1. Let p_d be the longest destructive penetrator subpath such that $p_d \leq p$ and p_d does not traverse a key-edge. Let $c_1 = chan(n_1)$ and es be the longest common prefix of $\mathsf{expath}(p_d)$ and $\mathsf{expath}^{\mathsf{c}}(c_1)$, i.e. $es \leq \mathsf{expath}(p_d)$ and $es \leq \mathsf{expath}^{\mathsf{c}}(c_1)$. Note that, by the above assumption, $es < \mathsf{expath}^{\mathsf{c}}(c_1)$. This is illustrated in Figure 5.8.

If $es < \mathsf{expath}(p_d)$ then $\mathsf{expath}(p_d) \not\leq \mathsf{expath}^{\mathsf{c}}(c_1)$ and $\mathsf{expath}^{\mathsf{c}}(c_1) \not\leq \mathsf{expath}(p_d)$ and therefore it must be the case that p_s is a crossing-path, contradicting our assumption that $\mathcal{B}$ is crossing-path-free. Hence, this case cannot occur.

Otherwise, it follows that $es = \mathsf{expath}(p_d)$ and thus that $\mathsf{expath}(p_d) < \mathsf{expath}^{\mathsf{c}}(c_1)$. Consider es' such that $\mathsf{expath}(p_d)^\frown es' = \mathsf{expath}^{\mathsf{c}}(c_1)$. Suppose, for a contradiction, that Decrypt in es', i.e. that there exists an es'' such that $es''^\frown\langle\mathsf{Decrypt}\rangle \leq es'$. Then it immediately follows that $extract(msg(n_1), es^\frown es'') = \{\!|m|\!\}_k$ for some m and k. Hence, $\{\!|m|\!\}_k \in \mathcal{E}_{trpt}$. Further, since p is directly-contributing, by Lemma 5.28, $msg(p_d(|p_d|)) \in \mathcal{A}_{app}$ and thus, by definition of ingredient and $\mathcal{A}_{app}$, $\{\!|m|\!\}_k \in \mathcal{A}_{app}$. Therefore, $\{\!|m|\!\}_k \in \mathcal{A}_{app} \cap \mathcal{E}_{trpt}$, contradicting Assumption 5.9 (8).

Therefore, Decrypt in es' and thus extra S strands can be added in $\mathcal{B}'$ to extend p'_d to produce a penetrator path p'_d that ends at a node n'_1 such that $\mathsf{expath}(p'_d) = \mathsf{expath}^{\mathsf{c}}(c_1)$. Moreover, extra C strands can be added to ensure the bundle remains essentially unchanged, except that p_c is extended. Hence, $\mathcal{B} \trianglerighteq \mathcal{B}'$; further, $\mathcal{B}'$ is still normal, and by Lemma 5.19 (3), such a transformation is ACP.

Since new penetrator paths are introduced by the above transformation, we may have created new crossing-paths and thus it does not immediately follow that $\mathcal{B}'$ is crossing-path-free. In particular, p may actually be a crossing-path since, by adding the new S and C strand pairs, we may have created new Send or Fake subpaths, thus causing a new crossing-path to be uncovered (cf. Definition 5.10). These new crossing-paths can be removed by applying Lemma 5.24 to yield a bundle $\mathcal{B}''$ that is normal, crossing-path-free and obtainable via an ACP transformation. Further, p is still abstractly-constructible since Lemma 5.24 connects p to either:

1. An extension of an existing destructive penetrator path starting at a M strand, which is abstractly-constructible by definition (Lemma 5.24 Case (2)); or

2. An extension of an existing destructive penetrator path that has been extended with S and D strands to make it into a Receive or Learn subpath, which are by definition abstractly-constructible (Lemma 5.24 Case (1)).

Therefore, it follows that p is abstractly correct in $\mathcal{B}''$, as required. □

5.4.3 Making Nodes Interference-Free

We can now combine the results of the above sections and show that any node can be made interference-free.

Lemma 5.31. Let $\mathcal{B}$ be a normal crossing-path-free bundle with a negative regular node $n_2 \in \mathcal{N}_{payload}$ that is not interference-free. Then there exists, by an ACP transformation, a normal, crossing-path-free, bundle $\mathcal{B}'$ such that n_2 is interference-free and $\mathcal{B} \trianglerighteq \mathcal{B}'$.

Proof. Let $\mathcal{B}$ and n_2 be as per the lemma. We proceed by constructing $\mathcal{B}'$ from $\mathcal{B}$. By Lemma 5.25 applied to n_2 there must exist a penetrator path p such that either:

1. $n_2 \in \mathcal{N}_\perp$ and $n_1 \rightarrow n_2$; or

2. $n_2 \in \mathcal{N}_{trpt}$, p is constructive and there exists an extraction path es, not containing Decrypt, such that $es^\frown\mathsf{expath}^\sim(p) = \mathsf{expath}^{\mathsf{c}}(c)$; or

3. $n_1, n_2 \in \mathcal{N}_{trpt}$, $appmsg(n_1) = appmsg(n_2)$, $chan(n_1) = chan(n_2)$ and p transports the application-layer message.

Clearly, case (3) cannot apply as otherwise n_2 is already interference-free. Hence, suppose case (2) holds. It therefore follows that extra S strands followed by C strands could be added (care would need to be taken to avoid introducing non-normal paths) to yield a penetrator path p' such that $\mathsf{expath}^{\sim}(p') = \mathsf{expath}^{\mathsf{c}}(chan(n_2))$. Further, as p' is a message-construction path it follows, by Lemma 5.30, that the start node of p' can be made abstractly constructible. Hence, n_2 in $\mathcal{B}'$ is interference-free.

Alternatively, if case (1) applies and n_1 is regular then observe $n_1 \in \mathcal{N}_\perp$ as otherwise $n_1 \rightarrow n_2$ would be a crossing-path, contradicting the assumption that $\mathcal{B}$ is crossing-path-free. However, in such a case n_2 is already interference-free. Therefore, n_1 must be a penetrator node and thus Lemma 5.30 can be applied to n_1 to deduce a normal bundle $\mathcal{B}'$ such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and in which n_1 is abstractly constructible. Thus, n_2 in $\mathcal{B}'$ is interference-free and reachable by an ACP transformation. □

Remark The above lemma may be applied inductively since the only penetrator paths that are altered are those concerning n_2. Further, if the penetrator paths to n_2 were shared with other nodes then they could be duplicated before being altered. This ensures that any node that was interference-free in $\mathcal{B}$ will also be interference-free in the resulting bundle $\mathcal{B}'$, thus allowing the lemma to be applied repeatedly.

Recall that Proposition 4.31 required that each innocuous penetrator subpath in $\mathcal{B}$ is expanded. Thus, before we can apply the proposition, we need to ensure that all innocuous subpaths are expanded. Clearly, such a transformation should be possible, since an innocuous penetrator subpath is equivalent to a $\mathsf{Receive}$ subpath followed by a Send subpath. However, in general, this transformation would be very hard to specify, since arbitrary encryption keys could be required in order to achieve the necessary decryption or encryptions. Therefore, we instead make an assumption on the underlying transport-layer protocol that each innocuous penetrator subpath can be expanded. Whilst this does impose a proof obligation on the transport layer, it is essentially requiring that the transport layer allows the penetrator to send and receive messages from his own channel ends. Clearly, every sensible channel will satisfy this.

Assumption 5.32. Let $\mathcal{B}$ be a normal bundle and p be an innocuous non-expanded penetrator subpath between n and n'. Then there exists a bundle $\mathcal{B}'$, obtainable via a ACP transformation, in which p is expanded and such that $\mathcal{B} \trianglerighteq \mathcal{B}'$.

Corollary 5.33. Let $\mathcal{B}$ be a normal bundle of a low-level strand space Σ. Then, there exists, by an ACP transformation, a normal bundle $\mathcal{B}'$ in which every innocuous penetrator subpath has been expanded and such that $\mathcal{B} \trianglerighteq \mathcal{B}'$.

Using the above results we are now able to prove our main proposition as follows.

Proposition 5.34. Let $\mathcal{B}$ be a bundle of an strand space Σ that satisfies layer-disjoint encryption. Then there exists a normal, abstractable bundle $\mathcal{B}'$ of the enlarged strand space Σ_E such that $\mathcal{B} \trianglerighteq \mathcal{B}'$.

Proof. Let $\mathcal{B}$ be a bundle. By Lemma 5.6, there exists a normal bundle $\mathcal{B}_1$ such that $\mathcal{B} \trianglerighteq \mathcal{B}_1$. By definition of Σ_E, $\mathcal{B}_1$ is also a bundle of Σ_E. Further, Σ_E also satisfies layer-disjoint encryption, by Lemma 5.16. By Lemma 5.24, there exists a crossing-path-free bundle $\mathcal{B}_2$ such that $\mathcal{B}_1 \trianglerighteq \mathcal{B}_2$. Therefore, Lemma 5.31 can be applied

multiple times to yield a normal, interference-free, bundle $\mathcal{B}_3$ such that $\mathcal{B}_2 \trianglerighteq \mathcal{B}_3$. Lastly, Corollary 5.33 can be applied to expand the innocuous penetrator subpaths of $\mathcal{B}_3$ to yield a bundle $\mathcal{B}_4$. Further, by Assumption 5.9 (9), $\mathcal{B}$ is abstractly correct and therefore, as all transformations were ACP, $\mathcal{B}_4$ is abstractly correct. Thus, it follows by Proposition 4.31 that $\mathcal{B}_4$ is abstractable and $\mathcal{B} \trianglerighteq \mathcal{B}_4$, as required. □

5.5 Summary

In this chapter we have generalised the results of Chapter 4 and proven that whenever there exists a low-level bundle in a strand space satisfying our main condition, defined in Definition 5.8, then there exists a related (according to Definition 5.5) bundle that can be abstracted. The advantage of the result in this chapter is that the assumption is a statically-checkable condition, rather than the semantic condition from the prior chapter. Further, the condition allows us to understand more about the problem. For example, the required assumptions indicate where care should be taken by protocol designers to explicitly avoid key sharing.

We make use of this result in Chapter 6 in order to prove that whenever there is an attack against a bundle in the low-level model, there must be an attack against a bundle in the high-level model. This shows that the high-level strand spaces model is *sound*, in that if a proof of protocol correctness can be constructed in the high-level model, then there are no attacks against the corresponding low-level model.

In this chapter we began by defining several sets of encryptions which were used in the assumption, and the bundle relation $\trianglerighteq$, which formalises how a bundle is related to its abstractable version. Then, in Section 5.2 we defined the nine separate clauses that make up Assumption 5.9. In Section 5.3 we considered one of the more difficult cases of the proof, where the penetrator takes values from the transport layer and plays them into the application layer via a so-called crossing-path. In particular, we proved that all crossing-paths are incidental, in that they can be safely removed. This required carefully enlarging the strand space to include more values in $\mathcal{A}_\mathcal{P}$. In Section 5.4 we then showed that the assumptions defined in Section 5.2 imply that all bundles can be transformed to a related (using $\trianglerighteq$) bundle that is interference-free, allowing the results from Chapter 4 to be re-used.

Given Definition 5.15, when proving the correctness of the application-layer protocol, $\mathcal{A}_\mathcal{P}$ in the high-level strand space must include at least $\mathcal{A}_\mathcal{P}^{\Sigma_E}$. In practice, this is not an issue since $\mathcal{A}_\mathcal{P}^{\Sigma_E} = \mathcal{A}_\mathcal{P}^{\Sigma} \cup \mathcal{T}_X$ and $\mathcal{T}_X$ contains nothing from $\mathcal{T}_{app}$. However, from a practical point of view, it means that the assumptions that the application-layer makes on the contents of $\mathcal{A}_\mathcal{P}$ should be expressed like those of WebAuth in Assumption 3.21. In particular, the assumption should specify what $\mathcal{A}_\mathcal{P}$ *cannot* contain, rather than exactly what $\mathcal{A}_\mathcal{P}$ does contain.

We discuss related work at the end of Chapter 7, in Section 7.4.

Chapter 6

Abstracting Correctness Properties

In the previous chapters we have proven that, subject to certain assumptions, any low-level bundle can be transformed to a bundle that can be abstracted. Further, the transformed bundle has the same regular application-layer behaviour as the original bundle. In this chapter we prove that whenever there is an attack against a bundle in the low-level model, there must be an attack against a bundle in the high-level model. This shows that the high-level strand spaces model is *sound*, in that if a proof of protocol correctness can be constructed in the high-level model, then there are no attacks against the corresponding low-level model. We actually prove a slightly stronger version of the above result: we show that if a low-level bundle does not satisfy a given correctness property, it can be transformed to an abstractable bundle such that its abstraction does not satisfy the correctness property in question.

We firstly need to define what a correctness property is. In Section 6.1 we formalise a logic in which protocol-correctness properties can be expressed. This logic has not been designed to be complete, but can express all of the correctness properties for WebAuth and other protocols that we have looked at. We then define, in Section 6.2 and Section 6.3, two semantics for the logic, one that applies to low-level bundles and one that applies to high-level bundles. Clearly, it is important for the two semantics to be equivalent, modulo abstraction. In particular, in order to prove our main result we will require that if an abstractable low-level bundle does not satisfy a correctness property, then the abstracted high-level bundle must also not satisfy the property. In order to prove this result we, in Section 6.4, consider how the values of the terms of the logic are related when interpreted in both a low-level bundle and a high-level bundle that abstracts it. Then, in Section 6.5, we prove the required result and show that, for a large subclass of correctness properties, whenever an abstractable low-level bundle does not satisfy a correctness property, then its high-level abstraction also does not. In Section 6.6, we firstly show that the transformations applied in Chapter 5 preserve the correctness of bundle predicates. Then, we combine the results of this chapter and Proposition 5.34 and prove that whenever a low-level bundle does not satisfy a correctness property, the bundle can be transformed to an abstractable bundle such that its abstraction does not satisfy the correctness property.

6.1 A Logic of Correctness Properties

In this section we define a logic in which we can express correctness properties. Before defining the logic we consider what predicates the logic would require in order to express the WebAuth correctness properties from Section 3.3. By inspection we can see that the logic needs to be able to express:

1. A node either sends a message to, or causally precedes, another node;
2. A term (e.g. the user's password in WebAuth) is confidential or uniquely originating;
3. A channel is confidential or authentic;
4. Certain message components are equal (e.g. the request tokens in WebAuth were required to match between multiple messages);
5. A node lies on a particular type of strand (e.g. on an *AS1* strand).

The logic that we will define allows all of the above to be expressed, with the exception of a node sending a message *directly* to another node. The reason why we cannot express the latter is that it is, in general, too complex to be practical. Defining what it means in a normal high-level bundle is not particularly difficult; a reasonable definition might be that either the nodes must be directly connected using $\rightarrow$, or there must be a TX strand between them. However, we also need to define an equivalent semantics for both normal and non-normal low-level bundles, which is far more complicated. For example, in the low-level bundle the penetrator may receive an application-layer message (using a $\mathsf{Receive}$ subpath), and then incorporate the message as part of the transport-layer packaging (i.e. creating a crossing path). This could then be passed through any number of regular nodes, all as part of the transport-layer packaging, before being extracted. However, as the above path would abstract to a direct message path (i.e. a TX strand or a direct edge), any low-level definition would have to hold on such paths. Clearly it would be very difficult to define a property that covers such cases. Whilst this may appear to be a restriction of the logic, in reality this property is not required; what really matters is the application-layer behaviour of the regular agents and, to a lesser extent, the causal ordering (i.e. $\preceq$) of nodes. Therefore, we do not consider the lack of this property to be a major problem for the logic.

We start, by defining the terms in our logic, which we call *logical terms* (in contrast to message terms, from $\mathcal{A}$). The informal meaning behind the terms of the logic is given in Figure 6.1.

Definition 6.1. We assume the existence of a set of variables V. The set of *logical terms* is then defined by the following grammar, where c denotes a constant and v denotes a variable:

$$\begin{aligned} Term ::= {} & Term \hat{} Term \mid \{\!|Term|\!\}_{Term} \mid \mathsf{appmsg}(\widehat{Term}) \mid g(Term) \\ & \mid \mathsf{endpoint\_name}(Endpoint) \mid c \mid v \\ Natural ::= {} & \mathsf{height}(Strand) \mid \mathsf{seqno}(\widehat{Term}) \\ & \mid Natural\, \sigma\, Natural, \sigma \in \{+, -, \ldots\} \mid c \mid v \end{aligned}$$

Logical Term	Informal Meaning
v	$v \in V$, a variable
c	$c \in \mathcal{A} \cup \mathcal{P}(\mathcal{A})$, a constant term or set of terms
$\{\!\|t_1\|\!\}_{t_2}$	encryption of a logical term
$t_1 \hat{} t_2$	concatenation of two logical terms
$g(t)$	the application of the key-generation function g to t
$t_{t'}$	a channel endpoint with name t and channel end t'
$\sigma(t_1,\ t_2,\ t_3,\ t_4,\ t_5)$	$\sigma \in \{+, -\}$, a high-level term
$\mathsf{height}(t)$	bundle height of a strand
$\mathsf{msg}(t)$	high-level message of a node
$\mathsf{sign}(t)$	sign of a high-level term
$\mathsf{sender}(t)$	sending endpoint of a high-level term
$\mathsf{recipient}(t)$	receiving endpoint of a high-level term
$\mathsf{seqno}(t)$	sequence number of a high-level term
$\mathsf{appmsg}(t)$	application message of a high-level term
$\mathsf{channel}(t)$	channel of a high-level term
$t_1\ \sigma\ t_2$	$\sigma \in \{+, -, \ldots\}$, standard mathematical functions
$\mathsf{endpoint\_name}(t)$	name contained in an endpoint
$\mathsf{endpoint\_end}(t)$	channel end contained in an endpoint
$\mathsf{role_k}$	the set of strands for the regular agent in role k (e.g. for WebAuth this includes *LS1*, *LS2*, *AS1*, etc.)
$\mathsf{node}(t, n)$	n^{th} node on a strand t
$\mathsf{strand}(t)$	strand of a node

Figure 6.1: An informal explanation of the meaning of the terms of our correctness property logic.

Predicate	Informal Meaning
$t_1 \preceq t_2$	t_1 causally precedes t_2
$\mathsf{AuthChan}(t)$	t is an authenticated channel
$\mathsf{ConfChan}(t)$	t is a confidential channel
$\mathsf{Confidential}(t)$	t is a confidential term
$\mathsf{UniquelyOriginates}(t_1, t_2)$	t_1 uniquely originates at the node t_2
$t_1\ \sigma\ t_2$	$\sigma \in \{\sqsubseteq, =, <\}$, comparison of t_1 and t_2
$\forall v \in t \cdot \phi$	universal quantification
$\exists v \in t \cdot \phi$	existential quantification
$\neg\phi_1$	logical negation
$\phi_1 \wedge \phi_2$	logical and
$\phi_1 \vee \phi_2$	logical or

Figure 6.2: An informal explanation of the bundle predicates.

$$
\begin{aligned}
ChannelEnd &::= \mathsf{endpoint\_end}(Endpoint) \mid c \mid v \\
Endpoint &::= Term_{ChannelEnd} \mid \mathsf{sender}(\widehat{Term}) \mid \mathsf{recipient}(\widehat{Term}) \mid c \mid v \\
\widehat{Term} &::= \mathsf{msg}(Node) \\
&\quad \mid Sign(Endpoint,\ Endpoint,\ Natural,\ Term,\ Channel) \mid v \\
Channel &::= \mathsf{channel}(\widehat{Term}) \mid c \mid v \\
Node &::= \mathsf{node}(Strand, Natural) \mid v \\
Strand &::= \mathsf{strand}(Node) \mid v \\
Sign &::= \mathsf{sign}(\widehat{Term}) \mid + \mid - \mid v \\
Set(Term) &::= c \mid v \\
Set(Strand) &::= \mathsf{role_k} \mid v
\end{aligned}
$$

We denote the least-fixed point of the above by $LogicalTerm$. A logical term is *well-typed* with respect to a particular variable assignment Γ iff every variable and constant has been instantiated with a value of the correct type.

Using the above definition of logical terms we are now able to define what a *bundle predicate* is. The informal meaning behind the predicates of the logic is given in Figure 6.2.

Definition 6.2. The set of *high-level bundle predicates* is given by the following grammar:

$$
\begin{aligned}
Formula ::= {} & Node \preceq Node \mid \mathsf{AuthChan}(Channel) \mid \mathsf{ConfChan}(Channel) \\
& \mid \mathsf{Confidential}(Term) \mid \mathsf{UniquelyOriginates}(Term, Node) \\
& \mid LogicalTerm \sqsubseteq LogicalTerm \mid LogicalTerm = LogicalTerm \\
& \mid Natural < Natural \mid Formula \wedge Formula \\
& \mid Formula \vee Formula \mid \neg Formula \\
& \mid \forall v \in Set(Term) \cdot Formula \mid \exists v \in Set(Term) \cdot Formula \\
& \mid \forall v \in Set(Strand) \cdot Formula \mid \exists v \in Set(Strand) \cdot Formula \\
& \mid \forall v \in \mathbb{N} \cdot Formula \mid \exists v \in \mathbb{N} \cdot Formula
\end{aligned}
$$

A *free variable* in a high-level bundle predicate is a variable v that is not enclosed by a quantification of the form $\forall v \in S$ or $\exists v \in S$. A high-level bundle predicate ϕ is *closed* iff it contains no free variables.

In the definition above we include a number of cases that could be removed by using logical equivalences (e.g. $\phi_1 \wedge \phi_2 = \neg(\neg\phi_1 \vee \neg\phi_2)$). We include them in order to allow us to obtain formulas in *negation normal form*, where negations only occur next to atomic predicates. This is used in several of the proofs in this section, including Proposition 6.19.

Whilst the above logic may appear to have relatively few predicates, note that it is possible to express all of the WebAuth correctness properties. Further, to ease the expression of correctness predicates we can also define some macros as follows. For example, we can express the strand ordering relation, $\Rightarrow$, as follows:

$$t_1 \Rightarrow t_2 \mathrel{\widehat{=}} \mathsf{strand}(t_1) = \mathsf{strand}(t_2) \\ \wedge \exists i \in \mathbb{N} \cdot \mathsf{node}(\mathsf{strand}(t_1), i) = t_1 \wedge \mathsf{node}(\mathsf{strand}(t_2), i+1) = t_2.$$

$\Rightarrow^+$ can be expressed as:

$$\mathsf{t_1} \Rightarrow^+ \mathsf{t_2} \mathrel{\widehat{=}} \mathsf{strand}(t_1) = \mathsf{strand}(t_2) \wedge \exists i \in \mathbb{N} \cdot \exists j \in \mathbb{N} \cdot \\ i < j \wedge \mathsf{node}(\mathsf{strand}(t_1), i) = t_1 \wedge \mathsf{node}(\mathsf{strand}(t_2), j) = t_2.$$

Lastly, we can test for set membership as follows:

$$\mathsf{member}(t, S) \mathrel{\widehat{=}} \exists x \in S \cdot x = t.$$

This macro, when combined with constant sets of terms, can be used to test if values are in $\mathcal{A}_\mathcal{P}$, or $\mathcal{A}^*_\mathcal{P}$, for instance. When defining high-level bundle predicates we will use, without loss of generality, the above macros.

As an example, we can express Proposition 3.25 (1) as a bundle predicate as follows:

$$\forall LS \in \mathcal{T}^{reg}_{names} \cdot \forall U \in \mathcal{T}_{names} \cdot \forall AS' \in \mathcal{T}^{reg}_{names} \cdot \forall \psi_3 \in \mathcal{C} \cdot \forall \phi_3 \in \mathcal{C} \cdot \forall k \in \mathcal{K} \cdot \\ \forall r_2 \in \mathcal{A} \cdot \forall st^2_{LS} \in \mathsf{role_{LS2}} \cdot \\ \mathsf{height}(st^2_{LS}) = 2 \wedge \mathsf{msg}(\mathsf{node}(st^2_{LS}, 1)) = \\ -(?_{\psi_3}, LS_{\phi_3}, U \hat{}\, passwd_{LS}(U) \hat{}\, \{|\mathsf{req} \hat{}\, r_2|\}_k \hat{}\, \{|\mathsf{webkdc\_service} \hat{}\, AS' \hat{}\, k|\}_{SK(LS)}, \\ {}_{TLS^{C \to S}}) \\ \Longrightarrow \exists \psi_1 \in \mathcal{C} \cdot \exists \phi_1 \in \mathcal{C} \cdot \exists stok \in \mathcal{A} \cdot \exists st^1_{AS} \in \mathsf{role_{AS1}} \cdot \\ \mathsf{height}(st^1_{AS}) = 2 \\ \wedge \mathsf{msg}(\mathsf{node}(st^1_{AS}, 2)) = +(AS'_{\phi_1},\ ?_{\psi_1},\ \{|\mathsf{req} \hat{}\, r_2|\}_{Sh^{AS'}_{LS}} \hat{}\, stok,\ {}_{TLS^{S \to C}}).$$

Remark 6.3. Since any statement of Peano arithmetic can be translated into a high-level bundle predicate, preserving satisfaction, it follows that the set of valid formulas of the logic is not recursively enumerable. Thus, the logic is not *axiomatizable*, in that the set of valid formulas cannot be enumerated from a set of axioms. In practice, this means that the logic is not suitable for analysis by methods that make use of a proof theory.

Note also that the logic is not closed under isomorphism; i.e. if there exist two strand spaces that have an isomorphism between them, then satisfaction of a formula in one strand space does not necessarily imply that it is satisfied in the other. This is because logical terms allow algebraic terms to be encoded directly. Hence, if the two strand spaces differ in how they represent a certain value then it is possible for a formula to be satisfied by one strand space but not the other. For example, let: ϕ be the open formula $x = c$, where $x \in V$ and c is a term constant (i.e. $c \in \mathcal{A}$); Γ be a substitution such that $\Gamma(x) = c$; H be an isomorphism that exchanges c with a $c' \neq c$. Hence, ϕ is satisfied under Γ since $\Gamma(x) = c$, but not under $H \circ \Gamma$, since $(H \circ \Gamma)(x) = c' \neq c$.

Since the aim of this chapter is to define a logic for the purposes of proving that the high-level strand spaces model is sound, we do not consider the above properties to be essential. More importantly, in Proposition 6.22 we will prove that satisfaction of all formulas of the logic is preserved under abstraction. Thus, if a well-behaved sub-logic is defined, then it follows that satisfaction of the sub-logic is still preserved under abstraction. Since we are focused on proving the preservation of satisfaction by abstraction, we leave the development of such a sub-logic for future work.

6.2 High-Level Semantics

We now consider how to define the semantics of this logic when applied to both low and high-level bundles. We start by considering the easier case of the high-level semantics. In order to define this we will need to assume the existence of a set of high-level strands, $\widehat{Role_k}$, for each regular role k. For example, in the case of WebAuth, this would mean that we had a set of strands that represent all possible *AS1* strands.

We define the semantics relative to a particular bundle as follows. Firstly, we define the set of *logical term values* that includes all values that logical terms will be evaluated to. We then define a function that, given a logical term, returns the logical term value for the current bundle. We then define the satisfaction relation between high-level bundles and high-level bundle predicates using the above interpretation function. In the following $height_{\hat{\mathcal{B}}}(st)$ denotes the $\hat{\mathcal{B}}$-height of the strand st.

Definition 6.4. The set of *high-level logical term values* in a high-level strand space $\hat{\Sigma}$ is given by:

$$\mathcal{A} \cup \mathbb{N} \cup \mathcal{C} \cup \mathcal{I} \cup \hat{\mathcal{A}} \cup Channels \cup \hat{\mathcal{N}} \cup \hat{\Sigma} \cup \{+, -\} \cup \mathcal{P}(\mathcal{A}) \cup \mathcal{P}\left(\hat{\Sigma}\right).$$

Let $\hat{\mathcal{B}}$ be a high-level bundle, t be a logical term and $\hat{\Gamma}$ be a type-respecting *high-level term map*, from variables to high-level logical term values, defined on all free-variables in t such that t is well-typed. Then, the *high-level logical term value* of t in $\hat{\mathcal{B}}$, denoted $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$, is defined inductively as follows:

- $[\![v]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\widehat{=}} \hat{\Gamma}(v)$;
- $[\![c]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\widehat{=}} c$;
- $[\![\{\!|t_1|\!\}_{t_2}]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\widehat{=}} \{\!|[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}|\!\}_{[\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}}$;

- $[\![t_1 \hat{} t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} [\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \hat{} [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$;
- $[\![t_{t'}]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} ([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}, [\![t']\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$;
- $[\![g(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} g([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$;
- $[\![\mathsf{height}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} height_{\hat{\mathcal{B}}}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$;
- $[\![\mathsf{sign}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} \sigma$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![\mathsf{msg}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} msg([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$;
- $[\![\mathsf{sender}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} A_\phi$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![\mathsf{recipient}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} B_\psi$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![\mathsf{seqno}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} i$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![\mathsf{appmsg}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} m$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![\mathsf{channel}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} c$, where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = \sigma(A_\phi,\ B_\psi,\ i,\ m,\ c)$;
- $[\![t_1\ \sigma\ t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} [\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}\, \sigma [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$, where $\sigma \in \{+, -, \ldots\}$;
- $[\![\mathsf{endpoint\_name}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} A$ where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = A_\phi$;
- $[\![\mathsf{endpoint\_end}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} \phi$ where $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} = A_\phi$;
- $[\![\mathsf{role_k}]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} \widehat{Role}_k \cap \{strand(\hat{n}) \mid \hat{n} \in \mathcal{N}^{reg}_{\hat{\mathcal{B}}}\}$;
- $[\![\mathsf{node}(t, n)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} ([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}, [\![n]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$, providing $[\![n]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \leq height_{\hat{\mathcal{B}}}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$;
- $[\![\mathsf{strand}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \mathrel{\hat{=}} strand([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$.

Let $\hat{\mathcal{B}}$ be a high-level bundle, ϕ be a high-level bundle predicate, and $\hat{\Gamma}$ be a a high-level term map defined on all free-variables of ϕ. Then, $\hat{\mathcal{B}}$ and $\hat{\Gamma}$ *satisfy* ϕ, denoted $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} \phi$, according to the following inductive definition:

- $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} t_1 \preceq t_2$ iff $[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \preceq_{\hat{\mathcal{B}}} [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} \mathsf{AuthChan}(t)$ iff $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ satisfies $\mathcal{A}$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} \mathsf{ConfChan}(t)$ iff $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ satisfies $\mathcal{C}$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} \mathsf{Confidential}(t)$ iff $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ is confidential (Definition 3.14) in $\hat{\mathcal{B}}$ and $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \in \mathcal{A}_{app}$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \mathrel{\hat{\models}} \mathsf{UniquelyOriginates}(t_1, t_2)$ iff $[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \in \mathcal{T} \setminus \mathcal{A}^*_{\mathcal{P}}$ and uniquely originates on $[\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$;

- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} t_1 \ \sigma \ t_2$, $\sigma \in \{\sqsubseteq, =, <\}$ iff $[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \ \sigma \ [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \forall v \in S \cdot \phi$ iff for all $t \in [\![S]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$, $(\hat{\mathcal{B}}, \hat{\Gamma} \cup \{v \mapsto t\}) \hat{\models} \phi$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \exists v \in S \cdot \phi$ iff there exists $t \in [\![S]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$, $(\hat{\mathcal{B}}, \hat{\Gamma} \cup \{v \mapsto t\}) \hat{\models} \phi$.
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \neg\phi_1$ iff $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\not\models} \phi_1$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_1 \wedge \phi_2$ iff $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_1$ and $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_2$;
- $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_1 \vee \phi_2$ iff $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_1$ or $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi_2$.

In the semantics of $\mathsf{Confidential}(t)$ we require t to be in $\mathcal{A}_{app}$, to ensure that the predicate does not claim that a transport-layer value is confidential. If the predicate were to require that a transport-layer nonce, for instance, was confidential then clearly this could be true in the high-level bundle, as it may have been abstracted away, but may not be true in the low-level bundle. Further, this matches the intention of the logic; we are interested in specifying that the application-layer protocol is correct, not that certain transport-layer details are correct.

Note that in the semantics of $\mathsf{UniquelyOriginates}(t_1, t_2)$, we restrict our attention to atoms that the penetrator cannot obtain from a term that he initially knows (i.e. atoms in $\mathcal{T} \setminus \mathcal{A}^*_{\mathcal{P}}$). This is to simplify the low-level semantics, which we define below, as the low-level semantics has to correspond to the high-level semantics. However, as we are only interested in writing correctness properties that require values that regular agents created to uniquely originate, this is not a problematic restriction.

6.3 Low-Level Semantics

In this section we develop a semantics for high-level bundle predicates when applied to low-level bundles. As we aim to prove that the correctness of bundle predicates can be abstracted, we take care to define the semantics of the logic in a way that is both intuitive and compatible with the high-level semantics.

6.3.1 Unique Origination

In order to define a low-level semantics that corresponds to the high-level semantics we need to consider how to interpret $\mathsf{UniquelyOriginates}(t_1, t_2)$ in the low-level bundle. Unfortunately, we cannot map it directly to the standard notion of unique origination in the low-level bundle, as this would also consider values that originate as part of the transport-layer packaging. Hence, we introduce a new definition that gives the set of values that originate in the *application layer* in low-level bundles.

Definition 6.5. Let $\mathcal{B}$ be a low-level bundle. An atom $t \notin \mathcal{A}^*_{\mathcal{P}}$ *originates in the application layer* at a regular node $n \in \mathcal{N}_{\mathcal{B}}$ iff $t \sqsubseteq \mathit{appmsg}(n)$ and, for all n' such that $n' \Rightarrow^+ n$, $t \not\sqsubseteq \mathit{appmsg}(n')$. An atom $t \notin \mathcal{A}^*_{\mathcal{P}}$ *uniquely originates in the application layer* at a regular node $n \in \mathcal{N}_{\mathcal{B}}$ iff t originates in the application layer on a unique n.

6.3.2 The Semantics

We now consider how to define the semantics of high-level bundle predicates when applied to low-level bundles.

We need to assume a few things about how the strand spaces are defined. Firstly, we assume that for each regular role there is a set of strands $Role_k$. Clearly, we will need this set to be related to $\widehat{Role}_k$ and we formalise this, in Assumption 6.10. We also define several partial functions:

- $app\_node(st, n)$ gives the the index of the n^{th} application-layer node on st:

$$app\_node(st, n) \mathrel{\widehat{=}} i \text{ where } (st, i) \in \mathcal{N}_{payload} \wedge app\_node^{-1}(st, i) = n.$$

- $app\_node^{-1}(st, n)$ gives the application-layer index of the n^{th} node if one exists:

$$app\_node^{-1}(st, n) \mathrel{\widehat{=}} \#\{i \mid 1 \leq i \leq n \wedge (st, i) \in \mathcal{N}_{payload}\} \text{ if } (st, n) \in \mathcal{N}_{payload}.$$

- $app\_height_{\mathcal{B}}(st)$ gives the application-layer height of the transport-layer strand st from $\mathcal{B}$:

$$app\_height_{\mathcal{B}}(st) \mathrel{\widehat{=}} \#\{i \mid (st, i) \in \mathcal{N}_{\mathcal{B}} \cap \mathcal{N}_{payload}\}.$$

We also need to define what it means for a term t to be confidential in a low-level bundle, analogously to Definition 3.14.

Definition 6.6. A term t is *confidential* in a low-level bundle $\mathcal{B}$ iff no equivalent bundle contains a positive node n such that $msg(n) = +t$.

Definition 6.7. The set of *low-level logical term values* in a low-level strand space Σ is given by:

$$\mathcal{A} \cup \mathbb{N} \cup \mathcal{C} \cup \mathcal{I} \cup \hat{\mathcal{A}} \cup Channels \cup \mathcal{N} \cup \Sigma \cup \{+, -\} \cup \mathcal{P}(\mathcal{A}) \cup \mathcal{P}(\Sigma).$$

Let $\mathcal{B}$ be a low-level bundle, t be a logical term and Γ be a type-respecting *low-level term map*, from variables to low-level logical term values, defined on all free-variables in t, such that t is well-typed. Then, the *low-level logical term value* of t in $\mathcal{B}$, denoted $[\![t]\!]^{\Gamma}_{\mathcal{B}}$, is defined inductively as follows (for ease we omit cases that are equivalent to the high-level semantics):

- $[\![\mathsf{node}(t, n)]\!]^{\Gamma}_{\mathcal{B}} \mathrel{\widehat{=}} ([\![t]\!]^{\Gamma}_{\mathcal{B}}, app\_node([\![t]\!]^{\Gamma}_{\mathcal{B}}, [\![n]\!]^{\Gamma}_{\mathcal{B}}))$, providing $[\![n]\!]^{\Gamma}_{\mathcal{B}} \leq app\_height_{\mathcal{B}}([\![t]\!]^{\Gamma}_{\mathcal{B}})$;
- $[\![\mathsf{strand}(t)]\!]^{\Gamma}_{\mathcal{B}} \mathrel{\widehat{=}} strand([\![t]\!]^{\Gamma}_{\mathcal{B}})$;
- $[\![\mathsf{height}(t)]\!]^{\Gamma}_{\mathcal{B}} \mathrel{\widehat{=}} app\_height_{\mathcal{B}}([\![t]\!]^{\Gamma}_{\mathcal{B}})$;
- $[\![\mathsf{role_k}]\!]^{\Gamma}_{\mathcal{B}} \mathrel{\widehat{=}} Role_k \cap \{strand(n) \mid n \in \mathcal{N}^{reg}_{\mathcal{B}}\}$;
- $[\![\mathsf{msg}(t)]\!]^{\Gamma}_{\mathcal{B}} \mathrel{\widehat{=}} \hat{\alpha}_{\mathcal{B}}([\![t]\!]^{\Gamma}_{\mathcal{B}})$.

Let $\mathcal{B}$ be a low-level bundle, ϕ be a high-level bundle predicate, and Γ be a low-level term map defined on all free-variables of ϕ. Then, $\mathcal{B}$ and Γ *satisfy* ϕ, denoted $(\mathcal{B}, \Gamma) \vDash \phi$, according to the following inductive definition:

- $(\mathcal{B}, \Gamma) \vDash t_1 \preceq t_2$ iff $[\![t_1]\!]^\Gamma_\mathcal{B} \preceq_\mathcal{B} [\![t_2]\!]^\Gamma_\mathcal{B}$;
- $(\mathcal{B}, \Gamma) \vDash \mathsf{AuthChan}(t)$ iff $[\![t]\!]^\Gamma_\mathcal{B}$ satisfies $\mathcal{A}$;
- $(\mathcal{B}, \Gamma) \vDash \mathsf{ConfChan}(t)$ iff $[\![t]\!]^\Gamma_\mathcal{B}$ satisfies $\mathcal{C}$;
- $(\mathcal{B}, \Gamma) \vDash \mathsf{Confidential}(t)$ iff $[\![t]\!]^\Gamma_\mathcal{B}$ is confidential in $\mathcal{B}$ and $[\![t]\!]^\Gamma_\mathcal{B} \in \mathcal{A}_{app}$;
- $(\mathcal{B}, \Gamma) \vDash \mathsf{UniquelyOriginates}(t_1, t_2)$ iff $[\![t_1]\!]^\Gamma_\mathcal{B} \in \mathcal{T} \setminus \mathcal{A}^*_\mathcal{P}$ and uniquely originates in the application layer on $[\![t_2]\!]^\Gamma_\mathcal{B}$;
- $(\mathcal{B}, \Gamma) \vDash t_1 \ \sigma \ t_2$, $\sigma \in \{\sqsubseteq, =, <\}$ iff $[\![t_1]\!]^\Gamma_\mathcal{B} \sigma [\![t_2]\!]^\Gamma_\mathcal{B}$;
- $(\mathcal{B}, \Gamma) \vDash \phi_1 \wedge \phi_2$ iff $(\mathcal{B}, \Gamma) \vDash \phi_1$ and $(\mathcal{B}, \Gamma) \vDash \phi_2$;
- $(\mathcal{B}, \Gamma) \vDash \phi_1 \vee \phi_2$ iff $(\mathcal{B}, \Gamma) \vDash \phi_1$ or $(\mathcal{B}, \Gamma) \vDash \phi_2$;
- $(\mathcal{B}, \Gamma) \vDash \neg\phi_1$ iff $\mathcal{B} \nvDash \phi_1$;
- $(\mathcal{B}, \Gamma) \vDash \forall v \in S \cdot \phi$ iff for all $t \in [\![S]\!]^\Gamma_\mathcal{B}$, $(\mathcal{B}, \Gamma \cup \{v \mapsto t\}) \vDash \phi$;
- $(\mathcal{B}, \Gamma) \vDash \exists v \in S \cdot \phi$ iff there exists $t \in [\![S]\!]^\Gamma_\mathcal{B}$, $(\mathcal{B}, \Gamma \cup \{v \mapsto t\}) \vDash \phi$.

6.4 Term Equivalence

In order to prove that the low and high-level semantics of our logic correspond, we firstly consider the slightly simpler problem of relating the terms of the logic. We start by defining a relation $\nearrow_{\hat{\psi}}$ that pairs low-level term values (which could be terms, nodes, strands etc) with the high-level term value that they abstract to. We then prove a number of lemmas that build towards proving that, given a logical term t and a high-level bundle $\hat{\mathcal{B}}$ that abstracts a low-level bundle $\mathcal{B}$, $[\![t]\!]^\Gamma_\mathcal{B} \nearrow_{\hat{\psi}} [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$.

Definition 6.8. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a bundle that abstracts it using a node map $\hat{\psi}$. x is *abstracted by* y, denoted $x \nearrow_{\hat{\psi}} y$, iff one of the following clauses applies:

- $x, y \in \mathcal{A} \cup \mathbb{N} \cup \mathcal{C} \cup \mathcal{I} \cup \hat{\mathcal{A}} \cup \mathit{Channels} \cup \{+, -\}$ and $x = y$;
- $x \in \mathcal{N}_\mathcal{B}$, $y \in \mathcal{N}_{\hat{\mathcal{B}}}$ and $x \ \hat{\psi} \ y$;
- x is a low-level strand, y is a high-level strand, $\mathit{app\_height}_\mathcal{B}(x) = \mathit{height}_{\hat{\mathcal{B}}}(y)$ and, for all indices $i \leq \mathit{height}_{\hat{\mathcal{B}}}(y)$, $(x, \mathit{app\_node}(x, i)) \nearrow_{\hat{\psi}} (y, i)$;
- x is a set of strands, y is a set of strands and there exists a bijection μ between x and y such that, for each pair $(st, st') \in \mu$, $st \nearrow_{\hat{\psi}} st'$.

We lift the above definition to apply to logical term environments. $\Gamma \nearrow_{\hat{\psi}} \hat{\Gamma}$ iff Γ and $\hat{\Gamma}$ are defined on the same variables and for each variable v, $\Gamma(v) \nearrow_{\hat{\psi}} \Gamma(v)$.

Note that the above definition has a clause for each possible logical term type.

Lemma 6.9. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts $\mathcal{B}$ using a node map $\hat{\psi}$. Further, let $n \in \mathcal{B}$ and $\hat{n} \in \hat{\mathcal{B}}$ be regular nodes such that $n \mathrel{\hat{\psi}} \hat{n}$. Then, $\mathit{strand}(n) \nearrow_{\hat{\psi}} \mathit{strand}(\hat{n})$.

Proof. This follows immediately from the definitions. □

In order to prove compatibility we need to assume that the set of regular strands (i.e. Role_k and $\widehat{\mathit{Role}}_k$) are properly defined as follows.

Assumption 6.10. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts $\mathcal{B}$ using an abstraction function $\hat{\psi}$. Then, for all k:

$$\{\mathit{strand}(n) \mid n \in \mathcal{N}_{\mathcal{B}}\} \cap \mathit{Role}_k \nearrow_{\hat{\psi}} \widehat{\mathit{Role}}_k \cap \{\mathit{strand}(\hat{n}) \mid \hat{n} \in \mathcal{N}_{\hat{\mathcal{B}}}\}.$$

One consequence of the way that logical terms were defined is that we only ever consider regular nodes and regular strands, i.e. it is impossible to write a term that is interpreted as a penetrator node. This can be formalised as follows.

Lemma 6.11. Let t be a logical term, $\mathcal{B}$ be a low-level bundle, $\hat{\mathcal{B}}$ be a high-level bundle and Γ be a high or low-level term environment. Then, if $\Gamma \vdash t : \mathit{Node}$ then $[\![t]\!]^{\Gamma}_{\mathcal{B}}$ and $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ are regular and $[\![t]\!]^{\Gamma}_{\mathcal{B}} \in \mathcal{N}_{\mathit{payload}}$. Also, if $\Gamma \vdash t : \mathit{Strand}$ then $[\![t]\!]^{\Gamma}_{\mathcal{B}}$ and $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ are regular.

Proof (Sketch). This can be proven by a trivial structural induction over t. □

Using the above results we are now able to prove the main result of this section.

Lemma 6.12. Let t be a logical term, $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts $\mathcal{B}$ using a node map $\hat{\psi}$. Further, let Γ and $\hat{\Gamma}$ be logical term environments defined on all free variables in t such that $\Gamma \nearrow_{\hat{\psi}} \hat{\Gamma}$. Then, $[\![t]\!]^{\Gamma}_{\mathcal{B}} \nearrow_{\hat{\psi}} [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$.

Proof. Let t, $\mathcal{B}$, $\hat{\mathcal{B}}$, Γ, $\hat{\Gamma}$ and $\hat{\psi}$ be as per the lemma. We prove the result by structural induction on the logical term, t.

Case $\mathsf{height}(t)$ By the inductive hypothesis $[\![t]\!]^{\Gamma}_{\mathcal{B}} \nearrow_{\hat{\psi}} [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Hence, by definition of $\nearrow_{\hat{\psi}}$, $\mathit{app\_height}_{\mathcal{B}}([\![t]\!]^{\Gamma}_{\mathcal{B}}) = \mathit{height}_{\hat{\mathcal{B}}}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$, as required.

Case $\mathsf{role_k}$ This follows immediately from Assumption 6.10.

Case $\mathsf{node}(t, n)$ By applying the inductive hypothesis it follows that $[\![n]\!]^{\Gamma}_{\mathcal{B}} = [\![n]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ and that $[\![t]\!]^{\Gamma}_{\mathcal{B}} \nearrow_{\hat{\psi}} [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Thus, by definition of the $\nearrow_{\hat{\psi}}$ relation on strands, it follows that (assuming $[\![n]\!]^{\Gamma}_{\mathcal{B}} \leq \mathit{height}_{\hat{\mathcal{B}}}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$):

$$\begin{aligned}
[\![\mathsf{node}(t, n)]\!]^{\Gamma}_{\mathcal{B}} &= ([\![t]\!]^{\Gamma}_{\mathcal{B}}, \mathit{app_node}([\![t]\!]^{\Gamma}_{\mathcal{B}}, [\![n]\!]^{\Gamma}_{\mathcal{B}})) \\
&\nearrow_{\hat{\psi}} ([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}, [\![n]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}) \\
&= [\![\mathsf{node}(t, n)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}.
\end{aligned}$$

Case $\mathsf{strand}(t)$ By the inductive hypothesis $[\![t]\!]^{\Gamma}_{\mathcal{B}}\ \hat{\psi}\ [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Further by Lemma 6.11 both $[\![t]\!]^{\Gamma}_{\mathcal{B}}$ and $[\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ are regular and therefore Lemma 6.9 can be applied to deduce that $\mathsf{strand}([\![t]\!]^{\Gamma}_{\mathcal{B}}) \nearrow_{\hat{\psi}} \mathsf{strand}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}})$, as required.

Case $\mathsf{msg}(t)$ As $\Gamma \vdash t : Node$ it follows, by the inductive hypothesis, that $[\![t]\!]^{\Gamma}_{\mathcal{B}}\ \hat{\psi}\ [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Therefore, by definition of a regular node map (Definition 4.12), it must be the case that:

$$\hat{\alpha}_{\mathcal{B}}(\mathsf{msg}([\![t]\!]^{\Gamma}_{\mathcal{B}})) = \mathsf{msg}(\hat{\psi}([\![t]\!]^{\Gamma}_{\mathcal{B}}) = \mathsf{msg}([\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}).$$

Hence, by definition of $[\![\mathsf{msg}(t)]\!]^{\Gamma}_{\mathcal{B}}$ and $[\![\mathsf{msg}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ it follows that $[\![\mathsf{msg}(t)]\!]^{\Gamma}_{\mathcal{B}} = [\![\mathsf{msg}(t)]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$, as required.

The remaining cases follow trivially from the induction hypothesis. □

6.5 Logical Equivalence

In this section we prove that, if $\hat{\mathcal{B}}$ abstracts $\mathcal{B}$ and $\mathcal{B}$ does not satisfy a bundle predicate ϕ, then $\hat{\mathcal{B}}$ also does not satisfy ϕ. We start by proving several lemmas that cover the more complicated cases in the above proof.

6.5.1 Origination

Firstly, we prove that our definition of application-layer origination (cf. Definition 6.5) is correct, in that it matches the definition of origination in high-level bundles.

Lemma 6.13. Let $\mathcal{B}$ be a normal low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts it using a node map $\hat{\psi}$. Further, let $n \in \mathcal{N}_{\mathcal{B}} \cap \mathcal{N}_{payload}$ and $\hat{n} \in \mathcal{N}_{\hat{\mathcal{B}}}$ be regular nodes such that $n\ \hat{\psi}\ \hat{n}$. Then, for every atom $t \notin \mathcal{A}^{*}_{\mathcal{P}}$, t originates in the application layer at n in $\mathcal{B}$ iff t originates at $\hat{n}$ in $\hat{\mathcal{B}}$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$, t, n and $\hat{n}$ be as per the lemma. Firstly, suppose that t originates in the application layer at n in $\mathcal{B}$. As n is a regular node it follows that $t \sqsubseteq \mathit{appmsg}(n)$ and $t \not\sqsubseteq \mathit{appmsg}(n')$ for any $n' \in \mathcal{N}_{payload}$ such that $n' \Rightarrow^{+} n$. Hence, as $\mathit{appmsg}(n) = \mathit{appmsg}(\hat{n})$ (since $\hat{\phi}(n) = \hat{n}$) it follows that $t \sqsubseteq \mathit{appmsg}(\hat{n})$. Let $\hat{n}'$ be such that $\hat{n}' \Rightarrow^{+} \hat{n}$. By Lemma 6.9, $st \nearrow_{\hat{\psi}} \hat{st}$, and thus $\hat{\psi}^{-1}(\hat{n}')$ must lie on st. Therefore, by Definition 4.12, $\hat{\psi}^{-1}(\hat{n}') \Rightarrow^{+} n$ so $t \not\sqsubseteq \mathit{appmsg}(\hat{\psi}^{-1}(\hat{n}'))$ and $\mathit{appmsg}(\hat{\psi}^{-1}(\hat{n}')) = \mathit{appmsg}(\hat{n}')$ so $t \not\sqsubseteq \mathit{appmsg}(\hat{n}')$. Hence, it follows that t originates at $\hat{n}$ in $\hat{\mathcal{B}}$, as required.

Otherwise, if t originates at $\hat{n}$ in $\hat{\mathcal{B}}$ then a similar proof to the above can be used to show that t must originate in the application-layer at n. □

We now prove a simple extension of the above lemma that shows the high and low-level semantics of $\mathsf{UniquelyOriginates}(t, n)$ are compatible.

Corollary 6.14. Let $\mathcal{B}$ be a normal low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts it using a node map $\hat{\psi}$. Further, let $n \in \mathcal{N}_{\mathcal{B}}$ and $\hat{n} \in \mathcal{N}_{\hat{\mathcal{B}}}$ be such that $n \nearrow_{\hat{\psi}} \hat{n}$. Then, for every atom $t \notin \mathcal{A}^*_{\mathcal{P}}$, t uniquely originates in the application-layer at n in $\mathcal{B}$ iff t uniquely originates at $\hat{n}$ in $\hat{\mathcal{B}}$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$, n, $\hat{n}$ and t be as per the lemma. As $t \notin \mathcal{A}^*_{\mathcal{P}}$ it must be the case that t originates only on regular nodes (as atoms cannot originate on constructive penetrator strands) in both $\mathcal{B}$ and $\hat{\mathcal{B}}$. Further, by definition of application-layer origination, it follows that if t originates in the application-layer at a regular node n, then $n \in \mathcal{N}_{payload}$. The result therefore follows immediately from Lemma 6.13. □

6.5.2 Confidentiality

We now prove that the application-layer values that are confidential are identical in the low and high-level semantics. We firstly prove that if a value is confidential at the low-level, then it is confidential at the high-level. This result follows immediately from the observation that the penetrator knows at least as many values at the low-level as the high-level, so can still perform the requisite decryptions.

Lemma 6.15. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a bundle that abstracts it using an abstraction function $\hat{\psi}$. If t is confidential in $\mathcal{B}$ then t is confidential in $\hat{\mathcal{B}}$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$ and t be as per the lemma. Suppose, for a contradiction, that t is confidential in $\mathcal{B}$ but not in $\hat{\mathcal{B}}$. Thus, it follows that there must exist an equivalent high-level bundle $\hat{\mathcal{B}}'$ and a node $\hat{n} \in \hat{\mathcal{B}}'$ such that $msg(\hat{n}) = +(?,\ ?,\ \_,\ t,\ \bot)$. Further, a low-level bundle $\mathcal{B}'$ can be constructed such that $\hat{\mathcal{B}}'$ abstracts $\mathcal{B}'$ by altering $\mathcal{B}$ in the same way that $\hat{\mathcal{B}}$ was altered to obtain $\hat{\mathcal{B}}'$, mapping each high-level strand to a low-level subpath. Further, note that $\mathcal{B}$ and $\mathcal{B}'$ must be equivalent, as the high-level bundles that abstract them contain the same regular nodes. However, this contradicts the fact that t was confidential in $\mathcal{B}$. Hence, t is confidential in $\hat{\mathcal{B}}$, as required. □

We now consider how to prove the opposite direction; i.e. whenever a value is confidential in the high-level bundle, then it is confidential in the low-level bundle. The correctness of this lemma can be informally justified by recalling from the logical semantics that we restrict our attention to the confidentiality of application-layer values (i.e. terms from $\mathcal{A}_{app}$). Therefore, we would expect that the penetrator gains no more relevant information from the low-level bundle and hence, there should be no way for him to compromise the confidentiality of the value in the low-level bundle. The following proof is similar in structure to Lemma 5.23. As before, we only consider the case of non-concatenations but lift this restriction in Lemma 6.17

Lemma 6.16. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a bundle that abstracts it using an abstraction function $\hat{\psi}$. If t is not a concatenation, is confidential in $\hat{\mathcal{B}}$ and $t \in \mathcal{A}_{app}$, then t is confidential in $\mathcal{B}$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$ and t be as per the lemma. Note that since t is confidential in $\hat{\mathcal{B}}$, $t \notin \mathcal{A}_{\mathcal{P}}$. Suppose, for a contradiction, that t is not confidential in $\mathcal{B}$. Thus, it follows that there must exist an equivalent low-level bundle $\mathcal{B}'$ and a node n such that $msg(n) = +t$. Without loss of generality, assume that $\mathcal{B}'$ is normal (but still

includes the penetrator node n) and let n_{orig} be the node that originates the copy of t that ends up at n. Further, let p_{orig} be the path between n_{orig} and n, and, for each j, let $tpath_j$ be the extraction path used to extract the copy of t that is used at n from $p_{orig}(j)$. Note that, by definition, p_{orig} does not traverse any key edges or KG strands (although n_{orig} could be the positive node on a KG strand). Let i be the index, if it exists, of the last regular node on p_{orig} such that t occurs in the application layer; i.e. $p(i) \in \mathcal{N}_\perp$ or $p_{orig}(i) \in \mathcal{N}_{trpt}$ and $\mathsf{expath}^{\mathsf{c}}(chan(p(i))) \leq tpath_i$. There are three cases to consider:

Case 1 No such node $p_{orig}(i)$ exists. Consider the node n_{orig} and suppose, for a contradiction, that it is regular. Then it must be the case that either $n_{orig} \in \mathcal{N}_{non\text{-}payload}$, or $n_{orig} \in \mathcal{N}_{trpt}$ but $\mathsf{expath}^{\mathsf{c}}(chan(n_{orig})) \not\leq tpath_1$. However, as $t \in \mathcal{A}_{app} \setminus \mathcal{A}_{\mathcal{P}}$ and is not a concatenation, this contradicts Assumption 5.9 (7). Hence n_{orig} must be a penetrator node, and further, as t originates on n_{orig}, n_{orig} must be the positive node of either a M, E or KG strand, since t is not a concatenation.

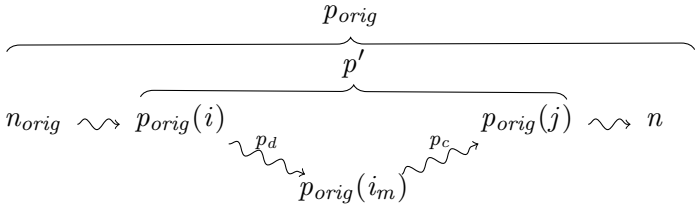
Suppose n_{orig} lies on a E strand. Let $t = \{\!|m|\!\}_k$ and note that at least one of m and k must be confidential in $\hat{\mathcal{B}}$. Thus, the same argument as the previous paragraph can be applied to derive an equivalent contradiction. Thus, n_{orig} cannot lie on a E strand. Further, an identical argument can be applied to show that n_{orig} cannot lie on a KG strand.

Otherwise, suppose n_{orig} lies on a M strand. Since t is confidential in $\hat{\mathcal{B}}$ it follows that there must exist $es < tpath_1$, m and k such that $\{\!|m|\!\}_k \sqsubseteq_{es} msg(n_{orig})$, $k^{-1} \notin \mathcal{A}_{\mathcal{P}}$ and k^{-1} is confidential in $\hat{\mathcal{B}}$. Let es be the shortest such extraction path. As $t \in \mathcal{A}_{app}$, Assumption 5.9 (6) can be applied to deduce that every encryption that encloses t in $msg(n_{orig})$ is in $\mathcal{E}_{app}$. Therefore, since $es < tpath_1$ we have $\{\!|m|\!\}_k \in \mathcal{E}_{app}$. We consider how the penetrator removes the encryption $\{\!|m|\!\}_k$ in $\mathcal{B}'$.

Firstly, suppose that the penetrator uses a D strand to decrypt $\{\!|m|\!\}_k$. As $\{\!|m|\!\}_k \in \mathcal{E}_{app}$, k^{-1} is the inverse key of an application-layer encryption and, by definition of $\mathcal{A}_{app}$, $k^{-1} \in \mathcal{A}_{app}$. Therefore, as k^{-1} is confidential in $\hat{\mathcal{B}}$ this lemma can be inductively applied[1] to prove that k^{-1} is confidential in $\mathcal{B}$, and thus $\mathcal{B}'$. This contradicts the existence of the negative key node on the D strand.

The only other possibility is that there exists a regular node at an index i such that $\{\!|m|\!\}_k \sqsubseteq msg(p_{orig}(i))$, and an index $i' > i$ such that $\{\!|m|\!\}_k$ has been decrypted (i.e. the penetrator uses the regular agent to do the decryption). However, in such a case it immediately follows, since t does not appear in the application-layer on p_{orig}, that $p_{orig}(i) \in \mathcal{N}_{non\text{-}payload}$ or $p_{orig}(i) \in \mathcal{N}_{trpt}$ but $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \not\leq tpath_i$. Therefore, at $p_{orig}(i)$, $\{\!|m|\!\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg}$ or $\{\!|m|\!\}_k \in \mathcal{E}_{trpt}$. However, this implies that either $\{\!|m|\!\}_k \in \mathcal{E}_{trpt} \cap \mathcal{E}_{app}$, contradicting Assumption 5.9 (2), or $\{\!|m|\!\}_k \in \mathcal{E}_{trpt\text{-}non\text{-}msg} \cap \mathcal{E}_{app}$. In the latter

[1] If there is an encryption cycle, e.g. if $\{\!|m|\!\}_k = \{\!|k^{-1}|\!\}_k$, then this is not well founded. It would be possible to rewrite this proof to take this into account by considering a set of terms that are required to be confidential, rather than an individual term. For ease of exposition, we do not give this version.



(a) The penetrator paths for Case (2).

Figure 6.3: An illustration of Lemma 6.16.

case, by Assumption 5.9 (4), $k^{-1} \in \mathcal{A}_{\mathcal{P}}$, contradicting the earlier assumption that $k^{-1} \notin \mathcal{A}_{\mathcal{P}}$. Hence, the penetrator cannot use a regular node to do the decryption.

As we derive a contradiction in all cases, this case cannot occur.

Case 2 Such a node $p_{orig}(i)$ exists and there exists an index $j > i$ such that $p_{orig}(j)$ is regular. Thus, let j be the first such regular node and note that t does not occur in the application layer at $p_{orig}(j)$. Hence, either $p_{orig}(j) \in \mathcal{N}_{non\text{-}payload}$ or $p_{orig}(j) \in \mathcal{N}_{trpt}$ but $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(j))) \not\leq t_j$. Consider the normal penetrator path p' between $p_{orig}(i)$ and $p_{orig}(j)$. Further, let $p_d \hat{} p_c = p'$ where p_d is destructive and p_c is constructive and let $p_{orig}(i_m)$ be the middle node (i.e. $p_c(1)$), as illustrated in Figure 6.3. We aim to apply Lemma 5.22, with $n_1 = p_{orig}(i)$, $n_2 = p_{orig}(j)$, $n_m = p_{orig}(i_m)$ and $es_t = t_{i_m}$. By definition of i and $tpath_i$ it follows that, if $p_{orig}(i) \notin \mathcal{N}_{\perp}$, $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \leq tpath_i = \mathsf{expath}(p_d) \hat{} t_{i_m}$, as required. Further, by definition of j it follows that $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(j))) \not\leq t_j = \mathsf{expath}^{\sim}(p_c) \hat{} t_{i_m}$, as required. Hence, all assumptions of Lemma 5.22 are satisfied and therefore any encryption that encloses t within $msg(p_{orig}(i_m))$ is in $\mathcal{E}_{trpt\text{-}non\text{-}msg} \cap \mathcal{E}_{app}$.

Therefore, by Assumption 5.9 (4), the inverse of every key enclosing t is in $\mathcal{A}_{\mathcal{P}}$. However, this clearly contradicts the confidentiality of t in $\hat{\mathcal{B}}$. In particular, p_d could be mapped to a RV or LN strand followed by D strands with keys from M strands, in order to to extract t from the application-layer payload.

Case 3 Such a node $p_{orig}(i)$ exists but no further node on p_{orig} is regular. In this case, as $\mathcal{B}$ is normal there exists a normal penetrator path p' from $p_{orig}(i)$ to n. Note that p' must be destructive as if it were to finish with a constructive path then t would originate at n since $msg(n) = t$. Thus p' is destructive and note that, by definition of $tpath_i$, $tpath_i = \mathsf{expath}(p')$. Hence, if $p_{orig}(i) \in \mathcal{N}_{trpt}$ then $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) \leq tpath_i = \mathsf{expath}(p')$. Therefore, there exists a path $p'' \leq p'$ such that $\mathsf{expath}^{\mathsf{c}}(chan(p_{orig}(i))) = \mathsf{expath}(p'')$ and thus, p'' can be mapped to either a RV or LN strand in $\hat{\mathcal{B}}$. Thus, there exists a bundle in $\hat{\mathcal{B}}'$ that is equivalent to $\hat{\mathcal{B}}$, but such that there can exist a node n' such that $msg(n') = (?, ?, \_, appmsg(p_{orig}(i)), \perp)$ (noting that it trivially follows if $p_{orig}(i) \in \mathcal{N}_{\perp}$).

Therefore, as t is confidential in $\hat{\mathcal{B}}$, it follows that t must be enclosed by an encryption $\{\!|m|\!\}_k$ in $appmsg(p_{orig}(i))$ such that the inverse key k^{-1} is confidential in $\hat{\mathcal{B}}$ (otherwise it would be trivial to extract t). Hence, $k^{-1} \notin \mathcal{A}_{\mathcal{P}}$

and further, since $\{\!|m|\!\}_k \sqsubseteq appmsg(tpath_i)$, $\{\!|m|\!\}_k \in \mathcal{E}_{app}$. Hence, k^{-1} is the inverse key of an application-layer encryption and therefore $k^{-1} \in \mathcal{A}_{app}$. Thus, the arguments of this lemma can be applied inductively to k^{-1} to deduce that k^{-1} must be confidential in $\mathcal{B}$ (and hence $\mathcal{B}'$), contradicting this case.

Hence, in each of the above cases we derive a contradiction and therefore it follows that t is confidential in $\mathcal{B}$, as required. □

We now lift the restriction on concatenations.

Lemma 6.17. Let $\mathcal{B}$ be a low-level bundle and $\hat{\mathcal{B}}$ be a bundle that abstracts it using an abstraction function $\hat{\psi}$. If t is confidential in $\hat{\mathcal{B}}$ and $t \in \mathcal{A}_{app}$, then t is confidential in $\mathcal{B}$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$ and t be as per the lemma. If t is not a concatenation the result follows immediately from Lemma 6.16. Otherwise, let $t = t_1 \hat{} t_2$ and observe that t is confidential in $\hat{\mathcal{B}}$ only if at least one of its components (i.e. t_1 or t_2) is. Thus, suppose that t_j, $j \in \{1, 2\}$, is confidential in $\hat{\mathcal{B}}$ and hence, by definition of $\mathcal{A}_{app}$, as $t \in \mathcal{A}_{app}$, $t_j \in \mathcal{A}_{app}$. Thus, this lemma may be applied inductively to prove that t_j is confidential in $\mathcal{B}$. Thus, it follows that $t_1 \hat{} t_2$ is confidential in $\mathcal{B}$, as required. □

6.5.3 Causal Precedence

We now consider under what circumstances the $\preceq$ relation between application-layer nodes is preserved between low and high-level bundles. Consider two nodes n_1 and n_2 in a low-level bundle such that $n_1 \preceq n_2$. If the path between n_1 and n_2 transfers only transport-layer values then the path does not need to be abstracted. Hence, it is not necessarily the case that the same relation holds amongst the abstracted nodes and, therefore, we can only prove that if $\hat{n}_1 \preceq \hat{n}_2$ in the high-level bundle, then $n_1 \preceq n_2$. This result follows from the observation that any path in the high-level bundle must be reflected (by the definition of a node map) in a (possibly extended) path in the low-level bundle.

Lemma 6.18. Let $\mathcal{B}$ be a normal low-level bundle and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts it using a node map $\hat{\psi}$. Then, if $n_1, n_2 \in \mathcal{N}_{payload}$, $\hat{\psi}(n_1) = \hat{n}_1$, $\hat{\psi}(n_2) = \hat{n}_2$ and $\hat{n}_1 \preceq_{\hat{\mathcal{B}}} \hat{n}_2$, then $n_1 \preceq_{\mathcal{B}} n_2$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$, $\hat{\psi}$, n_1, n_2, $\hat{n}_1$ and $\hat{n}_2$ be as per the lemma. Recall that $\preceq$ is the reflexive transitive closure of $\rightarrow \cup \Rightarrow$. Thus, there exists a sequence of nodes $\hat{n}_1 = \hat{n}'_1, \ldots, \hat{n}'_m = \hat{n}_2$ such that for each $i \in \{1..m-1\}$, $\hat{n}_i \ \sigma_{\hat{\mathcal{B}}} \ \hat{n}_{i+1}$ where $\sigma_{\hat{\mathcal{B}}} \mathrel{\widehat{=}} \rightarrow_{\hat{\mathcal{B}}} \cup \Rightarrow^+_{\hat{\mathcal{B}}}$. If there exists i such that $\hat{n}_i \rightarrow_{\hat{\mathcal{B}}} \hat{n}_{i+1}$ then, by Definition 4.23, $\hat{\psi}^{-1}(\hat{n}_i) \rightarrow_{\mathcal{B}} \hat{\psi}^{-1}(\hat{n}_{i+1})$.

Alternatively, suppose $\hat{n}_i \Rightarrow_{\hat{\mathcal{B}}} \hat{n}_{i+1}$. We prove the existence of a series of nodes $n'_1, \ldots, n'_j$ such that $\hat{\psi}^{-1}(\hat{n}_i) = n'_1$, $\hat{\psi}^{-1}(\hat{n}_{i+1}) = n'_j$ and for each $k \in \{1..j-1\}$, $n_k \ \sigma_{\mathcal{B}} \ n_{k+1}$ where $\sigma_{\mathcal{B}} \mathrel{\widehat{=}} \rightarrow_{\mathcal{B}} \cup \Rightarrow_{\mathcal{B}}$. Thus, define n'_1 and n'_j as above. There are two cases to consider. If $\hat{n}_i$ and $\hat{n}_{i+1}$ are regular, then by Definition 4.12, $strand(n'_1) = strand(n'_j)$ and $n'_1 \Rightarrow^+_{\mathcal{B}} n'_j$.

Otherwise, $\hat{n}_i$ and $\hat{n}_{i+1}$ are penetrator nodes. By Definition 4.23 n'_1 and n'_j are either related by an application-layer penetrator node map, or a transport-layer

penetrator node map. In the former case, Definition 4.13 requires the $\Rightarrow$ relation to be preserved and thus $n'_1 \Rightarrow n'_j$. Otherwise, by Definition 4.21, there must exist a penetrator subpath (i.e. a Receive, Learn, Fake subpath etc.) between n'_1 and n'_j. Hence, we can find a sequence of nodes from the subpath such that for each $k \in \{1..j-1\}$, $n'_k \; \sigma_{\mathcal{B}} \; n'_{k+1}$, as required. □

6.5.4 The Main Result

Using the results from the previous section we are now able to prove our main result, namely that bundle satisfaction is preserved by abstraction. In light of Lemma 6.18 we are only able to consider formulas where $\preceq$ only occurs positively (i.e. within the scope of an even number of negations only). However, this is not a restriction in practice; all the predicates that we are likely to want to express will take the form of an implication where $\preceq$ will occur only on the right hand side (i.e. as a conclusion of the predicate), and thus will occur positively.

Proposition 6.19. Let ϕ be a closed high-level bundle predicate in which $\preceq$ occurs only positively. Further, let $\mathcal{B}$ be a normal low-level bundle in a strand space that satisfies layer-disjoint encryption and $\hat{\mathcal{B}}$ be a high-level bundle that abstracts $\mathcal{B}$. If $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \phi$ then $(\mathcal{B}, \Gamma) \models \phi$.

Proof. Let $\mathcal{B}$, $\hat{\mathcal{B}}$ and ϕ be as per the lemma. Without loss of generality we assume that ϕ is in negation normal form. We prove the claim by structural induction on the formula ϕ using the inductive hypothesis that $\Gamma \nearrow_{\hat{\psi}} \hat{\Gamma}$, and if $(\mathcal{B}', \hat{\Gamma}) \hat{\models} \phi$ then $(\mathcal{B}, \Gamma) \models \phi$. Most cases follow immediately from Lemma 6.12 or the inductive hypothesis, so are omitted.

Case $t_1 \preceq t_2$**:** By Lemma 6.12 it follows that $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} \; \hat{\psi} \; [\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ and $[\![t_2]\!]^{\Gamma}_{\mathcal{B}} \; \hat{\psi} \; [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Further, by Lemma 6.11, $[\![t_1]\!]^{\Gamma}_{\mathcal{B}}$, $[\![t_2]\!]^{\Gamma}_{\mathcal{B}}$, $[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ and $[\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ are all regular and, further, $[\![t_1]\!]^{\Gamma}_{\mathcal{B}}, [\![t_2]\!]^{\Gamma}_{\mathcal{B}} \in \mathcal{N}_{payload}$. Thus, suppose $[\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}} \preceq [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Therefore, Lemma 6.18 can be applied to deduce that $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} \preceq [\![t_2]\!]^{\Gamma}_{\mathcal{B}}$, as required.

Cases $\mathsf{Confidential}(t)$**,** $\neg\mathsf{Confidential}(t)$**:** By Lemma 6.12 it follows that $[\![t]\!]^{\Gamma}_{\mathcal{B}} = [\![t]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Let $t' = [\![t]\!]^{\Gamma}_{\mathcal{B}}$. Suppose $(\mathcal{B}, \Gamma) \models \mathsf{Confidential}(t)$. Then, t' must be confidential in $\mathcal{B}$ and t' must be the inverse key of an application-layer encryption, or $t' \in \mathcal{A}_{app}$. Hence, by Lemma 6.17, t' is confidential in $\hat{\mathcal{B}}$. Hence, $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \mathsf{Confidential}(t)$.

Conversely, suppose $(\hat{\mathcal{B}}, \hat{\Gamma}) \hat{\models} \mathsf{Confidential}(t)$. It thus follows that t is confidential in $\hat{\mathcal{B}}$ and t must be the inverse key of an application-layer encryption, or $t \in \mathcal{A}_{app}$. Thus, Lemma 6.15 can be applied to show that t is confidential in $\mathcal{B}$ and thus that $(\mathcal{B}, \Gamma) \models \mathsf{Confidential}(t)$, as required.

Cases $\mathsf{UniquelyOriginates}(t_1, t_2)$**,** $\neg\mathsf{UniquelyOriginates}(t_1, t_2)$**:** By Lemma 6.12 it immediately follows that $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} = [\![t_1]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$ and $[\![t_2]\!]^{\Gamma}_{\mathcal{B}} \nearrow_{\hat{\psi}} [\![t_2]\!]^{\hat{\Gamma}}_{\hat{\mathcal{B}}}$. Hence, let t be equal to $[\![t_1]\!]^{\Gamma}_{\mathcal{B}}$. If $t \notin \mathcal{T} \setminus \mathcal{A}^*_{\mathcal{P}}$ then the result immediately follows. Otherwise, $t \in \mathcal{T} \setminus \mathcal{A}^*_{\mathcal{P}}$ and the result follows immediately from Corollary 6.14. □

6.6 Proofs by Abstraction

We now consider how to combine the results of Proposition 5.34 and Proposition 6.19 to show that the high-level strand spaces model is sound. In particular, we show that whenever a low-level bundle $\mathcal{B}$ does not satisfy a bundle predicate then there is a high-level bundle $\hat{\mathcal{B}'}$ that also does not satisfy the bundle predicate. We prove the above by using Proposition 5.34 to obtain a bundle $\mathcal{B}'$ that is abstractable and such that $\mathcal{B} \trianglerighteq \mathcal{B}'$. Then, we can apply Proposition 6.19 to deduce that if $\mathcal{B}'$ does not satisfy a particular bundle predicate, then there exists a high-level bundle $\hat{\mathcal{B}'}$ that abstracts $\mathcal{B}$ and such that $\hat{\mathcal{B}'}$ also does not. Hence, it suffices to show that whenever $\mathcal{B} \trianglerighteq \mathcal{B}'$ and $\mathcal{B}$ does not satisfy a property, $\mathcal{B}'$ also does not satisfy this property. We begin by proving this result.

Lemma 6.20. Let $\mathcal{B}$ and $\mathcal{B}'$ be equivalent low-level bundles, t be a logical term and Γ a low-level term environment defined on all free variables of t. Then, $[\![t]\!]^{\Gamma}_{\mathcal{B}} = [\![t]\!]^{\Gamma}_{\mathcal{B}'}$.

Proof. This follows from a trivial structural induction on t. □

Lemma 6.21. Let ϕ be a closed high-level bundle predicate in which $\preceq$ occurs only positively. Let $\mathcal{B}$, $\mathcal{B}'$ be two low-level bundles such that $\mathcal{B} \trianglerighteq \mathcal{B}'$ and $(\mathcal{B}', \emptyset) \vDash \phi$. Then $(\mathcal{B}, \emptyset) \vDash \phi$.

Proof. Let ϕ, $\mathcal{B}$ and $\mathcal{B}'$ be as per the lemma. Let ϕ' be the negation normal form of ϕ. Since, for any low-level bundle $\mathcal{B}$, $(\mathcal{B}, \Gamma) \vDash \phi$ iff $(\mathcal{B}, \Gamma) \vDash \phi'$, it follows it is sufficient to consider ϕ'. We prove the claim by structural induction on the formula ϕ' using the inductive hypothesis that if $(\mathcal{B}', \Gamma) \vDash \phi'$ then $(\mathcal{B}, \Gamma) \vDash \phi'$. We omit cases that follow immediately from Lemma 6.20 or the inductive hypothesis.

Case $t_1 \preceq t_2$**:** By Lemma 6.20, $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} = [\![t_1]\!]^{\Gamma}_{\mathcal{B}'}$ and $[\![t_2]\!]^{\Gamma}_{\mathcal{B}} = [\![t_2]\!]^{\Gamma}_{\mathcal{B}'}$. Further, by Lemma 6.11, $[\![t_1]\!]^{\Gamma}_{\mathcal{B}}, [\![t_2]\!]^{\Gamma}_{\mathcal{B}} \in \mathcal{N}_{payload}$. Thus, assume $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} \preceq_{\mathcal{B}'} [\![t_2]\!]^{\Gamma}_{\mathcal{B}}$ and note that, as $\mathcal{B} \trianglerighteq \mathcal{B}'$, $\preceq_{\mathcal{B}'} \subseteq \preceq_{\mathcal{B}}$. Hence, $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} \preceq_{\mathcal{B}} [\![t_2]\!]^{\Gamma}_{\mathcal{B}}$, as required.

Cases $\mathsf{Confidential}(t)$**,** $\neg\mathsf{Confidential}(t)$**:** These follow immediately from Lemma 6.20 and the observation that confidentiality is closed under bundle equivalence.

Cases $\mathsf{UniquelyOriginates}(t_1, t_2)$**,** $\neg\mathsf{UniquelyOriginates}(t_1, t_2)$**:** By Lemma 6.20 $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} = [\![t_1]\!]^{\Gamma}_{\mathcal{B}'}$ and $[\![t_2]\!]^{\Gamma}_{\mathcal{B}} = [\![t_2]\!]^{\Gamma}_{\mathcal{B}}$. Suppose $(\mathcal{B}, \Gamma) \vDash \mathsf{UniquelyOriginates}(t_1, t_2)$. Then, by definition of $\vDash$ it follows that $t \in \mathcal{T} \setminus \mathcal{A}^*_{\mathcal{P}}$. Thus, t can only originate on regular nodes (as t is an atom it cannot originate on a constructive penetrator strand). Therefore, since $\mathcal{B}$ and $\mathcal{B}'$ are equivalent, it follows that the set of regular nodes is identical between the two strands and thus application-layer origination on regular nodes is unaffected. The other direction can be proven in an identical fashion. □

Lastly, we can prove our main result that shows that a proof of protocol correctness in the high-level model is sound, in that it will consider all attacks that could have occurred in the low-level model.

Proposition 6.22. Let ϕ be a closed high-level bundle predicate in which $\preceq$ occurs only positively. Further, let Σ be a low-level strand space satisfying layer-disjoint

encryption and $\hat{\Sigma_E}$ be a high-level strand space abstracting Σ_E such that Σ and Σ_E are related according to Definition 5.15. Then, if every high-level bundle of $\hat{\Sigma_E}$ satisfies ϕ, then every low-level bundle of Σ satisfies ϕ.

Proof. Let ϕ, Σ and $\hat{\Sigma_E}$ be as per the lemma. Suppose, for a contradiction, that there exists a low-level bundle $\mathcal{B}$ of Σ such that $(\mathcal{B}, \emptyset) \not\models \phi$, but that every bundle of $\hat{\Sigma_E}$ satisfies ϕ. Then, by Proposition 5.34 there exists a normal abstractable bundle $\mathcal{B}'$ of Σ_E such that $\mathcal{B} \trianglerighteq \mathcal{B}'$. Further, by Lemma 6.21 it follows that $(\mathcal{B}', \emptyset) \not\models \phi$. Let $\hat{\mathcal{B}'}$ abstract $\mathcal{B}'$ and note that, by Proposition 6.19, $(\hat{\mathcal{B}'}, \emptyset) \hat{\not\models} \phi$, contradicting the assumption. □

Note that the above proposition requires the high-level strand space to abstract Σ_E, rather than Σ. Therefore, when doing the proof of protocol correctness in the high-level model the penetrator's initial knowledge must include $\mathcal{T}_X$. In practice, this should not cause any difficulty as $\mathcal{T}_X$ contains only items that are not in $\mathcal{A}_{app}$, meaning that nothing of consequence to the application-layer security of the protocol is added to $\mathcal{A}_{\mathcal{P}}$. This means that when constructing proofs of protocol correctness, any assumptions on the contents of $\mathcal{A}_{\mathcal{P}}$ should be negative, in that they should specify what is not in $\mathcal{A}_{\mathcal{P}}$, rather than what is.

6.6.1 High-Level Proofs

When proving the correctness of application-layer protocols in the high-level strand spaces model we will frequently only consider normal bundles, for ease. Further, recall that the high-level bundles produced as abstractions of low-level bundles in Proposition 4.31 contain HJRN and TX strands, which we do not normally consider in high-level correctness proofs. We therefore need to prove that this is sound. The proof of this is identical to the proof of Lemma 6.21 for low-level bundles, and thus we simply state the lemmas and definitions without proof or much explanation. We start by defining a high-level version of $\trianglerighteq$, $\hat{\trianglerighteq}$, and then proving that, given an arbitrary high-level bundle, we can find a normal high-level bundle that relates to the original bundle using $\hat{\trianglerighteq}$ and, further, contains no HJRN or TX strands.

Definition 6.23. A high-level bundle $\hat{\mathcal{B}}$ *can be reduced to* $\hat{\mathcal{B}'}$, denoted $\hat{\mathcal{B}} \hat{\trianglerighteq} \hat{\mathcal{B}'}$, iff $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}'}$ are equivalent and $\preceq_{\hat{\mathcal{B}'}}$ relates no more regular nodes than $\preceq_{\hat{\mathcal{B}}}$, i.e:

$$\left(\mathcal{N}^{reg}_{\hat{\mathcal{B}}} \times \mathcal{N}^{reg}_{\hat{\mathcal{B}}}\right) \cap \preceq_{\hat{\mathcal{B}}} \supseteq \left(\mathcal{N}^{reg}_{\hat{\mathcal{B}'}} \times \mathcal{N}^{reg}_{\hat{\mathcal{B}'}}\right) \cap \preceq_{\hat{\mathcal{B}'}}.$$

Lemma 6.24. Let $\hat{\mathcal{B}}$ be a high-level bundle. Then there exists a normal bundle $\hat{\mathcal{B}'}$ such that $\hat{\mathcal{B}} \hat{\trianglerighteq} \hat{\mathcal{B}'}$.

Proof. This can be proven by adapting Lemma 3.13 in an identical fashion to Lemma 5.6 by noting that removing redundancies does not increase $\preceq$. □

Lemma 6.25. Let $\hat{\mathcal{B}}$ be a normal high-level bundle. Then there exists a normal high-level bundle $\hat{\mathcal{B}'}$ such that $\hat{\mathcal{B}} \hat{\trianglerighteq} \hat{\mathcal{B}'}$ and such that $\hat{\mathcal{B}'}$ contains no TX or HJRN strands.

Proof. The result immediately follows from the fact that the transformation used in Lemma 4.16 to remove the TX and HJRN strands does not increase $\preceq$. □

Lemma 6.26. Let ϕ be a closed high-level bundle predicate in which $\preceq$ occurs only positively and let $\hat{\mathcal{B}}$, $\hat{\mathcal{B}'}$ be two high-level bundles such that $\hat{\mathcal{B}} \hat{\trianglerighteq} \hat{\mathcal{B}'}$ and $(\hat{\mathcal{B}'}, \emptyset) \hat{\models} \phi$. Then $(\hat{\mathcal{B}}, \emptyset) \hat{\models} \phi$.

Proof. This can be proven in identical fashion to Lemma 6.21. □

Proposition 6.27. Let $\hat{\Sigma}$ be a high-level strand space and ϕ be a closed high-level bundle predicate in which $\preceq$ occurs only positively. Then, if every normal bundle of $\hat{\Sigma}$ that contains no TX or HJRN strands satisfies ϕ, then every bundle of $\hat{\Sigma}$ satisfies ϕ.

Proof. Let $\hat{\Sigma}$ and ϕ be as per the lemma. Suppose every normal bundle of $\hat{\Sigma}$ that contains no TX or HJRN strands satisfies ϕ. Let $\hat{\mathcal{B}}$ be an arbitrary bundle of $\hat{\Sigma}$. By Lemma 6.24 it follows that there exists a normal bundle $\hat{\mathcal{B}'}$ such that $\hat{\mathcal{B}} \hat{\trianglerighteq} \hat{\mathcal{B}'}$. Further, by Lemma 6.25 it follows that there exists a normal bundle $\hat{\mathcal{B}''}$ that contains no TX or HJRN strands and such that $\hat{\mathcal{B}'} \hat{\trianglerighteq} \hat{\mathcal{B}''}$. Thus, as $\hat{\trianglerighteq}$ is transitive and since $(\hat{\mathcal{B}'}, \emptyset) \hat{\models} \phi$, Lemma 6.26 can be applied to deduce that $(\hat{\mathcal{B}}, \emptyset) \hat{\models} \phi$. □

Henceforth, in all high-level strand spaces proofs we can, without loss of generality, consider only normal bundles.

6.7 Summary

In this chapter we have proven the main soundness result of the thesis. In particular, we proved in Proposition 6.22 that, if a protocol has been proven to satisfy a given property in the high-level strand spaces model, then it also satisfies the same property in any corresponding low-level strand spaces model, assuming the transport protocols satisfy the relevant channel types. We also showed, in Proposition 6.27, that if a correctness property is satisfied by all high-level normal bundles then that it is satisfied by all high-level bundles. This is particularly important as it is often necessary to make such an assumption in order to easily prove application-layer protocols correct.

In order to prove these results, we began, in Section 6.1, by formally defining a logic of correctness properties that can express a number of useful correctness properties. Then, in Section 6.2 and Section 6.3 we formally defined what it meant for high and low-level bundles to satisfy the correctness properties. In Section 6.4 we then proved that, given a high-level bundle that abstracts a low-level bundle, the value of any logical term in the low-level bundle is related, according to Definition 6.8, to its value in the high-level bundle. This was then used in Section 6.5 and Section 6.6 to prove that whenever a low-level bundle satisfies a correctness property, then the corresponding high-level bundle also satisfies the property.

Related Work Many other logics for expressing protocol correctness properties have been proposed in the past. In general, most logics are, like the one proposed in this chapter, tightly coupled to the theory in which the properties are being considered, making general comparisons less useful. As an example, Casper [Low97] has a simple past-time temporal logic that can be used to express that certain messages received by certain agents must agree. In [ACC07] the authors specify correctness

properties in LTL, where the individual predicates express that an agent knows a message, or that a particular agent is in a particular role, amongst others. The former is a generalisation of $\mathsf{Confidential}(t)$ which we can use to express what the penetrator knows, whilst the latter can be expressed using role_k. Many of the model checking tools, including Scyther [Cre08a] ProVerif [Bla01] and Casper, have simple assertion-based logics in which correctness properties can be expressed. In particular, all of the above tools supports simple assertions that terms are secret, or that two agents have successfully completed a run of the protocol together.

We discuss related work more generally at the end of Chapter 7, in Section 7.4.

Chapter 7

TLS

As TLS [DR08] is the most widespread secure transport protocol, we need to ensure that our model allows application-layer protocols that use TLS to be analysed. TLS is really two separate protocols: *bilateral* TLS, where the server is authenticated to the client and the client is authenticated to the server, and *unilateral* TLS, where only the former occurs. Unfortunately, as we illustrate in Section 7.1, TLS does not satisfy Assumption 5.9 as it is possible for some encryptions to be shared between the two forms of TLS in a (harmless) way that it is not allowed by Assumption 5.9. In this chapter we describe how any low-level bundle that uses TLS can be transformed into a bundle of a strand space that does satisfy Assumption 5.9 and has precisely the same application-layer behaviour.

We start in Section 7.1 by describing what the problem is. We then formalise the transformation of low-level bundles to the new strand space by defining a new notion of low-level bundle equivalence that is true whenever the bundles have the same application-layer behaviour. In Section 7.2 we then prove that any low-level bundle that uses TLS and does not satisfy a bundle predicate ϕ (i.e. as per Definition 6.2) can be transformed to a bundle that also does not satisfy ϕ, and is in the strand space that satisfies Assumption 5.9. In this section we follow the presentation of TLS given in Section 3 of [KL11].

7.1 The Transformation

Unilateral and bilateral TLS both have two stages. The first phase, called the *handshake* phase, involves the two participants exchanging data in order to derive a master key; this is then used in the second phase to protect the messages sent in the session. In the second phase, the identities of the sender and recipient of the message are not included in the message packaging. Thus, if the penetrator can somehow cause two sessions to use the same encryption keys then he will be able to pass messages between the two sessions without the other participants detecting it. If the two participants are regular agents then this is not possible [KL11]. However, if the penetrator is a legitimate participant in two different TLS sessions, one as the client and one as the server (i.e. both regular agents are aware they are communicating with the penetrator), he is able to manipulate the messages to cause this to occur.

The encryption keys used in the second phase are generated from a nonce n_c sent by the client, a nonce n_s sent by the server and the *pre-master secret pms*, which is

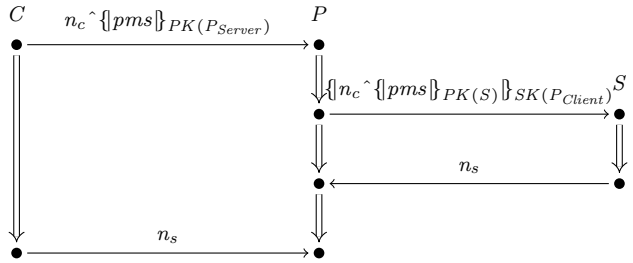


Figure 7.1: A example of a session where the penetrator causes a unilateral TLS session between C and P_{Server} to use the same encryption key as the bilateral TLS session between P_{Client} and S. The above uses a simplified version of TLS, rather than the full version.

generated by the client (and sent securely so it can only be received by the server). Thus, the penetrator could act as the server in one session, taking the client nonce and pre-master secret and play these into a second session, in which he acts as the client. Further, he can pass back the server nonce from the second session to the client in the first session, as illustrated in Figure 7.1. The participants will then all generate the same master key for use in the protocols.

If one of the sessions is using unilateral TLS and the other is using bilateral TLS then $\mathcal{E}_{trpt}^{TLS^{C \to S}}$ and $\mathcal{E}_{trpt}^{TLS^{C \leftrightarrow S}}$ are not disjoint, as required by Assumption 4.10 and Assumption 5.9 (1). Further, there are also some penetrator paths that start at a bilateral TLS node and end at a unilateral TLS node, as the penetrator can simply forward messages from one session directly into the other. There is no high-level strand that corresponds to this behaviour, as no strand allows the channel to be changed.

Whilst such behaviour contradicts our assumptions, it is clear that any bundle in which such behaviour occurs could be replaced by a bundle in which disjoint message encryption is satisfied by:

1. Replacing every path that crosses between the two sessions by a Receive subpath, then a Send subpath; then

2. Making the penetrator generate fresh nonces, rather than passing the nonces directly between the sessions.

The first step ensures that messages are no longer passed directly between the strands, whilst the second step ensures that the sessions will use disjoint keys and hence $\mathcal{E}_{trpt}^{TLS^{C \to S}}$ and $\mathcal{E}_{trpt}^{TLS^{C \leftrightarrow S}}$ are disjoint. The first transformation step is possible as the above situation can only arise when the penetrator is legitimately involved with both sessions, and thus he knows the keys required to send and receive messages. The second transformation is permitted as the penetrator was the intended participant, so can change the values that he uses.

Unfortunately, the low-level bundle that results from the above transformation is not equivalent to the original bundle, since the regular behaviour has changed as the keys used have been altered. However, the application-layer behaviour has not

changed. We can formalise the notion of equivalence between the two bundles as follows.

Definition 7.1. Two low-level bundles $\mathcal{B}$ and $\mathcal{B}'$ are *application-layer-equivalent* iff there exists a bijection γ between the regular application-layer nodes of $\mathcal{B}$ and $\mathcal{B}'$ (i.e. $\gamma : \mathcal{N}_{\mathcal{B}}^{reg} \cap \mathcal{N}_{payload} \to \mathcal{N}_{\mathcal{B}'}^{reg} \cap \mathcal{N}_{payload}$) such that:

1. For all $(n, n') \in \gamma$, $\hat{\alpha}_{\mathcal{B}}(msg(n)) = \hat{\alpha}_{\mathcal{B}'}(msg(n'))$;
2. γ respects the regular strand structure on application-layer nodes, i.e. for all $n_1, n_2 \in \mathcal{N}_{payload}$, $n_1 \Rightarrow_{\mathcal{B}}^{+} n_2$ iff $\gamma(n_1) \Rightarrow_{\mathcal{B}'}^{+} \gamma(n_2)$.

In the above, the first clause ensure that the application-layer messages are identical. The second clause then ensures that none of the nodes have been re-ordered, as two regular strands will be related by this bijection iff every application-layer node is related by γ. Together, these two points ensure that, if two regular strands in two application-layer-equivalent bundles are related by γ, then the strands can abstract to the same high-level strand. Further, the above induces an obvious bijection on regular strands with application-layer nodes. We now prove that application-layer-equivalent bundles do indeed abstract to equivalent high-level bundles.

Lemma 7.2. Let $\mathcal{B}$ and $\mathcal{B}'$ be low-level application-layer-equivalent abstractable bundles that abstract to $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}'$ respectively. Then $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}'$ are equivalent.

Proof. Let $\mathcal{B}$, $\mathcal{B}'$, $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}'$ be as per the lemma. Let γ be the bijection between the regular application-layer nodes of $\mathcal{B}$ and $\mathcal{B}'$. As $\mathcal{B}$ and $\mathcal{B}'$ abstract to $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}'$, respectively, it follows that there must exist node maps $\hat{\psi}$ and $\hat{\psi}'$ between the nodes of $\mathcal{B}$ and $\hat{\mathcal{B}}$, and $\mathcal{B}'$ and $\hat{\mathcal{B}}'$, respectively. We define the function $\gamma' : \mathcal{N}_{\hat{\mathcal{B}}}^{reg} \to \mathcal{N}_{\hat{\mathcal{B}}'}^{reg}$ by $\gamma'(\hat{n}) \mathrel{\widehat{=}} \hat{\psi}'(\gamma(\hat{\psi}^{-1}(\hat{n})))$. This is a bijection as γ, $\hat{\psi}$ and $\hat{\psi}'$ are bijections on regular application-layer nodes. Further, γ' satisfies the following properties:

1. For all $(\hat{n}, \hat{n}') \in \gamma'$, $msg(\hat{n}) = msg(\hat{n}')$;
2. For all regular nodes $\hat{n}, \hat{n}'$, $\hat{n} \Rightarrow_{\hat{\mathcal{B}}}^{+} \hat{n}'$ iff $\gamma'(\hat{n}) \Rightarrow_{\hat{\mathcal{B}}'}^{+} \gamma'(\hat{n}')$.

The above properties immediately imply that $\hat{\mathcal{B}}$ and $\hat{\mathcal{B}}'$ are equivalent. □

As with standard bundle equivalence, application-layer equivalence does not preserve the correctness of application-layer correctness properties that include $\preceq$ (cf. Section 5.1.3). Therefore, we introduce a relation $\trianglerighteq_{app}$, analogous to $\trianglerighteq$ (from Definition 5.5), that ensures $\preceq$ only decreases in size (recalling that we are only interesting in using $\preceq$ positively). We also require that transport-layer encryptions are replaced with encryptions such that the inverse keys are equivalently confidential. This ensures that the penetrator cannot learn the value of application-layer messages in one bundle, but not the other. This is required when we later consider how confidentiality of application-layer values is preserved by $\trianglerighteq_{app}$.

Definition 7.3. A low-level bundle $\mathcal{B}$ *can be application-layer reduced to* $\mathcal{B}'$, denoted $\mathcal{B} \trianglerighteq_{app} \mathcal{B}'$, iff:

1. $\mathcal{B}$ and $\mathcal{B}'$ are application-layer-equivalent using a bijection γ;

2. For all regular application-layer nodes $n_1, n_2 \in \mathcal{N}_{payload} \cap \mathcal{N}_{\mathcal{B}'}$, if $n_1 \preceq_{\mathcal{B}'} n_2$ then $\gamma^{-1}(n_1) \preceq_{\mathcal{B}} \gamma^{-1}(n_2)$;

3. For all regular application-layer nodes $n \in \mathcal{N}_{payload} \cap \mathcal{N}_{\mathcal{B}}$, if $es \leq \mathsf{expath}^{\mathsf{c}}(chan(n))$ is such that $\{|m|\}_k \sqsubseteq_{es} msg(n)$ then, letting $\{|m'|\}_{k'} \sqsubseteq_{es} msg(\gamma(n))$, k is complex iff k' is, and k'^{-1} is confidential iff k^{-1} is confidential.

It is clear that the transformation that we described above will result in a bundle that is application-layer-equivalent to the original bundle. Thus, we can formalise the notion that any TLS bundle can be converted into an application-layer-equivalent bundle that satisfies out assumptions as follows.

Claim 7.4. Let Σ be a low-level strand space that satisfies layer-disjoint encryption, except for disjoint message encryption or disjoint transport-layer payload messages between unilateral and bilateral TLS. Then, there exists a strand space $\Sigma^{TLS^{C \leftrightarrow S}}$, satisfying full layer-disjoint encryption and such that there is a bijection χ between the bundles of Σ and Σ' such that, for all $(\mathcal{B}, \mathcal{B}') \in \chi$, $\mathcal{B} \trianglerighteq_{app} \mathcal{B}'$.

We believe that the truth of the above claim is intuitively clear. However, a formal proof requires a complete analysis of both forms of TLS, which is beyond scope of this thesis.

7.2 Bundle Predicate Preservation

We now consider how to show that the transformation of Claim 7.4 preserves the correctness of high-level bundle predicates. We do so by proving that application-layer equivalence preserves the correctness of high-level bundle predicates. The proofs in this section follows the same outline as the proofs in Section 6.4 and Section 6.5.

Definition 7.5. Let $\mathcal{B}$ and $\mathcal{B}'$ be application-layer-equivalent bundles using a bijection γ. We write $x \prec_\gamma y$ iff one of the following clauses applies:

- $x, y \in \mathcal{A} \cup \mathbb{N} \cup \mathcal{C} \cup \mathcal{I} \cup \hat{\mathcal{A}} \cup \mathit{Channels} \cup \{+, -\}$ and $x = y$;
- $x \in \mathcal{N}_{\mathcal{B}}$, $y \in \mathcal{N}_{\mathcal{B}'}$, $\gamma(x) = y$ and $\hat{\alpha}_{\mathcal{B}}(msg(x)) = \hat{\alpha}_{\mathcal{B}}(msg(y))$;
- x is a low-level strand in $\mathcal{B}$, y is a low-level strand in $\mathcal{B}'$, $app\_height_{\mathcal{B}}(st) = app\_height_{\mathcal{B}'}(st)$ and for all $i \leq app\_height_{\mathcal{B}}(st)$, $(x, app\_node(x, i)) \prec_\gamma (y, app\_node(y, i))$;
- x, y are sets of strands in $\mathcal{B}$ and $\mathcal{B}'$ respectively such that there exists a bijection μ such that for each pair $(st, st') \in \mu$, $st \prec_\gamma st'$.

We lift the above definition to logical term environments as follows: $\Gamma \prec_\gamma \Gamma'$ iff Γ and Γ' are defined on the same variables and for each variable v, $x(v) \prec_\gamma y(v)$.

Note that $\prec_\gamma$ is an equivalence relation. We now prove that if two bundles are application-layer-equivalent, then evaluating a logical term in each of the bundles results in a value that is related using the $\prec_\gamma$ relation.

Lemma 7.6. Let $\mathcal{B}$ and $\mathcal{B}'$ be low-level bundles that are application-layer-equivalent using a bijection γ. Further, let t be a logical term and Γ, Γ' be low-level term environments defined on all free variables of t such that $\Gamma \prec_\gamma \Gamma'$. Then $[\![t]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![t]\!]^{\Gamma'}_{\mathcal{B}'}$.

Proof. Let t, $\mathcal{B}$, $\mathcal{B}'$, γ, Γ and Γ' be as per the lemma. We prove the result by structural induction on the logical term, t.

Case $\mathsf{node}(n, t)$**:** By applying the inductive hypothesis it follows that $[\![n]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![n]\!]^{\Gamma'}_{\mathcal{B}'}$ and that $[\![t]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![t]\!]^{\Gamma'}_{\mathcal{B}'}$. Thus, by definition of the $\prec_\gamma$ relation on strands, it follows that (assuming $n \leq \mathit{app\_height}_{\mathcal{B}}([\![t]\!]^{\Gamma}_{\mathcal{B}})$):

$$\begin{aligned}[\![\mathsf{node}(n,t)]\!]^{\Gamma}_{\mathcal{B}} &= ([\![t]\!]^{\Gamma}_{\mathcal{B}}, \mathit{app\_node}([\![t]\!]^{\Gamma}_{\mathcal{B}}, [\![n]\!]^{\Gamma}_{\mathcal{B}})) \\ &\prec_\gamma ([\![t]\!]^{\Gamma'}_{\mathcal{B}'}, \mathit{app\_node}([\![t]\!]^{\Gamma'}_{\mathcal{B}'}, [\![n]\!]^{\Gamma'}_{\mathcal{B}'})) \\ &= [\![\mathsf{node}(n,t)]\!]^{\Gamma'}_{\mathcal{B}'}.\end{aligned}$$

Hence, $[\![\mathsf{node}(n, t)]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![\mathsf{node}(n, t)]\!]^{\Gamma'}_{\mathcal{B}'}$, as required.

Case $\mathsf{strand}(t)$**:** By the inductive hypothesis and the definition of $\prec_\gamma$ it follows that $[\![t]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![t]\!]^{\Gamma'}_{\mathcal{B}'}$. Thus, let $n = [\![t]\!]^{\Gamma}_{\mathcal{B}}$ and $n' = [\![t]\!]^{\Gamma'}_{\mathcal{B}'}$ and observe that, by Lemma 6.11, n and n' are regular. Thus, by definition of γ it follows that each application-layer node on $\mathit{strand}(n)$ must map to a node in $\mathcal{B}'$. Further, as γ requires the $\Rightarrow$ relation to be preserved, it follows that each node in $\mathcal{N}_{\mathit{payload}}$ on $\mathit{strand}(n)$ must map to a node on $\mathit{strand}(n')$ (since $\gamma(n) = n'$). Moreover, as every node in $\mathcal{N}_{\mathit{payload}}$ on $\mathit{strand}(n')$ must be mapped to by a node on $\mathit{strand}(n)$ and γ is a bijection, it follows that the strands must be of the same application-layer height (i.e. $\mathit{app\_height}_{\mathcal{B}}(\mathsf{strand}(n)) = \mathit{app\_height}_{\mathcal{B}'}(\mathsf{strand}(n'))$).

By definition of γ, the $\Rightarrow$ relation must be preserved. Hence the only option is to map each node $i \leq \mathit{app\_height}_{\mathcal{B}}(\mathit{strand}(n))$ to the i^{th} application-layer node on $\mathit{strand}(n')$ and thus, for each such i:

$$\begin{aligned}&\gamma((\mathit{strand}(n), \mathit{app\_node}(\mathit{strand}(n), i))) = \\ &\qquad (\mathit{strand}(n'), \mathit{app\_node}(\mathit{strand}(n), i)).\end{aligned}$$

Hence, $[\![\mathsf{strand}(t)]\!]^{\Gamma}_{\mathcal{B}} \prec_\gamma [\![\mathsf{strand}(t)]\!]^{\Gamma'}_{\mathcal{B}'}$.

The remaining cases follow trivially from the induction hypothesis. □

Given the above we can now prove that application-layer reduction preserves satisfaction of high-level bundle properties (cf. Lemma 6.21 and Lemma 6.26).

Lemma 7.7. Let $\mathcal{B}$ and $\mathcal{B}'$ be low-level bundles such that $\mathcal{B} \trianglerighteq_{\mathit{app}} \mathcal{B}'$ and ϕ be a closed high-level bundle predicate such that $\preceq$ occurs only positively. Then, if $(\mathcal{B}', \emptyset) \vDash \phi$ then $(\mathcal{B}, \emptyset) \vDash \phi$.

Proof. Let ϕ, $\mathcal{B}$ and $\mathcal{B}'$ be as per the lemma. Further, let γ be the bijection between $\mathcal{B}$ and $\mathcal{B}'$, as per Definition 7.1. Without loss of generality, we assume that ϕ is in negation normal form. We prove the claim by structural induction on the formula ϕ using the inductive hypothesis that, if $\Gamma \prec_\gamma \Gamma'$ and $(\mathcal{B}', \Gamma') \vDash \phi$, then $(\mathcal{B}, \Gamma) \vDash \phi$.

Case $t_1 \preceq t_2$**:** This follows immediately from Lemma 7.6 and the definition of application-layer reduction.

Cases $\mathsf{Confidential}(t)$, $\neg\mathsf{Confidential}(t)$**:** Informally, this holds because the confidentiality of application-layer values is not affected by the precise values of the transport-layer keys and both bundles use the same application-layer keys. This is because of our disjoint encryption assumptions, from Assumption 5.9, that prevent application-layer encryptions from intersecting with transport-layer encryptions, meaning that the penetrator will not be able to compromise any new application-layer encryptions in $\mathcal{B}'$. Further, application-layer reduction (Definition 7.3) requires that confidential transport-layer encryption keys are replaced only with confidential encryption keys, meaning that the penetrator will not be able to decrypt the contents of any more transport-layer messages in $\mathcal{B}'$.

If can be formally proven in the same way as Lemma 6.15, firstly using Lemma 7.6. The only difference in the proof is the use of penetrator subpaths, rather than high-level penetrator strands.

Cases $\mathsf{UniquelyOriginates}(t_1, t_2)$, $\neg\mathsf{UniquelyOriginates}(t_1, t_2)$**:** By Lemma 7.6 $[\![t_1]\!]^{\Gamma}_{\mathcal{B}} = [\![t_1]\!]^{\Gamma'}_{\mathcal{B}'}$ and $\gamma([\![t_2]\!]^{\Gamma}_{\mathcal{B}}) = [\![t_2]\!]^{\Gamma'}_{\mathcal{B}'}$. The result then follows by observing that γ preserves the application-layer content of messages. Therefore, it preserves origination in the application layer, and hence unique origination in the application layer.

Cases $t_1\ \sigma'\ t_2$, $\neg(t_1\ \sigma'\ t_2)$, $\sigma' \in \{\sqsubseteq, =, <\}$**:** This follows immediately from Lemma 7.6, noting that γ ensures there is a bijection on nodes and strands.

The remaining cases follow immediately from Lemma 7.6 and the inductive hypothesis. □

The following proposition shows that whenever a property is proven correct in a high-level abstraction of the transformed TLS strand space, then it is also correct in the original strand space that did not satisfy Assumption 5.9.

Proposition 7.8. Let Σ be a low-level strand space that satisfies layer-disjoint encryption, except for disjoint message encryption between unilateral and bilateral TLS or disjoint transport-layer payload messages between unilateral and bilateral TLS. Further, let $\Sigma^{TLS^{C\leftrightarrow S}}$ be the strand space resulting from Claim 7.4, $\Sigma_E^{TLS^{C\leftrightarrow S}}$ be related to $\Sigma^{TLS^{C\leftrightarrow S}}$ by Definition 5.15, ϕ be a closed high-level bundle predicate and $\hat{\Sigma}_E^{TLS^{C\leftrightarrow S}}$ be a strand space abstracting $\Sigma_E^{TLS^{C\leftrightarrow S}}$. Then, if every bundle of $\hat{\Sigma}_E^{TLS^{C\leftrightarrow S}}$ satisfies ϕ then every bundle of Σ satisfies ϕ.

Proof. Let Σ, $\Sigma_E^{TLS^{C\leftrightarrow S}}$, $\hat{\Sigma}_E^{TLS^{C\leftrightarrow S}}$ and ϕ be as per the lemma. By Claim 7.4, $\Sigma^{TLS^{C\leftrightarrow S}}$ satisfies layer-disjoint encryption and hence Proposition 6.22 can be applied to deduce that every bundle of $\Sigma^{TLS^{C\leftrightarrow S}}$ satisfies ϕ. Further, by Claim 7.4, there is a bijection χ between the bundles of Σ and $\Sigma^{TLS^{C\leftrightarrow S}}$ such that if $\chi(\mathcal{B}) = \mathcal{B}'$, then $\mathcal{B} \trianglerighteq_{app} \mathcal{B}'$. Hence, Lemma 7.7 can be applied to deduce that every bundle of Σ satisfies ϕ, as required. □

7.3 Summary

In this chapter we have shown how TLS can be transformed into a form that satisfies Assumption 5.9. In Section 7.1 we explained why TLS does not satisfy our assumptions, before defining a bundle transformation that preserves application-layer behaviour, but removes transport-layer behaviour that contradicts Assumption 5.9. Then, in Section 7.2 we proved that the transformations of the previous chapters are sound, in that they do not remove application-layer attacks against protocols. These results, together with those of Section 3.3, imply that WebAuth, if implemented using TLS as its transport-layer, satisfies the correctness properties of Section 3.3.

7.4 Related Work

In this section we discuss related work to Chapters 4–7. The problem of when it is safe to compose protocols has been widely considered before [GT00, DDMR07, CD08, ACG+08, CC10]. However, most existing approaches consider when it is safe to compose protocols together in parallel, rather than when it is safe to compose the protocols on top of one another, or *vertically*. These two problems are subtly different in a very important way: when parallel compositions are being considered, terms for one protocol do not often occur embedded inside the terms of another protocol. Clearly when protocols are being vertically composed, application-layer terms will occur inside the transport-layer terms. We believe that this is one reason why the result appears to be more complex to prove.

In [GM11] Groß and Mödersheim consider what they term *vertical protocol composition*, which corresponds to our notion of layering protocols. They develop a statically-checkable condition that guarantees arbitrary composition of the protocols introduces no new attacks. In one sense, this is more general than the result we prove since it permits arbitrary stacks of protocols to be composed, whereas we have only considered the two layer case. However, it only allows transport-layer protocols that establish two symmetric keys, one for protecting messages sent in each direction, meaning it is not a general result. Further, the disjointness condition that they require prohibits a large class of encryptions from being shared between protocols. In particular, if a transport and an application-layer protocol were to both use public-key certificates, then these protocols would not be considered disjoint and would violate the condition.

In [MV11] Mödersheim and Viganò also propose a statically-checkable condition that allows for the composition of precisely two layers in a very similar way to the approach presented in this thesis. However, their assumptions are more restrictive than ours, and prevent TLS from being used as a secure transport-layer (since it does not support transport-layer protocols with an unbounded number of messages), as well as prohibiting a large class of application-layer protocols. For example, they require the disjointness of *all* non-atomic messages, whereas we only require certain encryptions to be disjoint. They also require that distinct agents send disjoint payloads, which appears to prohibit messages from being forwarded by agents, as is commonplace. Additionally, they currently only consider transport-layer protocols where a single message is sent.

The conditions that we propose in this chapter most closely relate to those of

[GT00], in which the authors detail sufficient conditions for protocols to be composed securely in parallel, using the strand spaces model. Unlike other parallel composition methods, this does permit terms from one protocol to be re-used inside another (e.g. for login tickets). The main assumption in [GT00] essentially requires that encryptions produced by one protocol may only be removed by the same protocol. For example, if one protocol produces $\{\!|m|\!\}_k$ as a login ticket and then passes this value to a second protocol, then the second protocol may not undo the encryption but must treat this value as an indivisible value. Using this the authors prove that any protocols that satisfy the condition can be safely composed in parallel.

Protocol Composition Logic [DDMR07], henceforth PCL, is a logic in which security protocols can be proven correct. Uniquely, one of the main goals of PCL is to allow protocols to be easily composed either in parallel, or sequentially (i.e. protocol A runs and then protocol B is run using keys established by the first protocol). In PCL, an *invariant* is identified for each protocol such that the invariant implies that protocol correctness holds. Then, providing that the invariants are not violated by other protocols, it immediately follows that the protocols are still correct in parallel or sequential composition.

Chapter 8

Multi-Layer Protocol Analysis

The results of the previous chapters have provided an effective and sound way of verifying protocols that consist of precisely two layers. However, many application-layer protocols actually consist of many more layers. Further, some application-layer protocols might actually be easier to analyse if they are decomposed into a number of layers, rather than considering them as one monolithic protocol. For example, an application-layer protocol layered on a unilateral TLS connection might send a username and password pair as the first message, in order to authenticate the client to the server. This effectively turns a unilateral TLS connection into a bilateral TLS connection for subsequent application-layer messages. In this chapter we develop a technique that enables us to formalise and prove this notion.

In more detail, consider trying to prove the correctness of the above protocol. This protocol can be decomposed into three layers: the unilateral TLS layer, the username/password layer and the actual application-layer (excluding the username/password exchange). One way to prove that this protocol is correct is as follows:

1. Prove unilateral TLS is correct in the low-level strand spaces model and provides an $\mathcal{AC}$ channel, but such that the sending name is unauthenticated;
2. Prove that the username/password protocol is correct in that it provides a bilateral $\mathcal{AC}$ channel, providing it is layered on an arbitrary unilateral $\mathcal{AC}$ channel;
3. Prove that the application-layer protocol is correct in the high-level strand spaces model when layered on a bilateral $\mathcal{AC}$ channel.

Clearly (1) and (3) can already be accomplished (the former using the standard strand spaces model and the latter using the high-level model of Section 3.1). In this chapter we provide a way of proving (2), i.e. to prove that a concrete transport-layer protocol provides certain security guarantees when layered on an arbitrary (i.e. abstracted) secondary transport-layer. This allows an arbitrary number of layered to be combined together via a simple inductive argument. This also enables us to prove the correctness of the username and password example above (which must be one of the most widely deployed protocols on the internet). Further, thanks to the abstractions involved, the proofs of the correctness of each layer are entirely independent, meaning that any of the transport-layers could be exchanged for an

alternative that provides equivalent guarantees. Also, since the proofs of each layer are independent, it follows that the correctness proofs for each layer can be reused.

The Strand Spaces In order to formalise what we are proving, we firstly consider the different transport-layer protocols that are under consideration. Firstly, there is the *bottom* transport-layer, which, in the above username/password example, this would be the unilateral TLS protocol. The new transport-layer protocol we refer to as the *middle* transport-layer. In the above, this would correspond to the username/password protocol itself. The *combined* transport-layer protocol refers to the explicit combination of the two (i.e. considering them as one protocol).

There are also three different strand spaces under consideration in these proofs:

Bottom This is a low-level strand space that contains an explicit instantiation of the combined transport-layer protocol.

Middle This is a high-level strand space that is an abstraction of the bottom strand space, but only abstracts away from the bottom transport-layer.

Top This is a high-level strand space that abstracts the bottom strand space away from the combined transport-layer, meaning that it abstracts directly to the application-layer protocol.

Note that both the middle and the top strand spaces are high-level strand spaces: messages are tuples of the form $(A_\psi,\ B_\phi,\ i,\ m,\ c)$, and penetrator strands include SD, FK strands etc. Also, there are two ways of viewing the bottom strand space: we can view the strand space as containing transport-layer protocols that allow abstraction either to the middle strand space, or to the top strand space. We formalise this in Section 8.2.

In this chapter we prove that providing there are no penetrator subpaths of a prohibited form in the middle strand space for the combined transport-layer protocol, then there are no penetrator subpaths of a prohibited form in the low-level strand space for the combined transport layer. Hence, if a given middle-transport-layer protocol can be proven correct in the middle strand space, then it is guaranteed that its composition with *any* bottom transport layer that satisfies the required guarantees is also, itself, a secure transport layer. Further, this means that the top-most application-layer protocol can be analysed in the usual way.

We prove this result in two stages, along the lines of Chapter 4 and Chapter 5. We firstly prove that if there exists a normal interference-free bundle of the bottom strand space that contains a prohibited subpath for the combined protocol, then there exists a high-level bundle in the middle strand space that also contains a prohibited subpath. We then prove that if there is an arbitrary bundle of the bottom strand space that contains a prohibited subpath, then the transformations of Chapter 5 preserve the prohibited subpath. Together, these prove that analysing the middle transport layer in the middle strand space is sufficient.

Outline We start in Section 8.1 by formalising what a middle-transport-layer protocol is. Further, we define high-level analogues of the penetrator subpaths from Chapter 4 that represent subpaths in high-level bundles against the middle transport layer. In Section 8.2 we consider how the bottom, middle and top strand spaces

are related and define an abstraction function from the middle to the top strand space. Further, we also define what it means for a channel to be the combined channel, i.e. the explicit combination of the bottom and middle transport-layers. In Section 8.3 we then prove that, subject to the semantic condition from Chapter 4, if a low-level bundle contains a prohibited penetrator subpath, then there exists a high-level bundle in the middle strand space, containing a corresponding penetrator subpath. In Section 8.4 we use the results of Chapter 5 to change the semantic condition into a statically-checkable condition. In Section 8.5 we give two examples of how the theorems in this chapter can be applied by proving the correctness of two example multi-layer protocols, including the username/password protocol described above. Lastly, in Section 8.6, we summarise the results of the chapter and make detailed comparisons with related work.

8.1 High-Level Channels

Since we are going to prove the correctness of transport-layer protocols in the high-level strand spaces model (in the middle strand space), we firstly consider how to model arbitrary transport-layer protocols in a high-level strand space. In this section we define high-level analogues of many of the low-level strand space definitions of Section 4.1. We deliberately define these in similar ways to how they are defined for the low-level case, in order to allow us to prove, in later sections, that the two definitions are compatible.

Firstly, we define a high-level version of extraction paths (cf. Definition 4.2) that extract a particular term from the application-layer content of a high-level term. If the high-level term is sent on $\bot$, then we should be able to extract values from it in an identical fashion to the low-level case. However, we need to add an extra case to allow the application-layer content to be extracted from a message sent over a non-$\bot$ channel: this corresponds to the action of a RV or LN strand.

Definition 8.1. A *high-level extraction path* is either an extraction path (i.e. from $\mathcal{EP}$), or is $\langle \mathsf{Payload} \rangle \hat{} es$ where $es \in \mathcal{EP}$. The partial function $\widehat{extract}$ is defined as follows:

$$\widehat{extract}((A_\psi,\ B_\phi,\ i,\ m,\ c), \langle \mathsf{Payload} \rangle \hat{} es) \mathrel{\widehat{=}} extract(m, es)$$
$$\widehat{extract}((A_\psi,\ B_\phi,\ i,\ m,\ \bot), es) \mathrel{\widehat{=}} extract(m, es).$$

For example, assuming $c \neq \bot$, $\widehat{extract}((A_\psi,\ B_\phi,\ i,\ t_1 \hat{} t_2,\ c), \langle \mathsf{Payload}, 1 \rangle) = t_1$, whilst $\widehat{extract}((?,\ ?,\ \_,\ t_1 \hat{} t_2,\ \bot), \langle 1 \rangle) = t_1$.

We now prove a lemma analogous to Lemma 4.3 and show that there is an extraction path corresponding to each high-level constructive or destructive penetrator path.

Lemma 8.2. Let p be a constructive or destructive penetrator path in a high-level bundle $\hat{\mathcal{B}}$ starting at a node n_1 and ending at a node $n_{|p|}$. Providing p does not traverse a key edge or a KG strand there exists a high-level extraction path es such that:

- If p is destructive $\widehat{extract}(msg(n_1), es) = appmsg(n_{|p|})$; we define $\widehat{\mathsf{expath}}\,(p) \mathrel{\widehat{=}} es$;

- If p is constructive $\widehat{extract}(msg(n_{|p|}), es) = appmsg(n_1)$; we define $\widehat{\mathsf{expath}}^{\sim}(p) \mathrel{\widehat{=}} es$.

Proof. Let p, $\hat{\mathcal{B}}$, n_1 and $n_{|p|}$ be as per the lemma. We consider how to prove (1), noting that the proof for (2) can be constructed in a similar fashion. There are two cases to consider.

Firstly, if $chan(n_1) = \bot$ then it follows that all nodes on the penetrator path must have $chan(n) = \bot$, by definition of the penetrator strands. Thus, a proof identical to that of Lemma 4.3 can be applied to construct a suitable extraction path, noting the above observation.

Otherwise, it follows that p must start with either a LN or RV strand, since these are the only destructive penetrator strands that can have a non-$\bot$ channel. Further, it immediately follows that the remainder of the penetrator strand will be on $\bot$. Thus, as above, a suitable extraction path can be constructed and $\mathsf{Payload}$ prepended to yield the required extraction path. □

Using the above we can now define what a *high-level channel* is. Intuitively, these correspond to the middle transport layer and are high-level analogues of channels. More precisely, the username and password protocol is a family of high-level channels, with one high-level channel for every bottom transport-protocol. Thus, the bottom transport layer is modelled as a normal channel (i.e. it is a member of *Channels*), the middle transport layer is modelled as a high-level channel, as defined below, whilst the explicit combination of the bottom and middle transport protocols is modelled as a normal channel. In Definition 8.6 we define what it means for a channel to actually be the explicit combination of the bottom and middle transport-layer channels.

Definition 8.3. Let $\hat{\Sigma}$ be a high-level strand space. A *high-level channel* $\hat{c}$ requires the following to be defined:

- $\widehat{transport}(\hat{c}) \neq \bot$ that gives the channel the protocol is layered on[1];
- $\widehat{\mathcal{T}}^{\hat{c}}_{payload} \subseteq \hat{\mathcal{A}}$ that is the set of high-level terms that encode application-layer messages on this channel, which must be defined such that for all $t \in \widehat{\mathcal{T}}^{\hat{c}}_{payload}$, $chan(t) = \widehat{transport}(\hat{c})$;
- $\widehat{\mathsf{expath}^{\mathsf{c}}}(\hat{c})$ that is the high-level extraction path that extracts the application-layer message from terms in $\widehat{\mathcal{T}}^{\hat{c}}_{payload}$ (cf. $\mathsf{expath}^{\mathsf{c}}(c)$)[2].

For each high-level bundle $\hat{\mathcal{B}}$ of $\hat{\Sigma}$ we assume the existence of the following functions:

- $\widehat{sender}^{\hat{c}}_{\hat{\mathcal{B}}}(t) : \widehat{\mathcal{T}}^{\hat{c}}_{payload} \to \mathcal{I}$ that extracts the sender of t;
- $\widehat{recipient}^{\hat{c}}_{\hat{\mathcal{B}}}(t) : \widehat{\mathcal{T}}^{\hat{c}}_{payload} \to \mathcal{I}$ that extracts the recipient of t;
- $\widehat{seqno}^{\hat{c}}_{\hat{\mathcal{B}}}(t) : \widehat{\mathcal{T}}^{\hat{c}}_{payload} \to \mathcal{S}$ that extracts the sequence number of t.

[1] Equivalently, we could define a high-level channel as a function from a transport-layer channel to a high-level channel and remove this clause. We choose not to do this since it would only complicate the definitions.

[2] Since $\widehat{transport}(\hat{c}) \neq \bot$ it follows that $\widehat{\mathsf{expath}^{\mathsf{c}}}(\hat{c})$ must start with $\mathsf{Payload}$.

We elide the bundle from $\widehat{sender}$, $\widehat{recipient}$ and $\widehat{seqno}$ whenever it is clear from the context. The set of *high-level channels* is denoted as $\widehat{Channels}$. The union of $\widehat{\mathcal{T}}^{\hat{c}}_{payload}$ over all $\hat{c} \in \widehat{Channels}$ is denoted by $\widehat{\mathcal{T}}_{payload}$.

We require that $\widehat{transport}(\hat{c}) \neq \bot$ since it enables us to simplify a number of the results in this section. This is not a restriction in practice since a channel that is layered on the bottom channel can be analysed in the normal low-level strand spaces model.

As an example, which we will use as a running example throughout this chapter, consider the (middle) transport-layer protocol that takes an $\mathcal{AC}$ channel that does not provide sequence numbers, and augments it so that it does provide sequence numbers. This can be accomplished by simply pairing each application-layer message m with the corresponding sequence number, e.g. $1\hat{}m$. We formalise this as follows. $\widehat{transport}(\hat{c})$ is defined as an arbitrary $\mathcal{AC}$ channel that does not provide sequence numbers; $\widehat{\mathcal{T}}^{\hat{c}}_{payload}$ is defined as the set of all high-level terms of the form $(A_\psi,\ B_\phi,\ \_,\ i\hat{}m,\ \widehat{transport}(\hat{c}))$; $\widehat{\mathsf{expath}^{\mathsf{c}}}(\hat{c}) \mathrel{\widehat{=}} \langle \mathsf{Payload}, 2 \rangle$; $\widehat{seqno}^{\hat{c}}_{\hat{\mathcal{B}}}(A_\psi,\ B_\phi,\ \_,\ i\hat{}m,\ \widehat{transport}(\hat{c})) \mathrel{\widehat{=}} i$. Then, for each high-level bundle $\hat{\mathcal{B}}$, $\widehat{sender}^{\hat{c}}_{\hat{\mathcal{B}}}(t) \mathrel{\widehat{=}} sender^{\widehat{transport}(\hat{c})}(t)$ and $\widehat{recipient}^{\hat{c}}_{\hat{\mathcal{B}}}(t) \mathrel{\widehat{=}} recipient^{\widehat{transport}(\hat{c})}(t)$. Note that this example illustrates that the middle transport layer is allowed to promote values, such as the sender or recipient, directly from the bottom transport layer.

As with the low-level strand spaces model, we need to assume that the set of transport-layer terms on distinct channels is disjoint in order to make sure that we can detect over which channel a transport-layer message is sent. Whilst this might appear restrictive, note that for any channel where the channel ends are not ?, it will always be the case that the term sets are disjoint.

Assumption 8.4. For all $\hat{c}_1, \hat{c}_2 \in \widehat{Channels}$ such that $\hat{c}_1 \neq \hat{c}_2$, $\widehat{\mathcal{T}}^{\hat{c}_1}_{payload} \cap \widehat{\mathcal{T}}^{\hat{c}_2}_{payload} = \emptyset$.

8.2 Defining Layering of Channels

As mentioned in the introduction to this chapter, there are two ways of abstracting the bottom strand space: either the strand space could be abstracted directly to the top strand space (i.e. abstracting the combined channel), or could be abstracted to the middle strand space (i.e. abstracting the bottom transport-layer only). This suggests that there are actually two ways of viewing a low-level strand space: either it can be viewed with certain nodes being transport-layer nodes of the combined channel, or it can be viewed with a superset of those nodes being transport-layer nodes of the bottom transport-layer. For example, Figure 8.1 is a low-level bundle that contains a Renumber penetrator subpath for the combination of the running example and a simple bottom transport-layer protocol. In this figure, n_1 and n_{15} can be identified either as nodes for the combined transport-layer protocol, or as nodes for the bottom transport-layer protocol.

We formalise this by introducing the concept of a view, as follows.

Definition 8.5. A *view* v of a low-level strand space consists of a low-level strand space along with compatible definitions of $Channels, \mathcal{N}_{trpt}, \mathcal{N}_\bot, \mathcal{N}_{non\text{-}payload}, \mathcal{T}^c_{payload}$

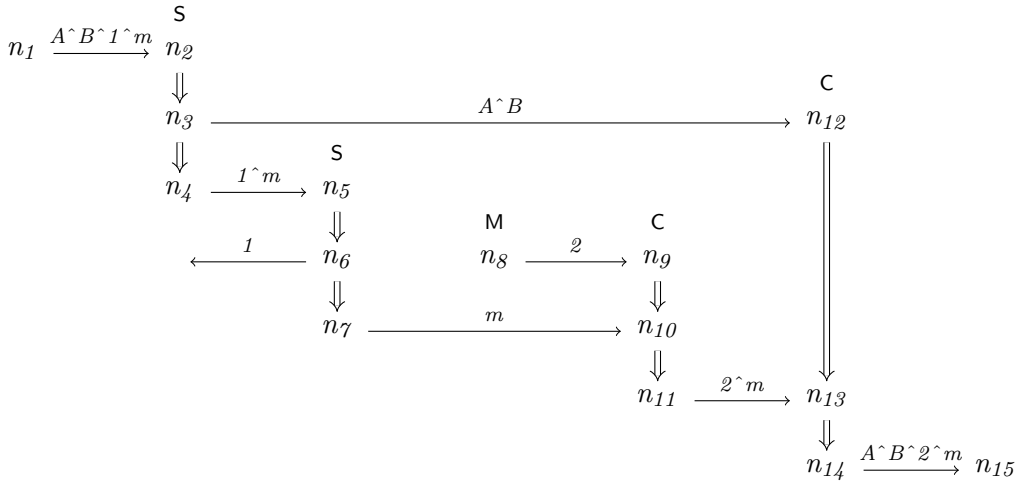


Figure 8.1: The penetrator path $\langle n_1, n_2, n_4, n_4, n_5, n_7, n_{10}, n_{11}, n_{13}, n_{14}, n_{15} \rangle$ is a $v_{B \to T}$-Renumber subpath for the running example layered on a bottom transport-layer protocol that encodes $(A_?,\ B_?,\ \_,\ m,\ c)$ as $A \hat{} B \hat{} m$. The above bundle is a low-level version of the HL-Renumber penetrator subpath of Figure 8.3.

and $\mathcal{T}^{c}_{non\text{-}msg}$ such that Assumptions 4.4, 4.5, 4.6, 4.8 and 4.10 hold. When the definition of a set, function, etc. is affected by the view, we indicate the view using a dot. For example, $v.\mathcal{N}_{trpt}$ or $v.\hat{\alpha}(t)$.

When the lemmas of earlier sections are referenced, in particular those of Chapter 4, we will explicitly indicate which view is being used.

Using the above, we can define what it means for a low-level channel to be the combined protocol in the bottom strand space. Thus, we are essentially defining how an implementation of the multiple layers must be constructed. Informally, we require:

1. There must be two views for the bottom strand space. Intuitively, the *top* view of the bottom strand space abstracts directly to the top strand space, whilst the *middle* view abstracts to the middle strand space.

2. The channel extraction path for the combined protocol must be composed from the extraction paths of the two transport layers.

3. The channels prohibit the same kinds of transport-layer penetrator subpaths.

4. Every transport-layer node for the combined protocol must be a transport-layer node for the bottom-transport-layer protocol.

5. Every high-level term obtained by abstracting a message of a transport-layer node of the combined protocol to the middle strand space must be a high-level transport-layer term for the middle transport layer.

6. The high-level abstraction of a low-level transport-layer term of the combined protocol matches that produced by abstracting via the middle strand space.

These correspond to (1)–(6) in the following definition. We claim that the above list is reasonable, and does in fact correspond to how implementations are likely to work, but of course this is a proof obligation on the user.

Definition 8.6. Let Σ be a low-level strand space, $\hat{\Sigma}$ be a high-level strand space and $\hat{c}$ be a high-level channel. Given two views $v_{B\to T}$ and $v_{B\to M}$, c *is the abstraction of* $\hat{c}$ *in* Σ iff:

1. The *top* view $v_{B\to T}$ of Σ includes $c \in \mathit{Channels}$. The *middle* view $v_{B\to M}$ of Σ includes $\widehat{\mathit{transport}}(\hat{c}) \in \mathit{Channels}$ but $c \notin \mathit{Channels}$.

2. $\mathsf{expath^c}(c) = \mathsf{expath^c}\left(\widehat{\mathit{transport}}(\hat{c})\right)\hat{}\, es$ where $\langle \mathsf{Payload}\rangle\hat{}\, es = \widehat{\mathsf{expath^c}}(\hat{c})$.

3. $\hat{c}$ and c prohibit the same transport-layer penetrator subpath types.

4. In the bottom strand space, every transport-layer payload or non-payload node for the combined protocol must be a transport-layer payload node for the bottom-transport-layer protocol. Formally:

$$v_{B\to T}.\mathcal{N}^c_{trpt} \cup v_{B\to T}.\mathcal{N}^c_{non\text{-}payload} \subseteq v_{B\to M}.\mathcal{N}^{\widehat{transport}(\hat{c})}_{trpt}.$$

   Note that these sets are not necessarily equal because many different high-level channels could be layered on $\widehat{\mathit{transport}}(\hat{c})$.

Further, for each low-level bundle $\mathcal{B}$ of Σ:

5. Every middle strand space term reached by abstracting the message of a transport-layer node of the combined protocol in the bottom strand space must be a transport-layer term for the high-level channel:

$$\{v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(n)) \mid n \in v_{B\to T}.\mathcal{N}^c_{trpt}\} \subseteq \widehat{\mathcal{T}}^{\hat{c}}_{payload}.$$

6. For all bundles $\hat{\mathcal{B}}$ in the middle strand space that abstract $\mathcal{B}$, and for all $t \in \mathcal{T}^c_{payload}$:

$$\begin{aligned} v_{B\to T}.\mathit{sender}^c_{\mathcal{B}}(t) &= \widehat{\mathit{sender}}^{\hat{c}}_{\hat{\mathcal{B}}}(v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(t)) \\ v_{B\to T}.\mathit{recipient}^c_{\mathcal{B}}(t) &= \widehat{\mathit{recipient}}^{\hat{c}}_{\hat{\mathcal{B}}}(v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(t)) \\ v_{B\to T}.\mathit{seqno}^c_{\mathcal{B}}(t) &= \widehat{\mathit{seqno}}^{\hat{c}}_{\hat{\mathcal{B}}}(v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(t)). \end{aligned}$$

As an example of how the top and middle viewes are related, consider Figure 8.1. In this figure, n_1 and n_{15} are in both $v_{B\to T}.\mathcal{N}_{trpt}$ and $v_{B\to M}.\mathcal{N}_{trpt}$, since $v_{B\to T}.\mathit{chan}(n_1)$ is the channel that represents the combined-transport-layer protocol, whilst $v_{B\to M}.\mathit{chan}(n_1)$ is the bottom transport-layer.

Note Definition 8.6 (6) is well defined thanks to Definition 8.6 (5). Further, Definition 8.6 (5) is well defined only if $v_{B\to T}.\mathcal{T}^c_{payload} \subseteq v_{B\to M}.\mathcal{T}^{\widehat{transport}(\hat{c})}_{payload}$. This is because $v_{B\to M}.\hat{\alpha}_{\mathcal{B}^{\widehat{transport}(\hat{c})}}$ can only be applied to terms from $v_{B\to M}.\mathcal{T}^{\widehat{transport}(\hat{c})}_{payload}$, but is being applied to terms from $v_{B\to T}.\mathcal{T}^c_{payload}$. We prove that this inequality holds as follows.

Lemma 8.7. Let Σ be a low-level strand space, $\hat{c}$ be a high-level channel and c be an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$ in Σ. Then:

$$v_{B\to T}.\mathcal{T}^{c}_{payload} \subseteq v_{B\to M}.\mathcal{T}^{\widehat{transport}(\hat{c})}_{payload}.$$

Proof. Let Σ, $\hat{c}$ and c be as per the lemma. By Definition 8.6 (4) it follows that $v_{B\to T}.\mathcal{N}^{c}_{trpt} \subseteq v_{B\to M}.\mathcal{N}^{\widehat{transport}(\hat{c})}_{trpt}$. Therefore, since Assumption 4.5 guarantees that (for any view), for any channel c, $\mathcal{T}^{c}_{payload} = \{msg(n) \mid n \in \mathcal{N}^{c}_{trpt}\}$, the required relationship immediately follows. □

We can now define two additional functions that give the application-layer message and the high-level channel for a high-level transport-layer term, respectively. The latter of these requires Assumption 8.4 in order to be well defined.

Definition 8.8. The following functions are defined on all terms in $t \in \widehat{\mathcal{T}}_{payload}$:

- $\widehat{appmsg}^{\hat{c}} : \widehat{\mathcal{T}}_{payload} \to \mathcal{A}$ that extracts the application-layer message, and is defined by $\widehat{appmsg}^{\hat{c}}(t) \mathrel{\widehat{=}} \widehat{extract}(\widehat{\mathsf{expath^c}}\,(\hat{c})\,, t)$.
- $\widehat{chan} : \widehat{\mathcal{T}}_{payload} \to \widehat{Channels}$ that gives the high-level channel on which the term is sent, and is defined as $\hat{c}$ such that $t \in \widehat{\mathcal{T}}^{\hat{c}}_{payload}$.

Given the above definition of $\widehat{chan}$, we can lift $\widehat{sender}_{\hat{\mathcal{B}}}$ to be defined independently of the channel in the obvious way: $\widehat{sender}_{\hat{\mathcal{B}}}(t) \mathrel{\widehat{=}} \widehat{sender}^{\widehat{chan}(t)}_{\hat{\mathcal{B}}}(t)$. We lift $\widehat{recipient}$, $\widehat{seqno}$ and $\widehat{appmsg}$ in similar ways.

We now define a function $\hat{\chi}$ that abstracts high-level terms from the middle strand space to high-level terms of the top strand space. This is defined in a similar way to $\hat{\alpha}$, which abstracts terms from the low-level bottom strand space to either the middle or the top strand space.

Definition 8.9. Let $\hat{\mathcal{B}}$ be a high-level bundle. The *high-level term mapping function* $\hat{\chi}_{\hat{\mathcal{B}}} : \widehat{\mathcal{T}}_{payload} \to \hat{\mathcal{A}}$ that maps high-level transport-layer terms to the corresponding high-level terms is defined as follows:

1. For $t \in \widehat{\mathcal{T}}_{payload}$, letting $\hat{c} = \widehat{chan}(t)$ and c be the explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$:

$$\hat{\chi}_{\hat{\mathcal{B}}}(t) = (\widehat{sender}^{\hat{c}}_{\hat{\mathcal{B}}}(t),\ \widehat{recipient}^{\hat{c}}_{\hat{\mathcal{B}}}(t),\ \widehat{seqno}^{\hat{c}}_{\hat{\mathcal{B}}}(t),\ \widehat{appmsg}^{\hat{c}}(t),\ c).$$

2. If t is a directed term then $\hat{\chi}_{\hat{\mathcal{B}}}(t)$ has the same direction as t.

For example, using our running example, $\hat{\chi}_{\hat{\mathcal{B}}}(A_\psi,\ B_\phi,\ \_,\ i\hat{}m,\ \widehat{transport}(\hat{c})) = (A_\psi,\ B_\phi,\ i,\ m,\ c)$, assuming c is the explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. The relationship between $\hat{\chi}$ and $\hat{\alpha}$ is indicated in Figure 8.2.

We now prove that the high-level term we reach by abstracting directly from the bottom to the top strand space is equivalent to the high-level term obtained by abstracting via the middle strand space (i.e. using $\hat{\alpha}$ to obtain a term of the middle strand space, and then $\hat{\chi}$ to obtain a term of the top strand space).

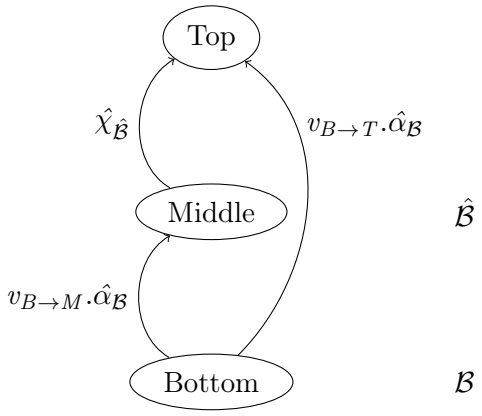


Figure 8.2: An illustration of how the terms of the three strand spaces are related.

Lemma 8.10. Let Σ be a low-level strand space, $\hat{\Sigma}$ be a middle strand space, $\hat{c}$ be a high-level channel and c be a channel that is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$ in Σ. For all bundles $\mathcal{B}$ of Σ, for all middle bundles $\hat{\mathcal{B}}$ of $\hat{\Sigma}$ that abstract $\mathcal{B}$ and for all $n \in v_{B\to T}.\mathcal{N}^c_{trpt} \cap \mathcal{N}_{\mathcal{B}}$:

$$v_{B\to T}.\hat{\alpha}_{\mathcal{B}}(msg(n)) = \hat{\chi}_{\hat{\mathcal{B}}}(v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(n))).$$

Proof. Let Σ, $\hat{\Sigma}$, $\hat{c}$ and c be as per the lemma. Firstly, we prove that the above equation is well defined. Since $v_{B\to M}.\hat{\alpha}$ can only be applied to terms in $v_{B\to M}.\mathcal{T}^{\widehat{transport}(\hat{c})}_{payload}$, but is being applied to terms in $v_{B\to T}.\mathcal{T}^{c}_{payload}$ we require that $v_{B\to T}.\mathcal{T}^{c}_{payload} \subseteq v_{B\to M}.\mathcal{T}^{\widehat{transport}(\hat{c})}_{payload}$. Further, $\hat{\chi}$ can only be applied to terms in $\widehat{\mathcal{T}}^{\hat{c}}_{payload}$ and hence we require that $\{v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(n)) \mid n \in v_{B\to T}.\mathcal{N}^c_{trpt}\} \subseteq \widehat{\mathcal{T}}^{\hat{c}}_{payload}$. The former follows from Lemma 8.7 whilst the latter follows immediately from Definition 8.6 (5).

Let $\mathcal{B}$ be an arbitrary low-level bundle of Σ, $n \in v_{B\to T}.\mathcal{N}^c_{trpt} \cap \mathcal{N}_{\mathcal{B}}$ and $\hat{\mathcal{B}}$ be an arbitrary bundle of $\hat{\Sigma}$ that abstracts $\mathcal{B}$. Further, define $\hat{t}^M \mathrel{\widehat{=}} v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(n))$, $\hat{t}^T_1 \mathrel{\widehat{=}} v_{B\to T}.\hat{\alpha}_{\mathcal{B}}(msg(n))$ and $\hat{t}^T_2 \mathrel{\widehat{=}} \hat{\chi}_{\hat{\mathcal{B}}}(\hat{t}^M)$. In order to prove the desired equivalence it is sufficient to prove that each component of $\hat{t}^T_1$ and $\hat{t}^T_2$ is equal, which we do as follows.

1. By Definition 8.6 (6) and the definition of $\hat{\chi}$, it immediately follows that $sender(\hat{t}^T_1) = \widehat{sender}^{\hat{c}}_{\hat{\mathcal{B}}}(\hat{t}_M) = sender(\hat{t}^T_2)$, as required. Similarly $recipient(\hat{t}^T_1) = \widehat{recipient}^{\hat{c}}_{\hat{\mathcal{B}}}(\hat{t}_M) = recipient(\hat{t}^T_2)$ and $seqno(\hat{t}^T_1) = \widehat{seqno}^{\hat{c}}_{\hat{\mathcal{B}}}(\hat{t}_M) = seqno(\hat{t}^T_2)$, as required.

2. Let $\langle \mathsf{Payload} \rangle \hat{} es = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$.

$$\begin{aligned}
&appmsg(\hat{t}^T_1)\\
&= appmsg^c_{\mathcal{B}}(msg(n))\\
&= extract(msg(n), \mathsf{expath}^{\mathsf{c}}\,(c))
\end{aligned}$$

$$
\begin{aligned}
&= extract(msg(n), \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)\hat{}\, es) && \langle \text{By Definition 8.6 (2)} \rangle \\
&= extract(v_{B\to M}.appmsg(msg(n)), es) \\
&= extract(\widehat{extract}(v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(n)), \langle \mathsf{Payload} \rangle), es) && \langle \text{By Definition 4.11} \rangle \\
&= \widehat{extract}(\hat{t}^M, \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})) && \langle \text{By Definition 8.6 (2)} \\
& && \quad \text{and Definition 8.1} \rangle \\
&= \widehat{appmsg}^{\hat{c}}_{\hat{\mathcal{B}}}(\hat{t}^T_2).
\end{aligned}
$$

3. Since $n \in v_{B\to T}.\mathcal{N}^c_{trpt}$, it immediately follows by definition of $v_{B\to T}.\hat{\alpha}$ that $chan(\hat{t}^T_1) = c$. Further, since $\hat{t}^M \in \widehat{\mathcal{T}}^{\hat{c}}_{payload}$ by Definition 8.6 (5), it therefore follows that $chan(\hat{T}_1) = c$, since $\widehat{chan}(\hat{t}^M) = \hat{c}$ and c is the explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$.

Hence, $\hat{t}^T_1 = \hat{t}^T_2$ and therefore the desired equivalence holds. □

8.2.1 High-Level Penetrator Subpaths

In this section we consider how to model the penetrator's interaction with transport-layer terms in the middle strand space. As in the low-level case, this requires us to define what each of the transport-layer penetrator strands in the top strand space (i.e. SD, FK, etc. strands) corresponds to in the middle strand space. We follow the technique used in the low-level model, in Definition 4.19, and define a high-level penetrator subpath in the middle strand space that corresponds to each type of penetrator strand in the top strand space. For example, considering the running example of a middle transport layer that adds session numbers to each message, a high-level penetrator subpath that represents a RN strand is given in Figure 8.3. In fact, Figure 8.1 gives a low-level version of the high-level penetrator HL-Renumber subpath of Figure 8.3. Thus, when abstracting the low-level bundle according to $v_{B\to M}$, Figure 8.3 would be obtained.

In order to define the high-level penetrator subpaths, we firstly define what it means for a high-level penetrator path to extract or send the application-layer message of a given node. This definition is a high-level analogue of Definition 4.17.

Definition 8.11. In a high-level strand space, we say that a destructive penetrator path p_d starting at a node n *extracts the high-level application message from* n, written p_d $\widehat{\mathsf{app\text{-}extracts}}$ n, iff $\widehat{\mathsf{expath}}\,(p_d) = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$ where $\hat{c} = \widehat{chan}(msg(n))$. A constructive penetrator path p_c ending at a node n *packages the high-level application message of* n, denoted p_c $\widehat{\mathsf{app\text{-}packages}}$ n, iff $\widehat{\mathsf{expath}}^{\sim}\,(p_c) = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$ where $\hat{c} = \widehat{chan}(msg(n))$.

For example, considering Figure 8.3, $\langle n_1, n_2, n_3, n_4, n_6, n_7 \rangle$ $\widehat{\mathsf{app\text{-}extracts}}$ n_1 and $\langle n_6, n_7, n_{10}, n_{11}, n_{12}, n_{13} \rangle$ $\widehat{\mathsf{app\text{-}packages}}$ n_{13}.

Recall that when defining low-level penetrator subpaths that correspond to normal high-level penetrator strands (e.g. HJ strands), we insisted that the penetrator did not divide the application-layer message (cf. Definition 4.18). In order to define high-level penetrator subpaths that correspond to normal high-level penetrator strands, we lift this definition to the high-level strand space model.

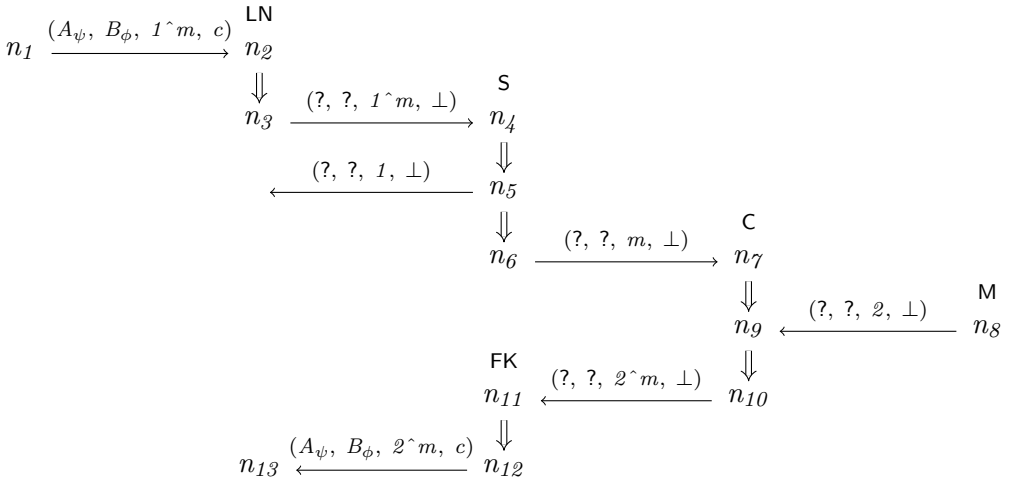


Figure 8.3: The penetrator path $\langle n_1, n_2, n_3, n_4, n_6, n_7, n_{10}, n_{11}, n_{12}, n_{13}\rangle$ is a high-level penetrator renumber subpath for the running example. In the above, c is the bottom-transport-layer and it is assumed that it permits FK and LN strands (thus, it is not an $\mathcal{AC}$ channel, in particular).

Definition 8.12. A normal high-level penetrator path p *transports the high-level application-layer message* iff $msg(p(1)), msg(p(|p|)) \in \widehat{\mathcal{T}}_{payload}$, $\widehat{chan}(p(1)) = \widehat{chan}(p(|p|))$, and there exists p_d and p_c such that $p = p_d\hat{}p_c$, p_d is destructive, p_c is constructive and $\widehat{\mathsf{expath}}\,(p_d) = \widehat{\mathsf{expath}}^{\sim}\,(p_c) \leq \widehat{\mathsf{expath}^{\mathsf{c}}}\left(\widehat{chan}(p(1))\right)$.

We now define the *high-level penetrator subpaths*, which are analogous to the low-level penetrator subpaths defined in Definition 4.19, except each use of *sender* is replaced by $\widehat{sender}$, etc.

Definition 8.13. A penetrator path p in a high-level bundle $\hat{\mathcal{B}}$, starting at n and ending at n' is a *transport-layer high-level penetrator subpath* iff it is of one of the following forms:

HL-Receive
1. p is destructive;
2. $msg(n) \in \widehat{\mathcal{T}}_{payload}$;
3. $\widehat{sender}(msg(n)) \in \mathcal{I}^{reg}$;
4. $\widehat{recipient}(msg(n)) \in \mathcal{I}^{pen}$;
5. p $\widehat{\mathsf{app\text{-}extracts}}$ n.

HL-Learn
1. p is destructive;
2. $msg(n) \in \widehat{\mathcal{T}}_{payload}$;
3. $\widehat{sender}(msg(n)) \in \mathcal{I}^{reg}$;
4. $\widehat{recipient}(msg(n)) \in \mathcal{I}^{reg}$;
5. p $\widehat{\mathsf{app\text{-}extracts}}$ n.

HL-Send
1. p is constructive;
2. $msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. $\widehat{sender}(msg(n')) \in \mathcal{I}^{pen}$;
4. $\widehat{recipient}(msg(n')) \in \mathcal{I}^{reg}$;
5. p $\widehat{\mathsf{app\text{-}packages}}$ n.

HL-Fake
1. p is constructive;
2. $msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. $\widehat{sender}(msg(n')) \in \mathcal{I}^{reg}$;
4. $\widehat{recipient}(msg(n')) \in \mathcal{I}^{reg}$;
5. p $\widehat{\mathsf{app\text{-}packages}}$ n.

HL-Hijack
1. p is normal;
2. $msg(n), msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. Either p is a normal penetrator strand or p transports the high-level application-layer message;
4. $\widehat{appmsg}(n) = \widehat{appmsg}(n')$;
5. $\widehat{chan}(msg(n)) = \widehat{chan}(msg(n'))$;
6. $\widehat{sender}(msg(n)) \neq \widehat{sender}(msg(n')) \wedge \widehat{sender}(msg(n')) \in \mathcal{I}^{reg}$ or $\widehat{recipient}(msg(n)) \neq \widehat{recipient}(msg(n'))$;
7. $\widehat{seqno}(msg(n)) = \widehat{seqno}(msg(n'))$.

HL-Renumber
1. p is normal;
2. $msg(n), msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. Either p is a normal penetrator strand or p transports the high-level application-layer message;
4. $\widehat{appmsg}(n) = \widehat{appmsg}(n')$;
5. $\widehat{chan}(msg(n)) = \widehat{chan}(msg(n'))$;
6. $\widehat{sender}(msg(n)) = \widehat{sender}(msg(n'))$;
7. $\widehat{recipient}(msg(n)) = \widehat{recipient}(msg(n'))$;
8. $\_ \neq \widehat{seqno}(msg(n)) \neq \widehat{seqno}(msg(n')) \neq \_$;

HL-Hijack-Renumber
1. p is normal;
2. $msg(n), msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. Either p is a normal penetrator strand or p transports the high-level application-layer message;
4. $\widehat{appmsg}(n) = \widehat{appmsg}(n')$;
5. $\widehat{chan}(msg(n)) = \widehat{chan}(msg(n'))$;
6. $\widehat{sender}(msg(n)) \neq \widehat{sender}(msg(n'))$ or $\widehat{recipient}(msg(n)) \neq \widehat{recipient}(msg(n'))$;

7. $\widehat{seqno}(msg(n)) \neq \widehat{seqno}(msg(n'))$.

HL-Transmit
1. p is normal;
2. $msg(n), msg(n') \in \widehat{\mathcal{T}}_{payload}$;
3. Either p is a normal penetrator strand or p transports the high-level application-layer message.
4. $\widehat{appmsg}(n) = \widehat{appmsg}(n')$;
5. $\widehat{chan}(msg(n)) = \widehat{chan}(msg(n'))$;
6. $\widehat{sender}(msg(n)) = \widehat{sender}(msg(n'))$;
7. $\widehat{recipient}(msg(n)) = \widehat{recipient}(msg(n'))$;
8. $\widehat{seqno}(msg(n)) = \widehat{seqno}(msg(n'))$.

Using the above we can now define what it means for a high-level channel to be abstractly correct in a high-level bundle. As in the low-level case (cf. Definition 4.29), we need to ensure that we don't prohibit HL-Hijack subpaths that correspond to a HL-Receive subpath followed by a HL-Send subpath.

Definition 8.14. A HL-Hijack or HL-Hijack-Renumber subpath p is a *innocuous high-level penetrator subpath* iff $\widehat{recipient}(p(1)) \in \mathcal{I}^{pen}$ and $\widehat{sender}(p(|p|)) \in \mathcal{I}^{pen}$.

Definition 8.15. Let $\hat{\Sigma}$ be a high-level strand space and $\hat{c}$ be a high-level channel. $\hat{c}$ is *abstractly correct* in $\hat{\Sigma}$ iff every transport-layer high-level penetrator subpath on $\hat{c}$ is either a penetrator strand allowed by the definition of $\hat{c}$, or is an innocuous high-level subpath.

The remainder of this chapter essentially proves that high-level abstract correctness implies low-level abstract correctness. In other words, proving the abstract correctness of a middle-transport-layer protocol in the high-level model is sufficient.

8.3 High-Level Abstract Correctness

We now consider how to prove our first soundness result. In this section we build on the results of Chapter 4 and prove that, subject to a semantic assumption, the multi-layer analysis is sound. In particular, we prove that if a $v_{B\to M}$-interference-free (Definition 4.28) bundle of the bottom strand space contains a prohibited penetrator path of the combined protocol, then the middle strand space contains a corresponding high-level penetrator path of the middle transport layer that is also prohibited. This therefore implies that, subject to the semantic assumption that all bundles can be made interference free, if no prohibited high-level penetrator subpaths exist in the middle strand space, then no prohibited penetrator subpaths exist for the combined transport-layer in the bottom strand space.

Before proving the above result, we prove a few simple lemmas that show how low-level penetrator subpaths in the bottom strand space are related in different views. For example, the penetrator subpath from n_1 to n_{15} of Figure 8.1 is both a $v_{B\to T}$-Renumber subpath, and the concatenation of a $v_{B\to M}$-Learn, an application-layer subpath (from n_5 to n_{11}) and a $v_{B\to M}$-Fake subpath.

Firstly, we prove that a destructive $v_{B\to T}$-penetrator subpath consists of a $v_{B\to M}$-penetrator subpath, followed by some application-layer penetrator path. This can be observed by noting that the middle-transport-layer protocol data is contained solely in the application-layer message of the bottom transport-layer. Hence, the penetrator can only manipulate this by first extracting it using a Receive or Learn subpath for the bottom transport-layer.

Lemma 8.16. Let Σ be a low-level strand space, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. Further, let $\mathcal{B}$ be a normal low-level bundle that contains a destructive $v_{B\to T}$-transport-layer penetrator subpath p. Then $p = p'\hat{}p''$ where p' is a destructive $v_{B\to M}$-transport-layer penetrator subpath (i.e. either a $v_{B\to M}$-Learn or Receive subpath).

Proof. Let Σ, c, $\hat{c}$ and $\mathcal{B}$ be as per the lemma. Since p is a destructive transport-layer penetrator subpath it follows that p must be either a Learn or Receive subpath. Thus, by Definition 4.19:

1. $p(1)$ is a positive regular node and $p(|p|)$ is a negative penetrator node;
2. $p(1) \in v_{B\to T}.\mathcal{N}^{c}_{trpt}$;
3. p $v_{B\to T}$. app-extracts $p(1)$;
4. $v_{B\to T}.sender_{\mathcal{B}}(p(1)) \in \mathcal{I}^{reg}$.

Consider p in $v_{B\to M}$. By Definition 8.6 (4), since $p(1) \in v_{B\to T}.\mathcal{N}^{c}_{trpt}$, $p(1) \in v_{B\to M}.\mathcal{N}^{\widehat{transport}(\hat{c})}_{trpt}$. Further, by Definition 8.6 (2), it follows that there exists i such that $p[1..i]$ $v_{B\to M}$. app-extracts $p(1)$ and thus:

$$\mathsf{expath}\,(p[1..i]) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right).$$

Therefore, define $p' \mathrel{\widehat{=}} p[1\ldots i]$ and $p'' \mathrel{\widehat{=}} p[i+1\ldots|p|]$ and note that by Assumption 4.6, $v_{B\to M}.sender_{\mathcal{B}}(msg(p'(1)) \in \mathcal{I}^{reg}$. Hence, by Definition 4.19, p' is a $v_{B\to M}$-Learn or Receive subpath (depending on whether $v_{B\to M}.recipient_{\mathcal{B}}(p(1))$ is in $\mathcal{I}^{reg}$ or $\mathcal{I}^{pen}$), as required. □

Note that in the above proof, there is nothing to prevent p being a Learn subpath whilst p' is a Receive subpath. This could occur when the bottom transport-layer protocol is not confidential, and therefore permits Learn subpaths, but the middle transport layer implements its own encryption to ensure confidentiality, and thus does not allow Learn subpaths.

We can also prove a similar result for constructive paths. By the same argument as above, it follows that for the penetrator to send an application-layer message over the combined transport layer, he must first construct the necessary values for the middle transport layer and then enclose these within the application-layer message of a middle transport-layer message using a Fake or Send subpath.

Lemma 8.17. Let Σ be a low-level strand space, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. Further, let $\mathcal{B}$ be a normal low-level bundle that contains a constructive $v_{B\to T}$-transport-layer penetrator subpath p. Then $p = p''\hat{}p'$ where p' is a constructive $v_{B\to M}$-transport-layer penetrator subpath (i.e. either a $v_{B\to M}$-Fake or Send subpath).

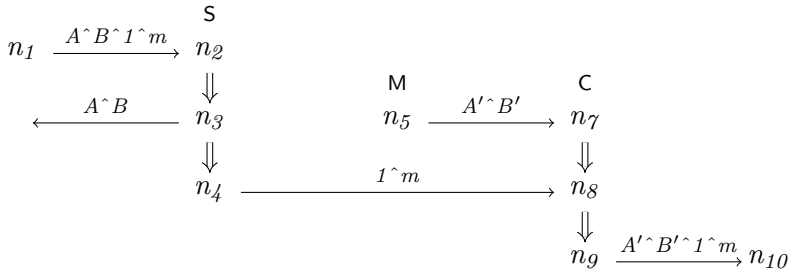


Figure 8.4: The penetrator path $\langle n_1, n_2, n_4, n_9, n_{10} \rangle$ is a $v_{B \to T}$-Hijack subpath for the running example layered on a bottom transport-layer protocol that encodes $(A_?,\ B_?,\ \_,\ am,\ c)$ as $A^\frown B^\frown am$ (in the above $am = 1^\frown m$). Note that the subpath is also a $v_{B \to M}$-Hijack subpath and hence will be abstracted to a HJ strand in the middle strand space.

Proof. This can be proven in a similar fashion to Lemma 8.16. □

We now consider how normal subpaths of $v_{B \to T}$ are related to subpaths in $v_{B \to M}$. As an example, suppose p is a $v_{B \to T}$-Hijack subpath in which *sender* has been altered and *recipient* has remained unchanged. There are, realistically, two different ways in which $\widehat{sender}$ could have been defined: either it merely promotes the values received from the underlying transport-layer, or, similarly to the running example, defines it as some function over the application-layer message extracted from the bottom transport layer. It therefore follows that there are two different ways for p to be viewed in $v_{B \to M}$:

1. p is also a normal $v_{B \to M}$-transport-layer penetrator subpath of some type;
2. p is not a normal $v_{B \to M}$-transport-layer penetrator subpath, but can be decomposed into the concatenation of a destructive $v_{B \to M}$-penetrator subpath, a $v_{B \to M}$-application-layer penetrator subpath, and a constructive $v_{B \to M}$-penetrator subpath.

The first case corresponds to $\widehat{sender}$ simply promoting the value contained in the lower layer, and thus the penetrator can hijack a message simply by using a penetrator subpath for the bottom transport layer, as illustrated in Figure 8.4. The second case corresponds to $\widehat{sender}$ being defined as some function over the application-layer message of the bottom transport layer. In this case it is clear the penetrator cannot alter the sender without first extracting the bottom-transport-layer application-layer message. A similar case for a Renumber subpath is illustrated in Figure 8.1.

Note that in the first case, the $v_{B \to M}$-penetrator subpath does not have to be of the same type. For example, suppose $\widehat{sender}$ was defined as some function over the sequence number field of the bottom transport-layer. In this case it follows that a $v_{B \to T}$-Hijack subpath could actually arise from a $v_{B \to M}$-Renumber subpath. Whilst such transport-layer protocols are unlikely to be practical, we do not prohibit them since it does not simplify the proofs.

We now prove that the above options are the only types of $v_{B \to T}$-penetrator subpaths.

Lemma 8.18. Let Σ be a low-level strand space, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. Further, let $\mathcal{B}$ be a normal low-level $v_{B\to M}$-interference-free bundle that contains a normal $v_{B\to T}$-transport-layer penetrator subpath p for c. Then either:

1. p is a normal $v_{B\to M}$-transport-layer penetrator subpath; or
2. $p = p_1 \hat{} p_d \hat{} p_c \hat{} p_2$ where p_1 is a $v_{B\to M}$-Learn or $\mathsf{Receive}$ penetrator subpath, p_d is a destructive penetrator subpath, p_c is a constructive penetrator subpath, p_2 is a $v_{B\to M}$-Fake or Send penetrator subpath and $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c)$.

Proof. Let Σ, c, $\hat{c}$ and $\mathcal{B}$ be as per the lemma. Since p is a normal $v_{B\to T}$-transport-layer penetrator subpath it follows that p must be either a Hijack, $\mathsf{Renumber}$, $\mathsf{Hijack\text{-}Renumber}$ or $\mathsf{Transmit}$ subpath. By Definition 8.6 (4), since $p(1), p(|p|) \in v_{B\to T}.\mathcal{N}^{c}_{trpt}$ (according to the definition of a $v_{B\to T}$-transport-layer penetrator subpath), $p(1), p(|p|) \in v_{B\to M}.\mathcal{N}^{\widehat{transport}(\hat{c})}_{trpt}$. Therefore, by definition of $\mathcal{N}_{trpt}$, $v_{B\to M}.chan(p(1)) = v_{B\to M}.chan(p(|p|)) = \widehat{transport}(\hat{c})$.

By Definition 4.19 it follows that p transports the (top-level) application-layer message and thus, by Definition 4.18, $p = p_d \hat{} p_c$ where p_d is destructive, p_c is constructive and $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c) \leq \mathsf{expath}^{\mathsf{c}}\,(c)$. Hence, it follows, by Definition 8.6 (2), that either, Case (1), $\mathsf{expath}\,(p_d) \leq \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)$, or, Case (2), $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right) \hat{} es$ where $es \neq \langle\rangle$. In Case (1), $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c) \leq \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)$. Therefore, let es be such that $\mathsf{expath}\,(p_d) \hat{} es = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)$ and observe that:

$$\begin{aligned}
v_{B\to M}.appmsg(p(1)) &= extract(\mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right), msg(p(1))) \\
&= extract(\mathsf{expath}\,(p_d) \hat{} es, msg(p(1))) \\
&= extract(es, msg(p_c(1)))) \\
&= extract(\mathsf{expath}^{\sim}\,(p_c) \hat{} es, msg(p(|p|))) \\
&= extract(\mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right), msg(p(|p|))) \\
&= v_{B\to M}.appmsg(p(|p|)).
\end{aligned}$$

Thus, by Definition 4.19, p must be a normal $v_{B\to M}$-transport-layer penetrator subpath and thus satisfies (1).

Otherwise, in Case (2), $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right) \hat{} es$ for an $es \neq \langle\rangle$. Hence, there must exist p_1, p'_d, p'_c and p_2 such that $p_d \hat{} p_c = p = p_1 \hat{} p'_d \hat{} p'_c \hat{} p_2$, p_1 and p'_d are destructive, p'_c and p_2 are constructive, $\mathsf{expath}\,(p_1) = \mathsf{expath}^{\sim}\,(p_2) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)$, and $\mathsf{expath}\,(p'_d) = \mathsf{expath}^{\sim}\,(p'_c)$. Further, by Assumption 4.6, $v_{B\to M}.sender_{\mathcal{B}}(msg(p_1(1))) \in \mathcal{I}^{reg}$ and $v_{B\to M}.recipient_{\mathcal{B}}(msg(p_2(|p_2|))) \in \mathcal{I}^{reg}$. Therefore, p_1 is either a $v_{B\to M}$-Learn or $\mathsf{Receive}$ subpath, and p_2 is a $v_{B\to M}$-Fake or Send subpath, as required by (2). □

Given low and high-level penetrator paths that are related according to the node abstraction function $\hat{\beta}_1$, it follows that the extraction paths for each should be exactly

the same. This follows from the fact that $\hat{\beta}_1$ maps nodes from a strand of one type to strands of the same type.

Lemma 8.19. Let $\hat{p}$ be either a destructive or constructive high-level penetrator path, and p be a penetrator path of the same type, such that $\hat{p}$ and p are related according to $\hat{\beta}_1$ (i.e. $|\hat{p}| = |p|$ and, for all $1 \leq i \leq |p|$, $\hat{p}(i)\ \hat{\beta}_1\ p(i)$). Then, if p is destructive, $\widehat{\mathsf{expath}}\,(\hat{p}) = \mathsf{expath}\,(p)$, and if p is constructive, $\widehat{\mathsf{expath}}^{\sim}\,(\hat{p}) = \mathsf{expath}^{\sim}\,(p)$.

Proof. This follows immediately from the observation that $\hat{\beta}_1$ maps strands to strands of the same type. □

We now prove the main result of this section and show that, given an interference-free bundle that contains a $v_{B\to T}$-transport-layer penetrator subpath prohibited by the definition of the combined channel, the bundle can be abstracted to a bundle of the middle strand space that contains a high-level version of the prohibited penetrator subpath. Thus, whilst most of the results of the previous pages have shown that a low-level penetrator subpath can be of different types in different views, the following proposition proves that the actual application-layer behaviour (i.e. in the top strand space) is always the same.

Proposition 8.20. Let Σ be a low-level strand space, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{\mathit{transport}}(\hat{c})$. Further, let $\mathcal{B}$ be a normal, $v_{B\to M}$-interference-free, $v_{B\to M}$-abstractly-correct low-level bundle such that c is not $v_{B\to T}$-abstractly correct. Then, there exists a high-level bundle $\hat{\mathcal{B}}$ in the middle strand space such that $\hat{\mathcal{B}}$ abstracts $\mathcal{B}$ and in which $\hat{c}$ is not abstractly correct.

Proof. Let Σ, c, $\hat{c}$ and $\mathcal{B}$ be as per the lemma. Since c is not $v_{B\to T}$-abstractly correct in $\mathcal{B}$ it follows that there must exist a transport-layer penetrator subpath p that is prohibited by the definition of c. We define a high-level bundle $\hat{\mathcal{B}}$ that contains a transport-layer penetrator subpath $\hat{p}$ that is prohibited by the definition of $\hat{c}$. Note that in the following, we partially define a high-level bundle $\hat{\mathcal{B}}$ of the middle strand space that $v_{B\to M}$-abstracts $\mathcal{B}$, but defer to Proposition 4.31 to construct the rest of the bundle. This is justified by observing that $\mathcal{B}$ is normal, $v_{B\to M}$-interference-free and $v_{B\to M}$-abstractly-correct and hence $\mathcal{B}$ can be $v_{B\to M}$-abstracted to a high-level bundle $\hat{\mathcal{B}}$.

We perform a case analysis on the penetrator p, as follows.

p is destructive Therefore, p must either be a $v_{B\to T}$-Learn or Receive[3] subpath for c. Suppose p is a Learn subpath; the proof when p is a Receive is almost identical and is therefore elided. Consider p in $v_{B\to M}$. We aim to construct a penetrator path $\hat{p}$ in a bundle of the middle strand space such that $\hat{p}$ is a high-level transport-layer penetrator subpath of the same type.

By Lemma 8.16, $p = p'^{\frown}p''$ where p' is a $v_{B\to M}$-Learn or Receive subpath, as illustrated in Figure 8.5. Hence, by Definition 4.21, p' can be $v_{B\to M}$-abstracted to a LN or RV strand. Therefore, we define two high-level nodes in $\hat{\mathcal{B}}$, $\hat{n}_1$ and

[3]Whilst no practical transport-layer protocol will prohibit RV strands, there is no reason mathematically why they cannot.

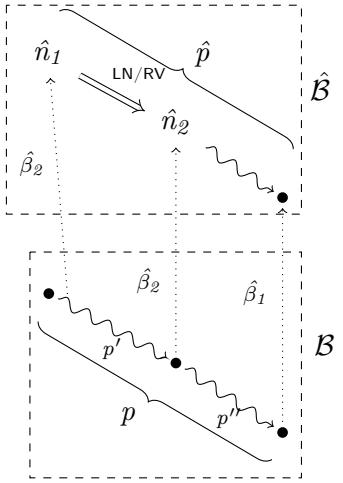


Figure 8.5: An illustration of the case of Proposition 8.20 where p is destructive. In the above, dotted lines indicate the relation that relates the nodes of the two bundles.

$\hat{n}_2$, such that $p'(2)\ \hat{\psi}\ \hat{n}_1$[4], $p'(|p'|-1)\ \hat{\psi}\ \hat{n}_2$ (cf. Definition 4.21) and $\langle \hat{n}_1, \hat{n}_2 \rangle$ is a LN or RV strand, as appropriate.

We now construct a high-level subpath $\hat{p}$ that abstracts p. Note that since p' is a Learn or Receive subpath, $p'(1)$ is a positive regular node and $p'(|p|)$ is a negative penetrator node. Further, as p is a Learn subpath, $p''(|p''|)$ is a negative penetrator node. Thus, p is as follows:

$$p'(1) \to p'(2) \Rightarrow^+ p'(3) \to \ldots p'(|p'|) \Rightarrow^+ p''(1) \to \ldots \to p''(|p''|).$$

We therefore define $\hat{p}$ by: $\hat{p}(1) \mathrel{\hat{=}} \hat{n}_1$, $\hat{p}(2) \mathrel{\hat{=}} \hat{n}_2$ and, for $1 \leq i \leq |p''|$, $\hat{p}(1+i)$ is defined according to the definition of $\hat{\beta}_1$ (cf. Definition 4.13). The remainder of $\hat{\mathcal{B}}$ is constructed according to Proposition 4.31.

We now prove that $\hat{p}$ is a HL-Learn subpath, by proving that (1)–(5) of the definition of a HL-Learn subpath hold, according to Definition 8.13:

1. $\hat{p}$ is destructive by definition of $\hat{p}$.
2. By Definition 4.21, $msg(\hat{p}(1)) = v_{B\to M}.\hat{\alpha}_{\mathcal{B}}(msg(p(1)))$ and therefore, by Definition 8.6 (5), $msg(\hat{p}(1)) \in \widehat{\mathcal{T}}_{payload}$, as required.
3. Note that by Lemma 8.10:

   $$v_{B\to T}.\hat{\alpha}(msg(n_1)) = \hat{\chi}(v_{B\to M}.\hat{\alpha}(msg(n_1))) = \hat{\chi}(msg(\hat{p}(1))).$$

   Therefore, by definition of $\hat{\chi}$ and $v_{B\to T}.\hat{\alpha}$, $\widehat{sender}(msg(\hat{p}(1))) = v_{B\to T}.sender(msg(n_1))$. Hence, since $v_{B\to T}.sender(msg(n_1)) \in \mathcal{I}^{reg}$ (as p is Learn subpath), $\widehat{sender}(msg(\hat{p}(1))) \in \mathcal{I}^{reg}$, as required.
4. As above, it follows that $\widehat{recipient}(msg(\hat{n}_1)) = v_{B\to T}.recipient(msg(n_1))$. Thus, since $v_{B\to T}.recipient(msg(n_1)) \in \mathcal{I}^{reg}$ (as p is Learn subpath), $\widehat{recipient}(msg(\hat{p}(1))) \in \mathcal{I}^{reg}$.

[4]This is not $p'(1)$ since, by definition of a Learn subpath, $p'(1)$ is a regular node.

5. Hence, we need to prove that $\hat{p}$ $\widehat{\mathsf{app\text{-}extracts}}$ $\hat{p}(\mathit{1})$, i.e. $\widehat{\mathsf{expath}}\,(\hat{p}) = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$. By definition of $\hat{p}$: $\widehat{\mathsf{expath}}\,(\hat{p}) = \langle \mathsf{Payload} \rangle \hat{} \widehat{\mathsf{expath}}\,(\hat{p}\,[\mathit{3} \ldots])$. Further, since $\hat{p}\,[\mathit{3} \ldots]$ is related to p'' by $\hat{\beta}_{\mathit{1}}$, it follows by Lemma 8.19 that $\widehat{\mathsf{expath}}\,(\hat{p}\,[\mathit{3} \ldots]) = \mathsf{expath}\,(p'')$. Hence, $\widehat{\mathsf{expath}}\,(\hat{p}) = \langle \mathsf{Payload} \rangle \hat{} \mathsf{expath}\,(p'')$ and therefore, letting $\langle \mathsf{Payload} \rangle \hat{} es = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$:

$$
\begin{aligned}
&\mathsf{expath}^{\mathsf{c}}\left(\widehat{\mathit{transport}}(\hat{c})\right) \hat{} es \\
&= \mathsf{expath}^{\mathsf{c}}\,(c) && \langle \text{By Definition 8.6 (2)} \rangle \\
&= \mathsf{expath}\,(p) && \langle \text{As } p \text{ is a } v_{B \to T}\text{-}\mathsf{Learn} \text{ subpath} \rangle \\
&= \mathsf{expath}\,(p') \hat{} \mathsf{expath}\,(p'') && \langle \text{By definition of } p' \text{ and } p'' \rangle \\
&= \mathsf{expath}^{\mathsf{c}}\left(\widehat{\mathit{transport}}(\hat{c})\right) \hat{} \mathsf{expath}\,(p'') \\
& && \langle p' \text{ is a } v_{B \to M}\text{-}\mathsf{Learn} \text{ or } \mathsf{Receive} \text{ subpath} \rangle
\end{aligned}
$$

   Thus it follows that $es = \mathsf{expath}\,(p'')$, hence $\widehat{\mathsf{expath}}\,(\hat{p}) = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$ and thus $\hat{p}$ $\widehat{\mathsf{app\text{-}extracts}}$ $\hat{p}(\mathit{1})$.

Therefore, $\hat{p}$ is a high-level transport-layer penetrator subpath of the same type as p. Thus, by Definition 8.6 (3) it follows that $\hat{p}$ is also disallowed by the definition of $\hat{c}$ and hence $\hat{\mathcal{B}}$ is not abstractly correct, as required.

p is constructive Therefore p is a Fake or Send subpath. The proof of this case is similar to the above case and is therefore elided.

p is normal Therefore p is a $v_{B \to T}$-Hijack, Hijack-Renumber, Renumber or Transmit subpath. As above, consider p in $v_{B \to M}$. We aim to construct a high-level penetrator path $\hat{p}$ in a bundle of the middle strand space such that $\hat{p}$ is a high-level transport-layer penetrator subpath of the same type. Firstly, we prove by case analysis, using Lemma 8.18, that p can be abstracted to a normal penetrator subpath $\hat{p}$ such that $\hat{p}$ is either, Case (1), a normal penetrator strand, or, Case (2), transports the high-level application-layer message.

For Case (1), suppose p is also a normal $v_{B \to M}$-transport-layer penetrator subpath of some type. By Definition 4.21, p can be $v_{B \to M}$-abstracted to a normal penetrator strand of the same type as the $v_{B \to M}$-transport-layer penetrator subpath (which may not be the same type as p in $v_{B \to T}$). Thus, we define two high-level nodes in $\hat{\mathcal{B}}$, $\hat{n}_{\mathit{1}}$ and $\hat{n}_{\mathit{2}}$, such that $p(\mathit{2})\ \hat{\psi}\ \hat{n}_{\mathit{1}}$, $p(|p| - \mathit{1})\ \hat{\psi}\ \hat{n}_{\mathit{2}}$ and $\langle \hat{n}_{\mathit{1}}, \hat{n}_{\mathit{2}} \rangle$ is a normal high-level strand of the type as p in $v_{B \to M}$. In this case, $\hat{p}$ is defined as $\langle \hat{n}_{\mathit{1}}, \hat{n}_{\mathit{2}} \rangle$ and thus, $\hat{p}$ is a normal penetrator strand, as required.

For Case (2), as illustrated in Figure 8.6, $p = p_{\mathit{1}} \hat{} p_d \hat{} p_c \hat{} p_{\mathit{2}}$ where p_{1} is a $v_{B \to M}$-Learn or Receive penetrator subpath, p_d is a destructive penetrator path, p_c is a constructive penetrator path, p_{2} is a $v_{B \to M}$-Fake or Send penetrator subpath, and $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c)$. Therefore, we abstract p_{1} to a high-level LN or RV strand in an identical fashion to the abstraction performed in the case when p is destructive. p_{2} is abstracted to a high-level SD or FK strand, as per the the case for p being constructive. Lastly, p_d and p_c are abstracted according to Definition 4.13 (i.e. under $\hat{\beta}_{\mathit{1}}$) to high-level penetrator paths $\hat{p}_d$ and $\hat{p}_c$. $\hat{p}$ can then be defined as the concatenation of the destructive high-level

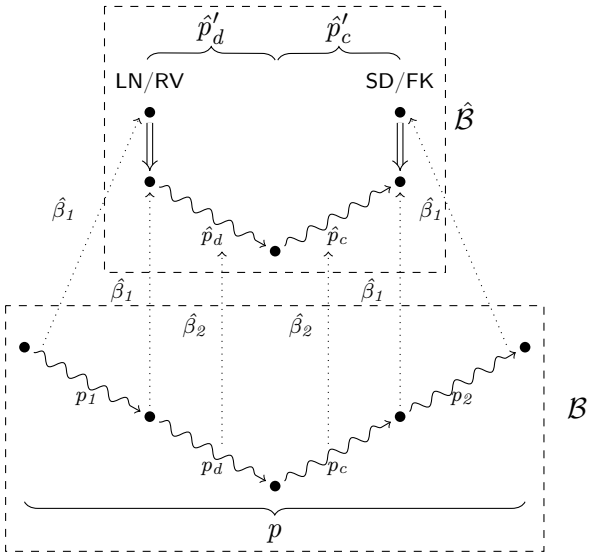


Figure 8.6: An illustration of the case of Proposition 8.20 where p is normal, and is not a normal $v_{B\to M}$-transport-layer penetrator subpath.

strand, the normal penetrator paths $\hat{p}_d$ and $\hat{p}_c$, and the constructive high-level strand.

We now prove that $\hat{p}$ transports the application-layer message. Firstly, note that since $\mathsf{expath}\,(p_d) = \mathsf{expath}^{\sim}\,(p_c)$, by Lemma 8.19, $\widehat{\mathsf{expath}}\,(\hat{p}_d) = \widehat{\mathsf{expath}}^{\sim}\,(\hat{p}_c)$. Further, by construction there exists $\hat{p}'_d$ and $\hat{p}'_c$ such that $\hat{p} = \hat{p}'_d \hat{}\, \hat{p}'_c$, where $\hat{p}'_d$ is destructive, $\hat{p}'_c$ is constructive (i.e. $\hat{p}'_d$ is a LN or RV strand followed by $\hat{p}_d$, and similarly for $\hat{p}'_c$) and such that:

$$\widehat{\mathsf{expath}}\left(\hat{p}'_d\right) = \langle \mathsf{Payload} \rangle \hat{}\, \mathsf{expath}\,(p_d) = \langle \mathsf{Payload} \rangle \hat{}\, \mathsf{expath}^{\sim}\,(p_c) = \widehat{\mathsf{expath}}^{\sim}\left(\hat{p}'_c\right).$$

Thus, all that remains to prove is that $\widehat{\mathsf{expath}}\,(\hat{p}'_d) \leq \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$. By Definition 8.6 (2), there exists es such that $\mathsf{expath}^{\mathsf{c}}\,(c) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right) \hat{}\, es$ where $\langle \mathsf{Payload} \rangle \hat{}\, es = \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$. Since p transports the application-layer message, $\mathsf{expath}\,(p_1 \hat{}\, p_d) \leq \mathsf{expath}^{\mathsf{c}}\,(c)$ and thus, as $\mathsf{expath}\,(p_1) = \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right)$, $p_d \leq es$. However, by construction, $\mathsf{expath}\,(p_d) = \widehat{\mathsf{expath}}\,(\hat{p}_d)$, and thus it follows that $\widehat{\mathsf{expath}}\,(\hat{p}_d) \leq es$. Therefore:

$$\begin{aligned}
&\widehat{\mathsf{expath}}\left(p'_d\right) \\
&= \langle \mathsf{Payload} \rangle \hat{}\, \widehat{\mathsf{expath}}\,(\hat{p}_d) \\
&\leq \langle \mathsf{Payload} \rangle \hat{}\, es \\
&= \mathsf{expath}^{\mathsf{c}}\left(\widehat{transport}(\hat{c})\right).
\end{aligned}$$

Hence, $\hat{p}$ transports the high-level application-layer message.

In both cases the rest of the high-level bundle is constructed according to Proposition 4.31. We now prove that $\hat{p}$ is a prohibited high-level transport-

layer penetrator subpath of the same type as p. Firstly, note that $\hat{p}$ is normal by construction and that, by Definition 8.6 (5), $msg(\hat{p}(1)), msg(\hat{p}(|\hat{p}|)) \in \widehat{\mathcal{T}}_{payload}$. Further, by construction, either, Case (1), $\hat{p}$ is a normal penetrator strand or, Case (2) $\hat{p} = \hat{p}'_d \hat{}\, \hat{p}'_c$ where $\hat{p}'_d$ is destructive, $\hat{p}'_c$ is constructive and $\widehat{\mathsf{expath}}\,(\hat{p}'_d) = \widehat{\mathsf{expath}}^{\sim}\,(\hat{p}'_c) \leq \widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c})$ so $\hat{p}$ transports the high-level application-layer message. Lastly, since $msg(\hat{p}(1)) = v_{B\to M}.\hat{\alpha}(msg(p(1)))$, it follows by Lemma 8.10 that $v_{B\to T}.\hat{\alpha}_{\mathcal{B}}(msg(p(1))) = \hat{\chi}(msg(\hat{p}(1)))$. Hence:

- $\widehat{sender}(msg(\hat{p}(1))) = v_{B\to T}.sender(msg(p(1)))$;
- $\widehat{recipient}(msg(\hat{p}(1))) = v_{B\to T}.recipient(msg(p(1)))$;
- $\widehat{seqno}(msg(\hat{p}(1))) = v_{B\to T}.seqno(msg(p(1)))$;
- $\widehat{appmsg}(msg(\hat{p}(1))) = v_{B\to T}.appmsg(msg(p(1)))$;
- $\widehat{chan}(msg(\hat{p}(1))) = \hat{c}$ and $c = v_{B\to T}.chan(msg(p(1)))$.

Further, since similar results hold for $\hat{p}(|\hat{p}|)$ and $p(|p|)$, it follows, by Definition 8.13, that $\hat{p}$ is a normal high-level transport-layer penetrator subpath of the same type as p. □

Note that in order to apply the above proposition, both views (i.e. $v_{B\to M}$ and $v_{B\to T}$) of the low-level strand space must be well-defined low-level strand spaces. Whilst it is reasonable to assume that $v_{B\to M}$ satisfies the assumptions (since we are abstracting away from bottom transport-layer), it is not reasonable to assume that $v_{B\to T}$ does since this is defined partially in terms of the middle transport-layer. Therefore, we now consider how to easily prove that $v_{B\to T}$ satisfies the necessary assumptions, assuming $v_{B\to M}$ does.

Firstly, observe Assumptions 4.4, 4.5 and 4.6 are essentially defining the low-level strand space, and thus impose no real proof obligations. Assumption 4.8 is a simple disjointness condition, and is a proof obligation on the user of the results to ensure that they only use the protocol in situations where it holds. Lastly, Assumption 4.10 specifies various conditions on the transport-layer terms, including that regular strands use distinct channel ends etc. Clearly, such conditions do impose proof obligations, and further, they would be more easily proven in the middle strand space than in the bottom strand space. Hence, we formalise this assumption as an assumption on the middle strand space and then prove that this implies Assumption 4.10 holds in $v_{B\to T}$.

Definition 8.21. A high-level channel $\hat{c}$ in a middle strand space $\hat{\Sigma}$ is *defined correctly* iff:

1. For all bundles $\hat{\mathcal{B}}$ of $\hat{\Sigma}$, and terms $t_1, t_2 \in \widehat{\mathcal{T}}^{\hat{c}}_{payload}$ sent on $\hat{c}$:
   (a) $name(\widehat{sender}_{\hat{\mathcal{B}}}(t_1)) = ?$ iff $name(\widehat{sender}_{\hat{\mathcal{B}}}(t_2)) = ?$;
   (b) $end(\widehat{sender}_{\hat{\mathcal{B}}}(t_1)) = ?$ iff $end(\widehat{sender}_{\hat{\mathcal{B}}}(t_2)) = ?$;
   (c) $name(\widehat{recipient}_{\hat{\mathcal{B}}}(t_1)) = ?$ iff $name(\widehat{recipient}_{\hat{\mathcal{B}}}(t_2)) = ?$;
   (d) $end(\widehat{recipient}_{\hat{\mathcal{B}}}(t_1)) = ?$ iff $end(\widehat{recipient}_{\hat{\mathcal{B}}}(t_2)) = ?$;
   (e) $\widehat{seqno}_{\hat{\mathcal{B}}}(t_1) = \_$ iff $\widehat{seqno}_{\hat{\mathcal{B}}}(t_2) = \_$;

(f) If $\widehat{seqno}_{\hat{\mathcal{B}}}(t_1) \neq \_$ then $end(\widehat{sender}_{\hat{\mathcal{B}}}(t_1)) \neq ?$ and $end(\widehat{recipient}_{\hat{\mathcal{B}}}(t_1)) \neq ?$ (and similarly for t_2).

2. For all bundles $\hat{\mathcal{B}}$ and strands st and st' in $\hat{\mathcal{B}}$, if $st \neq st'$ then $\widehat{ends}'_{\hat{\mathcal{B}}}(st) \cap \widehat{ends}'_{\hat{\mathcal{B}}}(st') = \emptyset$ (cf. Equation 3.1), where $\widehat{ends}'_{\hat{\mathcal{B}}}$ is formally defined as[5]:

$$\begin{aligned}\widehat{ends}'_{\hat{\mathcal{B}}}(st) \mathrel{\widehat{=}} \{end(\widehat{sender}_{\mathcal{B}}(n)) \mid n \in \mathcal{N}_{\hat{\mathcal{B}}} \wedge msg(n) \in \widehat{\mathcal{T}}^{\hat{c}}_{payload} \\ \wedge sign(n) = + \wedge n \text{ is on } st\} \\ \cup \{end(\widehat{recipient}_{\mathcal{B}}(n)) \mid n \in \mathcal{N}_{\hat{\mathcal{B}}} \wedge msg(n) \in \widehat{\mathcal{T}}^{\hat{c}}_{payload} \\ \wedge sign(n) = - \wedge n \text{ is on } st\}.\end{aligned}$$

3. For all bundles $\hat{\mathcal{B}}$ and for all regular strands st in $\hat{\Sigma}$, the function $th^{st}_{\hat{\mathcal{B}}} : \mathcal{C} \times \mathcal{C} \to \mathcal{S}^*$, which returns the list of sequence numbers sent between two channel ends on a strand, is defined by:

$$\begin{aligned}th^{st}_{\hat{\mathcal{B}}}(\hat{c}, \psi, \phi) \mathrel{\widehat{=}} \langle \widehat{seqno}_{\hat{\mathcal{B}}}(msg(st, i)) \mid i \in \langle 1..\rangle, i \leq |st|, n \in \mathcal{N}_{\hat{\mathcal{B}}}, \\ msg(n) \in \widehat{\mathcal{T}}^{\hat{c}}_{payload}, end(\widehat{sender}_{\hat{\mathcal{B}}}(msg(st, i))) = \psi, \\ end(\widehat{recipient}_{\hat{\mathcal{B}}}(msg(st, i))) = \phi\rangle.\end{aligned}$$

For all regular strands st, and all $\psi, \phi \in \mathcal{C}$, either $th^{st}(\hat{c}, \psi, \phi) \leq \langle 1..\rangle$, or $th^{st}(\hat{c}, \psi, \phi) \in \{\_\}^*$.

Using the above we now prove that this implies Assumption 4.10 holds in any corresponding bottom strand space.

Lemma 8.22. Let Σ be a low-level strand space, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. If $\hat{c}$ is defined correctly, then Σ satisfies Assumption 4.10 in $v_{B\to T}$ with respect to c.

Proof. This follows immediately from Definition 8.21, Assumption 4.10 and the definition of $\hat{\alpha}$. □

8.4 Disjoint Encryption

In the previous section we proved that, subject to $v_{B\to M}$-interference-freedom, if the middle transport-layer protocol is correct in the middle strand space, then it is correct in the bottom strand space. In this section we use the results of Chapter 5 to replace the semantic condition with a statically-checkable condition. In particular, we replace the requirement that each bundle is interference-free with a requirement that the strand space satisfies layer-disjoint encryption, as per Definition 5.8.

Firstly, we prove that if a low-level bundle of the bottom strand space contains a prohibited transport-layer penetrator subpath for the combined transport-layer protocol, then providing the strand space satisfies layer-disjoint encryption (with respect to $v_{B\to M}$) then there exists a bundle of the middle strand space that also contains a prohibited transport-layer penetrator subpath.

[5]Note this definition differs from that given in Definition 3.4, since the latter does not consider which bundle the strand is in.

Lemma 8.23. Let Σ be a low-level strand space that satisfies $v_{B\to M}$-layer-disjoint encryption, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. Further, let $\mathcal{B}$ be a low-level bundle such that c is not $v_{B\to T}$-abstractly correct and $\hat{\Sigma}_E$ be a high-level strand space that contains $\hat{c}$ and that abstracts the low-level strand space Σ_E according to Definition 5.15. Then, there exists a high-level bundle $\hat{\mathcal{B}}$ of $\hat{\Sigma}_E$ such that $\hat{\mathcal{B}}$ abstracts $\mathcal{B}$ and in which $\hat{c}$ is not abstractly correct.

Proof. Let Σ, c, $\hat{c}$, $\mathcal{B}$ and $\hat{\Sigma}_E$ be as per the lemma. Since c is not $v_{B\to T}$-abstractly correct in Σ, there must exist a transport-layer penetrator subpath p that is prohibited by the definition of c. By Proposition 5.34, there exists a normal, $v_{B\to M}$-interference-free bundle $\mathcal{B}'$ of the enlarged strand space Σ_E such that $\mathcal{B} \trianglerighteq \mathcal{B}'$. In order to apply Proposition 8.20 it is necessary to show that Σ_E is not abstractly correct, i.e. the transformations performed by Proposition 5.34 should not remove the prohibited penetrator subpath. Thus, we consider the penetrator subpaths that are *removed* by each of the transformations performed, as follows.

Lemma 5.20: In this lemma a new M strand is connected to a pre-existing message-construction path (Definition 5.4), replacing an existing crossing path (Definition 5.10). The only way this could have an effect is if p traversed a crossing path in a way that meant it was removed by the transformation. If p is a constructive transport-layer penetrator subpath, then it must be a suffix of the message-construction path and is hence unaffected. If p is a destructive transport-layer penetrator subpath then it is unaffected since the join happens in the constructive section. Otherwise, p must be a normal transport-layer penetrator subpath and therefore must be the crossing path. However, this contradicts the definition of a crossing path: all normal transport-layer penetrator subpaths require that the extraction path for the destructive portion is a prefix of the channel extraction path, which is explicitly not the case for a crossing path.

Lemma 5.23: This lemma only removes crossing paths, as above.

Lemma 5.24: This lemma does not remove any penetrator paths, but does alter them by adding redundant S and C strand pairs. If p is a destructive or constructive transport-layer penetrator subpath, then this transformation has no effect. If p is a normal penetrator subpath, then the transformation again has no effect since the values of *sender* etc are left unchanged and, further, since the S and C strands are introduced in pairs, any extraction paths that are required to be equivalent will remain equivalent. Clearly, if too many extra S and C strand pairs were added this could cause the penetrator path to not transport the application-layer message. However, since the extra S and C strands are added only to reveal Send and Receive subpaths, it immediately follows that the path still transports the application-layer message.

Lemma 5.30: This lemma only adds extra S and C strand pairs, as above.

Proposition 5.34: This proposition applies the above lemmas, but also applies Lemma 5.6 to obtain a normal bundle. Note that normalising a bundle will not remove p. Clearly, if p is already a normal penetrator path, then since

it is between two regular nodes it will not be removed. If p is a destructive transport-layer penetrator subpath, starting at a regular node, then again, the transformation can leave the path unaltered. Lastly, if p is a constructive path, it will end at a regular node and can again be preserved by the transformation.

Hence, Σ_E is not $v_{B\to T}$-abstractly correct and thus Proposition 8.20 can be applied to deduce that there exists a high-level bundle of $\hat{\Sigma}_E$ such that $\hat{\mathcal{B}}$ abstracts $\mathcal{B}$ and in which $\hat{c}$ is not abstractly correct, as required. □

Using the above we can now prove our main result. In particular, we prove that if the middle strand space contains no prohibited high-level penetrator subpaths, then the bottom strand space contains no prohibited penetrator subpaths. In particular, this means that we can now analyse protocols in the middle strand space and be confident that this is sound.

Proposition 8.24. Let Σ be a low-level strand space that satisfies $v_{B\to M}$-layer-disjoint encryption, c be a channel and $\hat{c}$ be a high-level channel such that c is an explicit layering of $\hat{c}$ on $\widehat{transport}(\hat{c})$. Further, let $\hat{\Sigma}_E$ be a high-level strand space that contains $\hat{c}$ and that abstracts the low-level strand space Σ_E according to Definition 5.15. Then, if $\hat{c}$ is abstractly correct in $\hat{\Sigma}_E$ then c is abstractly correct in Σ.

Proof. Let Σ, c, $\hat{c}$, $\mathcal{B}$ and $\hat{\Sigma}_E$ be as per the lemma. Suppose, for a contradiction that c is not abstractly correct in Σ. Then it follows that there exists a bundle $\mathcal{B}$ that is not $v_{B\to T}$-abstractly correct with respect to c. Thus, Lemma 8.23 can be applied to deduce that there exists a high-level bundle $\hat{\mathcal{B}}$ of $\hat{\Sigma}_E$ such that $\hat{\mathcal{B}}$ abstracts $\mathcal{B}$ and in which $\hat{c}$ is not abstractly correct, contradicting the abstract correctness of $\hat{\Sigma}_E$. □

8.5 Examples

In this section we present two simple examples of multi-layer protocols and prove them correct. In Section 8.5.1 we consider the running example of a middle transport-layer that ensures messages cannot be reordered. Secondly, in Section 8.5.2 we prove the correctness of the middle transport-layer that converts a unilateral channel into a bilateral channel by use of a username/password exchange, as described in the introduction to this chapter.

8.5.1 Sequence Numbers

In this section we prove the correctness of the running example throughout this chapter. Firstly, we formally define the transport-layer protocol that we are going to verify.

Definition 8.25. The *anti-renumbering* high-level channel, denoted, $\mathcal{S}$, is defined by:

- $\widehat{transport}(\mathcal{S})$ is defined as an arbitrary channel that satisfies $\mathcal{AC}$ and that authenticates the sender and recipient (i.e. if $t \in \hat{\mathcal{A}}$ and $chan(t) = \widehat{transport}(\mathcal{S})$, then $end(sender(t)) \neq ?_?$ and $end(recipient(t)) \neq ?_?$);

- $\mathcal{S}$ satisfies $\mathcal{AC}$;
- $\widehat{\mathcal{T}}^{\mathcal{S}}_{payload}$ is defined as the set of all high-level terms of the form $(A_\psi,\ B_\phi,\ \_,\ i \hat{}\ m,\ \widehat{transport}(\hat{c}))$;
- $\widehat{\mathsf{expath}^{\mathsf{c}}}\,(\hat{c}) \mathrel{\widehat{=}} \langle \mathsf{Payload}, 2 \rangle$;
- For each high-level bundle $\hat{\mathcal{B}}$, $\widehat{seqno}^{\mathcal{S}}_{\hat{\mathcal{B}}}(A_\psi,\ B_\phi,\ \_,\ i \hat{}\ m,\ \widehat{transport}(\hat{c})) \mathrel{\widehat{=}} i$;
- For each high-level bundle $\hat{\mathcal{B}}$, and for all $t \in \hat{\mathcal{A}}$, $\widehat{sender}^{\mathcal{S}}_{\hat{\mathcal{B}}}(t) \mathrel{\widehat{=}} sender(t)$ and $\widehat{recipient}^{\mathcal{S}}_{\hat{\mathcal{B}}}(t) \mathrel{\widehat{=}} recipient(t)$.
- $\mathcal{S}$ satisfies $\mathcal{AC}$.

We now make an assumption that ensures regular strands are correctly checking sequence numbers, as required by any implementation involving sequence numbers.

Assumption 8.26. Let $\hat{\Sigma}$ be a high-level strand space containing $\mathcal{S}$. Then, for all regular strands st and all channel ends ψ and ϕ, $th^{st}_{\hat{\mathcal{B}}}(\mathcal{S}, \psi, \phi) = \langle 1 .. \rangle$ (cf. Definition 8.21).

Using this we can now prove that $\mathcal{S}$ is defined correctly in any middle strand space. In light of Lemma 8.22, this means that if a particular user wishes to use $\mathcal{S}$ as a middle transport-layer, this reduces the proof obligations required to show that $v_{B \to T}$ satisfies the necessary assumptions.

Lemma 8.27. $\mathcal{S}$ is defined correctly in any middle strand space $\hat{\Sigma}$.

Proof. This follows from the fact that apart from $\widehat{seqno}$, all other values, such as $\widehat{sender}$ and $\widehat{recipient}$, are lifted directly from the underlying transport-layer and thus Definition 8.21 (1) and (2) hold. Further, by Assumption 8.26, it follows that sequence numbers are also correctly defined and thus Definition 8.21 (3) holds, as required. □

We now prove that any high-level bundle of the middle strand space does not contain any prohibited high-level transport-layer penetrator subpaths on $\hat{c}$, i.e. the only high-level penetrator subpaths that a high-level bundle contains must be HL-Send, HL-Receive and HL-Transmit subpaths. We prove this via a series of lemmas, several of which are applicable to other middle-transport-layer protocols.

Firstly, we prove if a high-level channel obtains its recipient directly from the underlying transport layer (i.e. $\widehat{recipient} = recipient$), and the underlying channel satisfies $\mathcal{C}$, then any high-level bundle cannot contain any HL-Learn subpaths. This follows from the fact that a HL-Learn subpath for such a channel must be built using a LN strand for the bottom transport-layer, since $\widehat{recipient} = recipient$. However, such strands are prohibited by the definition of $\mathcal{C}$. Note that this lemma is not specific to $\mathcal{S}$.

Lemma 8.28. Let $\hat{\mathcal{B}}$ be a normal high-level bundle and $\hat{c}$ be a high-level channel such that $\widehat{recipient}^{\hat{c}}_{\hat{\mathcal{B}}}(t) = recipient(t)$. If $\widehat{transport}(\hat{c})$ satisfies $\mathcal{C}$ and, for all $t \in \widehat{transport}(\hat{c})$, $recipient(t) \neq ?_?$, then $\hat{\mathcal{B}}$ contains no HL-Learn subpaths.

Proof. Let $\hat{\mathcal{B}}$, $\hat{c}$ be as per the lemma. Suppose, for a contradiction, that $\hat{\mathcal{B}}$ contains a HL-Learn subpath p. Thus, p $\widehat{\textsf{app-extracts}}$ $p(1)$ and, since $\widehat{transport}(\hat{c}) \neq \bot$ (by definition of a high-level channel), it follows that p must start with a RV or LN strand. Since $\widehat{transport}(\hat{c})$ satisfies $\mathcal{C}$ it follows that only a RV strand is permitted and therefore, $recipient(p(1)) \in \mathcal{I}^{pen}$. Since p is a HL-Learn subpath it follows that $\widehat{recipient}(p(1)) \in \mathcal{I}^{reg}$. Further, since, by assumption, $\widehat{recipient}(p(1)) = recipient(p(1))$, it follows that $recipient(p(1)) \in \mathcal{I}^{reg}$. However, this is a contradiction since $\mathcal{I}^{reg} \cap \mathcal{I}^{pen} = ?_?$ but, by assumption, $recipient(p(1)) \neq ?_?$. Therefore, $\hat{\mathcal{B}}$ contains no HL-Learn subpaths, as required. □

We now prove another lemma that is independent of $\mathcal{S}$ and show that, if a high-level channel obtains its sender directly from the underlying transport-layer (i.e. $\widehat{sender} = sender$), and the underlying channel satisfies $\mathcal{A}$, then no high-level bundle contains any HL-Fake subpaths. As above, this follows from the fact that any such HL-Fake subpath must start with a FK for the underlying channel, which are prohibited by the definition of $\mathcal{A}$.

Lemma 8.29. Let $\hat{\mathcal{B}}$ be a normal high-level bundle and $\hat{c}$ be a high-level channel such that $\widehat{sender}^{\hat{c}}_{\hat{\mathcal{B}}}(t) = sender(t)$. If $\widehat{transport}(\hat{c})$ satisfies $\mathcal{A}$ and, for all $t \in \widehat{transport}(\hat{c})$, $sender(t) \neq ?_?$, then $\hat{\mathcal{B}}$ contains no HL-Fake subpaths.

Proof. Let $\hat{\mathcal{B}}$, $\hat{c}$ be as per the lemma. Suppose, for a contradiction, that $\hat{\mathcal{B}}$ contains a HL-Fake subpath p. Thus, p $\widehat{\textsf{app-packages}}$ $p(|p|)$, and since $\widehat{transport}(\hat{c}) \neq \bot$ (by definition of a high-level channel), it follows that p must start with a SD or FK strand. Since $\widehat{transport}(\hat{c})$ satisfies $\mathcal{C}$, it follows that only a SD strand is permitted and therefore, $sender(p(|p|)) \in \mathcal{I}^{pen}$. Since p is a HL-Fake subpath it follows that $\widehat{sender}(p(|p|)) \in \mathcal{I}^{reg}$. Further, since, by assumption, $\widehat{sender}(p(|p|)) = sender(p(|p|))$, it follows that $sender(p(|p|)) \in \mathcal{I}^{reg}$. However, this is a contradiction since $\mathcal{I}^{reg} \cap \mathcal{I}^{pen} = ?_?$ but, by assumption, $sender(p(|p|)) \neq ?_?$. Therefore, $\hat{\mathcal{B}}$ contains no HL-Fake subpaths, as required. □

We now prove that no high-level bundle contains any prohibited high-level transport-layer penetrator subpaths for $\mathcal{S}$. This can be proven relatively simply, given the above definitions. In particular, the only type of high-level penetrator subpath that does not consist almost entirely of a high-level strand from the bottom transport-layer is a HL-Renumber subpath.

Lemma 8.30. Let $\hat{\mathcal{B}}$ be a normal high-level bundle. Then, $\mathcal{S}$ is abstractly correct in $\hat{\mathcal{B}}$.

Proof. Let $\hat{\mathcal{B}}$ and $\widehat{transport}(\mathcal{S})$ be as per the lemma. Suppose, for a contradiction, that $\hat{\mathcal{B}}$ is not abstractly correct and therefore contains a prohibited high-level transport-layer penetrator subpath p. Clearly p cannot be a HL-Receive, HL-Send or HL-Transmit subpath, since these are not prohibited. Further, by Lemma 8.28 and Lemma 8.29, p also cannot be a HL-Learn or HL-Fake subpath. Hence, p must not be an innocuous high-level subpath and be either a HL-Hijack, HL-Renumber or HL-Hijack-Renumber penetrator subpath.

Therefore, either p is a normal penetrator strand, or p transports the high-level application-layer message. Suppose the former holds. However, since $\widehat{transport}(\mathcal{S})$ satisfies $\mathcal{AC}$ no normal penetrator strands are permitted.

Hence, p must transport the high-level application-layer message. Since $chan(p(1)) = \widehat{transport}(\mathcal{S})$ satisfies $\mathcal{AC}$, it follows that $chan(p(1)) \neq \bot$ and hence the only penetrator strand that p can possibly start with is a RV strand. Hence, since $\widehat{transport}(\mathcal{S})$ satisfies $\mathcal{C}$, $recipient(p(1)) \in \mathcal{I}^{pen}$. Further, since $chan(p(|p|)) = chan(p(1))$ it follows that somewhere along p the penetrator must resend the application-layer message using a SD or FK strand. As $chan(p(1))$ satisfies $\mathcal{AC}$, this has to be using a SD strand which, since p is normal, must come at the end of p. Therefore, $sender(p(|p|)) \in \mathcal{I}^{pen}$ and hence, since $\widehat{sender}(p(|p|)) = sender(p(|p|))$ and $\widehat{recipient}(p(1)) = recipient(p(1))$, it follows that $\widehat{sender}(p(|p|)), \widehat{recipient}(p(1)) \in \mathcal{I}^{pen}$. Hence, p is a innocuous high-level subpath, contradicting the fact that p was a prohibited transport-layer penetrator subpath. Hence, $\hat{\mathcal{B}}$ contains no high-level penetrator subpaths on $\hat{c}$. □

Whilst the above lemma proves that no normal bundles contain any HL-Renumber subpaths, it does not show that an arbitrary high-level bundle does not contain any HL-Renumber subpaths. In the following lemma we prove that this is indeed the case. Note that the following lemma is not specific to $\mathcal{S}$ and can therefore be used for other high-level channels too.

Lemma 8.31. Let $\hat{\Sigma}$ be a high-level strand space and $\hat{c}$ be a high-level channel. If every normal bundle of $\hat{\Sigma}$ that does not contain TX or HJRN strands contains no prohibited high-level transport-layer penetrator subpath on $\hat{c}$, then no bundle of $\hat{\Sigma}$ contains a prohibited high-level transport-layer penetrator subpath on $\hat{c}$.

Proof. Let $\hat{\Sigma}$ and $\hat{c}$ be as per the lemma. Suppose, for a contradiction, that every normal bundle of $\hat{\Sigma}$ that does not contain TX or HJRN strands contains no prohibited high-level penetrator subpath on $\hat{c}$, but that some bundle $\hat{\mathcal{B}}$ of $\hat{\Sigma}$ does. Firstly, we normalise $\hat{\mathcal{B}}$. Note that all normal high-level transport-layer penetrator subpaths are normal paths between regular nodes, by definition, and hence will be present in any normalised bundle. If p is a destructive penetrator path then it immediately follows that, since p starts at a regular node, p can remain in any normal form of $\hat{\mathcal{B}}$. Thus, p must be a constructive penetrator path. However, in this case it follows p ends at a regular node and, therefore, p will be present in any normal form. Thus, it follows that any normalised version of $\hat{\mathcal{B}}$ must contain the prohibited penetrator path p. Hence, Lemma 3.13 can be applied to deduce the existence of a normal bundle $\hat{\mathcal{B}}'$ that contains p. Further, note that the transformation specified in Lemma 4.16 will not remove p: they may alter p by removing TX strands, for example, but the overall effect of p will remain the same. Thus, it follows that there exists a high-level bundle $\hat{\mathcal{B}}''$ that is normal, contains no TX or HJRN strands, but contains a prohibited transport-layer penetrator subpath p, contradicting the assumption that no such bundles exist. Hence, all bundles of $\hat{\Sigma}$ contain no prohibited penetrator subpaths on $\hat{c}$. □

Lastly, we combine the results of the above lemma and prove that if c is a combined protocol with $\mathcal{S}$ as the middle-layer-transport protocol then it is abstractly correct in every low-level strand space.

Proposition 8.32. Let Σ be a low-level strand space such that $v_{B\to M}$ and $v_{B\to T}$ are valid views, and such that Σ satisfies $v_{B\to M}$-layer-disjoint encryption. Further,

let c be a channel that is an explicit layering of $\mathcal{S}$ on an abstractly-correct channel c' that satisfies $\mathcal{AC}$ and authenticates the sender. Then, c is abstractly correct in Σ.

Proof. Let Σ, c, and c' be as per the lemma. Let $\hat{\Sigma}_E$ be an arbitrary high-level strand space that abstracts the low-level strand space $\hat{\Sigma}_E$, which is defined according to Definition 5.15. By Lemma 8.30 it follows that all normal bundles of $\hat{\Sigma}_E$ contain no prohibited transport-layer penetrator subpaths for $\mathcal{S}$. Further, by Lemma 8.31 it follows that all bundles of $\hat{\Sigma}_E$ contain no prohibited transport-layer penetrator subpaths for $\mathcal{S}$ and, hence, $\mathcal{S}$ is abstractly correct in $\hat{\Sigma}$. Thus, by Proposition 8.24, c is abstractly correct in Σ_E. Further, since the bundles of Σ are a subset of those of Σ_E, it immediately follows that c is abstractly correct in Σ. □

8.5.2 Username and Password Protocol

In this section we prove the correctness of a middle transport-layer protocol that authenticates the user using a username and password combination. In particular, this protocol can be used in conjunction with unilateral TLS to authenticate the client to the server and thus create a bilateral TLS-like connection. For example, a client C can login to a server S and send an application-layer message m by sending the following messages:

$$\begin{array}{lll} 1. & ?_\psi \rightarrow S_\phi & : C \hat{} passwd_S(C) \\ 2. & ?_\psi \rightarrow S_\phi & : m \end{array}$$

The primary complication when defining the username and password channel is how to extract the username that is being used by the client. For example, the second message of the above example needs to be abstracted to the high-level term $(C_\psi,\ S_\phi,\ m,\ \mathcal{UP})$, but note that C is not contained anywhere within the second message. We solve this in the following definition by defining a function over channel ends that gives the username that was used to login to the server. Thus, in the above example, the function would associate ψ with C. We formalise this function, and its domain, as follows.

Definition 8.33. $\mathcal{C}^{\mathcal{UP}} \subseteq \mathcal{C}$ is defined as the set of all channel ends that are used by clients running the username and password protocol (e.g. given the example in the introduction, $\psi \in \mathcal{C}^{\mathcal{UP}}$).

For each high-level bundle $\hat{\mathcal{B}}$, the function $\mathsf{username\_for}_{\hat{\mathcal{B}}} : (\mathcal{C}^{\mathcal{UP}}, \mathcal{C}^{\mathcal{UP}}) \rightarrow \mathcal{T}_{names}$ gives the username sent as the first message from the first to the second channel end. Formally, $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi)$ is defined as C if there exists n, S such that:

$$\begin{array}{l} n \in \mathcal{N}_{\hat{\mathcal{B}}} \wedge (n \text{ is regular} \vee \exists n' \in \mathcal{N}_{\hat{\mathcal{B}}} \cdot n \rightarrow_{\hat{\mathcal{B}}} n') \\ \qquad \wedge\ msg(n) = +(?_\psi,\ S_\phi,\ 1,\ C \hat{} passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C \rightarrow S})) \end{array} \tag{8.1}$$

or is defined as an arbitrary member of $\mathcal{T}^{pen}_{names}$, denoted by $N_\perp$.

In Assumption 8.36, we make assumptions that ensure $\mathsf{username\_for}$ is well defined, before proving that it is indeed well defined in Lemma 8.38. The definition of $\mathsf{username\_for}_{\hat{\mathcal{B}}}$ can be explained as follows. Suppose ψ is a channel end in $\mathcal{C}^{\mathcal{UP}}$. Then, assuming that the agent is correctly implementing the username and password protocol (we formalise this assumption in Assumption 8.36), it follows that the first

message sent from ψ (formalised by the last clause) must be the message to login to the server, i.e. $C \hat{} passwd_S(C)$. Thus, the above function correctly associates ψ with C. In order to ensure that $\mathsf{username\_for}$ is well defined, we also require the login message to actually be received by another agent (i.e. the $n \rightarrow n'$ clause). If we did not have this clause then it follows that $\mathsf{username\_for}$ in a bundle that contained lots of SD strands that are not connected to anything, each sending a login message, could have many different usernames.

We now formalise the username and password protocol as a high-level channel $\mathcal{UP}$. As with TLS, we actually define two channels: one from the client to the server, denoted $\mathcal{UP}^{C \rightarrow S}$, and one from the server to the client, $\mathcal{UP}^{S \rightarrow C}$. This is necessary because, in particular, the definition of $\widehat{sender}$ and $\widehat{recipient}$ are direction dependant. For example, from the client to the server, $\widehat{sender}$ needs to extract the username for the channel end, whilst from the server to the client $\widehat{sender}$ promotes the value from the layer below. We start by defining the channel from the client to server, leaving the definition of $\mathcal{UP}^{S \rightarrow C}$ until Definition 8.43.

Definition 8.34. The *client to server username and password* high-level channel $\mathcal{UP}^{C \rightarrow S}$, is defined by:

1. $\widehat{transport}(\mathcal{UP}^{C \rightarrow S})$ is defined as an arbitrary channel that satisfies $\mathcal{AC}$ and such that for all $t \in \mathcal{T}_{payload}^{\widehat{transport}(\mathcal{UP}^{C \rightarrow S})}$, $end(sender(t)) \neq ?$, $name(recipient(t)) \neq ?$ and $end(recipient(t)) \neq ?$.
2. $\widehat{\mathcal{T}}_{payload}^{\mathcal{UP}^{C \rightarrow S}}$ is defined as the set of all high-level terms of the form $(A_\psi,\ S_\phi,\ i,\ m,\ \widehat{transport}(\mathcal{UP}^{C \rightarrow S}))$ where $\psi \in \mathcal{C}^{\mathcal{UP}}$ and $i \neq 1$.
3. $\widehat{\mathsf{expath^c}}\,(\mathcal{UP}^{C \rightarrow S}) \mathrel{\widehat{=}} \langle \mathsf{Payload} \rangle$.
4. For each high-level bundle $\hat{\mathcal{B}}$, $\widehat{seqno}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{C \rightarrow S}}(t) = seqno(t) - 1$.
5. For each high-level bundle $\hat{\mathcal{B}}$, $\widehat{recipient}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{C \rightarrow S}}(t) \mathrel{\widehat{=}} recipient(t)$.
6. $\widehat{sender}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{C \rightarrow S}}(t) \mathrel{\widehat{=}} (\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, end(recipient(t))), \psi)$ where $\psi = end(sender(t)))$.
7. $\mathcal{UP}^{C \rightarrow S}$ satisfies $\mathcal{AC}$.

We now prove that $\mathcal{UP}^{C \rightarrow S}$ is defined correctly in any middle strand space. Again, in light of Lemma 8.22 this reduces the proof obligations required to show that $v_{B \rightarrow T}$ satisfies the necessary assumptions if $\mathcal{UP}^{C \rightarrow S}$ is used.

Lemma 8.35. $\mathcal{UP}^{C \rightarrow S}$ is defined correctly in any middle strand space $\hat{\Sigma}$.

Proof. This follows immediately from the definition of $\mathcal{UP}^{C \rightarrow S}$, Assumptions 3.5 and 3.3, and Equation 3.1. □

In order to prove that the username and password channel is correct we require several assumptions on the strand space. In particular, we require:

1. The penetrator does not initially posses any passwords for regular agents. Clearly if this is not the case the penetrator can trivially impersonate regular agents.

2. If $\psi \in \mathcal{C}^{\mathcal{UP}}$ is used to send a message from a regular node n, then there must be a node that precedes n in which the username and password are sent. Further, such a node must be unique. This is required to make sure that regular clients are correctly implementing the protocol by logging on exactly once, as the first message.

3. If $\psi \in \mathcal{C}^{\mathcal{UP}}$ is used to receive a message at a node n, then there must be a node that precedes n in which the valid username and password were received (and hence, checked). Further, such a node must be unique. As above, this ensures that regular servers are correctly implementing the protocol.

4. Usernames and passwords for regular agents are sent only as part of the login message. This is necessary to ensure that the penetrator cannot obtain the username and password via other means.

These are formalised as (1)–(5) below.

Assumption 8.36. $\hat{\Sigma}$ satisfies the following properties:

1. If $U, S \in \mathcal{T}^{reg}_{names}$ then $passwd_S(U) \notin \mathcal{A}^*_{\mathcal{P}}$ and $passwd_S(U)$ is an atom.

2. For all $\psi \in \mathcal{C}^{\mathcal{UP}}$, for all positive regular nodes $n \in \mathcal{N}_{\hat{\mathcal{B}}}$: if $end(sender(n)) = \psi$ then there must exist a positive n' such that $n' \Rightarrow^* n$, $msg(n') = +(?_\psi,\ S_\phi,\ 1,\ C \hat{}\ passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C \to S}))$ for some $C \in \mathcal{T}^{reg}_{names}$.

3. For all $\psi \in \mathcal{C}^{\mathcal{UP}}$, for all negative regular nodes $n \in \mathcal{N}_{\hat{\mathcal{B}}}$: if $end(sender(n)) = \psi$ then there must exist a negative n' such that $n' \Rightarrow^* n$, $msg(n') = -(?_\psi,\ S_\phi,\ 1,\ C \hat{}\ passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C \to S}))$.

4. For all positive regular nodes n in $\hat{\Sigma}$, if $passwd_S(C) \sqsubseteq appmsg(n)$ for some $C \in \mathcal{T}^{reg}_{names}$ and $S \in \mathcal{T}^{reg}_{names}$ then $name(recipient(n)) \neq ?$, $name(recipient(n)) \in \mathcal{T}^{reg}_{names}$ and $chan(n)$ must satisfy $\mathcal{C}$.

Note that the fourth assumption above does not preclude the user from using the same password for multiple honest servers. However, it does prevent the user from using the same password at an honest and a dishonest server.

We now prove that the password for a honest user logging into an honest server is confidential according to Definition 3.14.

Lemma 8.37. If $C, S \in \mathcal{T}^{reg}_{names}$ then $password_S(C)$ is confidential in all high-level bundles $\hat{\mathcal{B}}$.

Proof. Let C and S be as per the lemma and let $\hat{\mathcal{B}}$ be an arbitrary high-level bundle. By Assumption 8.36 (1), $passwd_S(C) \notin \mathcal{A}^*_{\mathcal{P}}$. We now prove that $passwd_S(C)$ is sent confidentially in $\hat{\mathcal{B}}$. Hence, let n be a node of $\hat{\mathcal{B}}$ such that $msg(n) = (A_\psi,\ B_\phi,\ i,\ m,\ c)$ where $passwd_S(C) \sqsubseteq m$. If n is regular then, by Assumption 8.36 (4), $chan(n)$ satisfies $\mathcal{C}$, $name(recipient(n)) \neq ?$ and $name(recipient(n)) \in \mathcal{T}^{reg}_{names}$. Therefore, $passwd_S(C)$ is sent confidentially from positive regular nodes. Thus, by Definition 3.17, t is a safe atom and therefore, by Lemma 3.18, $passwd_S(C)$ is confidential in $\hat{\mathcal{B}}$. □

We now prove that username_for is well defined. For regular channel ends this follows from Assumption 8.36 (2), which ensures that the login message is always sent as the first message. For penetrator channel ends, this is ensured by Assumption 8.36 (3), which ensures that the login message is the first message received by the server. In the following, we use the function $ends(\hat{\mathcal{B}})$ which returns the set of channel ends used in a bundle and is defined by: $ends(\hat{\mathcal{B}}) = \{end(sender(n)), end(recipient(n)) \mid n \in \mathcal{N}_{\hat{\mathcal{B}}}\}$.

Lemma 8.38. Let $\hat{\mathcal{B}}$ be a normal high-level bundle. Then, $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi)$ is well-defined on all $\psi \in ends(\hat{\mathcal{B}}) \cap \mathcal{C}^{\mathcal{UP}}$ and $\phi \in ends(\hat{\mathcal{B}})$.

Proof. Let $\hat{\mathcal{B}}$ be as per the lemma. Firstly, note that, by definition $\mathsf{username\_for}_{\hat{\mathcal{B}}}$ is defined on all ψ and ϕ. Hence, it suffices to show that $\mathsf{username\_for}_{\hat{\mathcal{B}}}$ is uniquely defined. Given the definition of $\mathsf{username\_for}_{\hat{\mathcal{B}}}$, it follows that if $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi) = N_\perp$, then it is by definition uniquely defined. Hence, suppose there exists n_1 such that: $msg(n_1) = +(?_\psi,\ S_\phi,\ 1,\ C \hat{}\ passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C \to S}))$ and either n_1 is regular or there exists n_1' such that $n_1 \rightarrow_{\hat{\mathcal{B}}} n_1'$. We prove that n_1 is uniquely defined by supposing there exists n_2 such that $msg(n_2) = +(?_\psi,\ S'_{\phi'},\ 1,\ C' \hat{}\ passwd'_S(C'),\ \widehat{transport}(\mathcal{UP}^{C \to S}))$ and either n_2 is regular or there exists n_2' such that $n_2 \rightarrow_{\hat{\mathcal{B}}} n_2'$. We prove that $n_1 = n_2$ by a case analysis on n_1.

Firstly, suppose n_1 is a regular node. Then it must be the case that $\psi \in \mathcal{C}^{reg}$ and thus, since $\psi \neq ?$ (by Definition 8.34 (1)), by Equation 3.1, n_2 must lie on the same strand as n_1. Therefore, since $seqno(n_1) = seqno(n_2)$, by Assumption 3.5, $n_1 = n_2$ and thus $\mathsf{username\_for}_{\hat{\mathcal{B}}}$ is well defined.

Otherwise, suppose n_1 is a penetrator node. Thus, $\psi \in \mathcal{C}^{pen}$ and therefore, since $\psi \neq ?$, it follows that n_2 must also be a penetrator node (as no positive regular node sends messages from a penetrator channel end). Further, since SD strands can only be used to send messages to regular channel ends, it follows that n_1' and n_2' must be regular nodes. By definition, $end(recipient(n_1')) = end(recipient(n_2'))$ and therefore, by Equation 3.1, n_1' and n_2' must lie on the same strand. Since Assumption 8.36 (3) requires that $seqno(msg(n_1)) = 1$ and $seqno(msg(n_2)) = 1$, it follows by Assumption 3.5 that $n_1' = n_2'$. Hence, as $n_1 \rightarrow_{\hat{\mathcal{B}}} n_1'$ and $n_2 \rightarrow_{\hat{\mathcal{B}}} n_2' = n_1'$, $n_1 = n_2$. Therefore, $\mathsf{username\_for}_{\hat{\mathcal{B}}}$ is well defined. □

Recall that $\widehat{sender}^{\mathcal{UP}^{C \to S}}_{\hat{\mathcal{B}}}(t)$ is defined as $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi)$ where $\psi = end(sender(t))$, and $\phi = end(recipient(t))$. Thus, in order for $\widehat{sender}$ to be well defined, it has to return a value in $\mathcal{I}$, which is defined as the union of $\mathcal{I}^{reg}$ and $\mathcal{I}^{pen}$. This means that $\widehat{sender}$ is only well defined if $\psi \in \mathcal{C}^{reg}$ iff $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi) \in \mathcal{C}^{reg}$. This can be proven as follows.

Lemma 8.39. Let $\hat{\mathcal{B}}$ be a normal high-level bundle, $\psi \in \mathcal{C}^{\mathcal{UP}}$ and $\phi \in \mathcal{C}$. Then, $\psi \in \mathcal{C}^{reg} \cap ends(\hat{\mathcal{B}})$ iff $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi) \in \mathcal{T}^{reg}_{names}$.

Proof. Let $\hat{\mathcal{B}}$ and ψ be as per the lemma. There are two cases to consider.

1. Firstly, suppose $\psi \in \mathcal{C}^{reg} \cap ends(\hat{\mathcal{B}})$. Then, by definition of *ends* there must exist a regular node n such that $end(sender(msg(n))) = \psi$. We prove that $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi) \in \mathcal{T}^{reg}_{names}$. By Assumption 8.36 (2), it follows that there must exist a positive node $n'' \Rightarrow^* n$ such that $msg(n'') =$

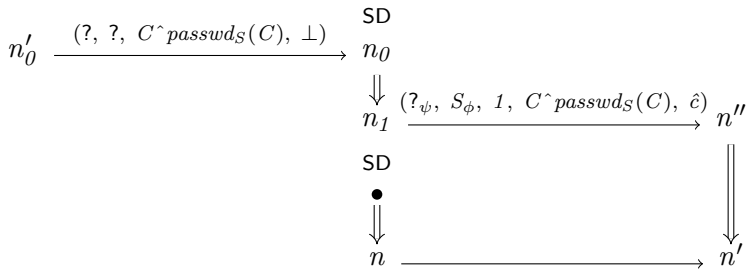


Figure 8.7: An illustration of the second case of Lemma 8.39 (where n is a penetrator node). In the above, $\hat{c} = \widehat{transport}(\mathcal{UP}^{C\to S})$.

$+(?_\psi,\ S_\phi,\ 1,\ C^\frown passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C\to S}))$ for some $C \in \mathcal{T}^{reg}_{names}$. Thus, by Definition 8.33 and Lemma 8.38, $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi) = C$ and thus $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi) \in \mathcal{T}^{reg}_{names}$, as required.

2. Otherwise, $\psi \in \mathcal{C}^{pen} \cap ends(\hat{\mathcal{B}})$. There are two cases to consider. Firstly, if there exists no n and S such that Equation 8.1 holds, then it immediately follows that $\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, \phi) = N_\bot \in \mathcal{T}^{pen}_{names}$, as required. Otherwise, there exists n and S such that Equation 8.1 holds. Then, it follows that n is a penetrator node and thus there exists a node n' such that $n \to n'$. Since $\widehat{transport}(\mathcal{UP}^{C\to S}) \neq \bot$ and satisfies $\mathcal{AC}$, it follows that n can only be the positive node on a SD strand. Thus, it follows that $sender(msg(n)) \in \mathcal{I}^{pen}$, $recipient(msg(n)) \in \mathcal{I}^{reg}$ and hence $\psi \in \mathcal{C}^{pen}$ and $S \in \mathcal{T}^{reg}_{names}$.

   As $recipient(msg(n)) \in \mathcal{I}^{reg}$ it immediately follows that n' is a regular node. By Assumption 8.36 (3), it follows that there must exist a negative node $n'' \Rightarrow^* n'$ such that $msg(n'') = -(?_\psi,\ S_\phi,\ 1,\ C^\frown passwd_S(C),\ \widehat{transport}(\mathcal{UP}^{C\to S}))$; this is illustrated in Figure 8.7. Let n_1 be such that $n_1 \to n''$. Since $\psi \in \mathcal{C}^{pen}$ (as $sender(msg(n)) \in \mathcal{I}^{pen}$) it follows that n_1 must be a positive node on a penetrator strand. Further, since $n_1 \to n''$, $msg(n_1) = +(?_\psi,\ S_\phi,\ 1,\ m,\ C^\frown passwd_S(C))$, as required. Hence, $passwd_S(C)$ is not confidential in $\hat{\mathcal{B}}$. Therefore, $C \notin \mathcal{T}^{reg}_{names}$, since by Lemma 8.37 $passwd_S(C)$ is confidential for all $S, C \in \mathcal{T}^{reg}_{names}$. Thus, $C \in \mathcal{T}^{pen}_{names}$, as required. □

We now prove the correctness of $\mathcal{UP}^{C\to S}$ by proving that it is abstractly correct in an arbitrary high-level bundle. We do this in a series of lemmas. Firstly, we prove that no high-level bundle contains any HL-Hijack, HL-Renumber or HL-Hijack-Renumber subpaths.

Lemma 8.40. Let $\hat{\mathcal{B}}$ be a normal high-level bundle. Then $\hat{\mathcal{B}}$ contains no prohibited HL-Hijack, HL-Renumber or HL-Hijack-Renumber subpaths on $\mathcal{UP}^{C\to S}$.

Proof. Let $\hat{\mathcal{B}}$ be as per the lemma. Suppose, for a contradiction, that $\hat{\mathcal{B}}$ contains a HL-Hijack, HL-Renumber or HL-Hijack-Renumber subpath p on $\mathcal{UP}^{C\to S}$. Thus, either p is a normal penetrator strand, or p transports the high-level application-layer message. If p is a normal penetrator strand, it follows that p must either be a HJ or RN strand. However, since $\widehat{transport}(\mathcal{UP}^{C\to S})$ satisfies $\mathcal{AC}$, p cannot be a RN or HJ strand.

Thus, p must transport the high-level application-layer message. Hence, there exists p_d, p_c such that $p = p_d \hat{} p_c$ and $\mathsf{expath}(p_d) = \mathsf{expath}^{\sim}(p_c)$. Clearly p_d and p_c must be non-empty, otherwise a renumber or a hijack is not possible. Hence, since $\widehat{transport}(\mathcal{UP}^{C\to S}) \neq \bot$, p_d must start with a LN or RV strand and p_c must end with a SD or FK strand. Since $\widehat{transport}(\mathcal{UP}^{C\to S})$ satisfies $\mathcal{AC}$ it follows that only RV and SD strands are permitted and therefore, $recipient(p(1)), sender(p(|p|)) \in \mathcal{I}^{pen}$. Thus, as $\widehat{recipient}(p(1)) = recipient(p(1))$, $\widehat{recipient}(p(1)) \in \mathcal{I}^{pen}$. Further, since $end(\widehat{sender}(p(|p|))) = end(sender(p(|p|)))$, it follows by Lemma 8.39, noting that $end(sender(p(|p|))) \in \mathcal{C}^{\mathcal{UP}}$ since p is a high-level penetrator path on $\widehat{transport}(\mathcal{UP}^{C\to S})$, that $\widehat{sender}(p(|p|)) \in \mathcal{I}^{pen}$. Therefore, p is a high-level innocuous penetrator subpath, contradicting the assumption that p is a prohibited subpath. Hence, $\hat{\mathcal{B}}$ contains no prohibited HL-Hijack, HL-Renumber or HL-Hijack-Renumber subpaths, as required. □

We now prove that $\mathcal{UP}^{C\to S}$ is abstractly correct in every normal bundle. Given the above lemma and Lemma 8.28, we are able to prove this simply by proving that there are no HL-Fake subpaths. This follows simply from Lemma 8.39.

Lemma 8.41. Let $\hat{\mathcal{B}}$ be a normal high-level bundle. Then, $\mathcal{UP}^{C\to S}$ is abstractly correct in $\hat{\mathcal{B}}$.

Proof. Let $\hat{\mathcal{B}}$ and $\widehat{transport}(\mathcal{UP}^{C\to S})$ be as per the lemma. Suppose, for a contradiction, that $\hat{\mathcal{B}}$ is not abstractly correct and therefore contains a prohibited high-level transport-layer penetrator subpath p. Clearly p cannot be a HL-Receive, HL-Send or HL-Transmit subpath, since these are not prohibited. Further, by Lemma 8.28, p also cannot be a HL-Learn subpath and by Lemma 8.40, p cannot be a HL-Renumber, HL-Hijack or HL-Hijack-Renumber penetrator subpath. Hence, p must be a HL-Fake subpath. Thus, as $\widehat{transport}(\mathcal{UP}^{C\to S}) \neq \bot$, p must end with either a SD or FK strand. Further, since p is a HL-Fake subpath, $\widehat{sender}(p(|p|)) \in \mathcal{I}^{reg}$ and thus, by Lemma 8.39, and the definition of $\widehat{sender}$, $sender(p(|p|)) \in \mathcal{I}^{reg}$. Therefore, p ends with a FK strand, contradicting the fact that $\widehat{transport}(\mathcal{UP}^{C\to S})$ satisfies $\mathcal{A}$. Thus, p is abstractly correct in $\hat{\mathcal{B}}$. □

Lastly, we use the above lemma and Proposition 8.24 to prove that if c is a combined protocol with $\mathcal{UP}^{C\to S}$ as the middle-layer-transport protocol, then it is abstractly correct in every low-level strand space.

Proposition 8.42. Let Σ be a low-level strand space that satisfies $v_{B\to M}$-layer-disjoint encryption, c be a channel that is an explicit layering of $\mathcal{UP}^{C\to S}$ on an abstractly correct channel c' that satisfies the appropriate properties. Then, c is abstractly correct in Σ.

Proof. Let Σ, c, and c' be as per the lemma. Let $\hat{\Sigma}_E$ be an arbitrary high-level strand space that abstracts the low-level strand space $\hat{\Sigma}_E$, which is defined according to Definition 5.15. By Lemma 8.41 it follows that all normal bundles of $\hat{\Sigma}$ contain no prohibited transport-layer penetrator subpaths for $\mathcal{UP}^{C\to S}$. Further, by Lemma 8.31 it follows that all bundles of $\hat{\Sigma}$ contain no prohibited transport-layer penetrator subpaths for $\mathcal{UP}^{C\to S}$ and, hence, $\mathcal{UP}^{C\to S}$ is abstractly correct in $\hat{\Sigma}$. Thus, by Proposition 8.24, c is abstractly correct in Σ_E. Further, since the bundles of Σ

are a subset of those of Σ_E, it immediately follows that c is abstractly correct in Σ. □

Recall that the above proposition only proves that the channel from the client to the server is secure, not that the corresponding channel from the server to the client is. Thus, we now consider the latter. Firstly, we formally define the channel from the server to the client, $\mathcal{UP}^{S\to C}$, similarly to Definition 8.34.

Definition 8.43. The *server to client username and password* high-level channel $\mathcal{UP}^{S\to C}$, is defined by:

1. $\widehat{transport}(\mathcal{UP}^{S\to C})$ is defined as an arbitrary channel that satisfies $\mathcal{AC}$ and such that for all $t \in \mathcal{T}_{payload}^{\widehat{transport}(\mathcal{UP}^{S\to C})}$, $end(recipient(t)) \neq ?$, $name(sender(t)) \neq ?$ and $end(sender(t)) \neq ?$.
2. $\widehat{\mathcal{T}}_{payload}^{\mathcal{UP}^{S\to C}}$ is defined as the set of all high-level terms of the form $(A_\psi,\ S_\phi,\ i,\ m,\ \widehat{transport}(\mathcal{UP}^{S\to C}))$ where $\psi \in \mathcal{C}^{\mathcal{UP}}$ and $i \neq 1$.
3. $\widehat{\mathsf{expath}^{\mathsf{c}}}\,(\mathcal{UP}^{S\to C}) \mathrel{\widehat{=}} \langle \mathsf{Payload} \rangle$.
4. For each high-level bundle $\hat{\mathcal{B}}$, $\widehat{seqno}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{S\to C}}(t) = seqno(t) - 1$.
5. For each high-level bundle $\hat{\mathcal{B}}$, $\widehat{sender}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{S\to C}}(t) \mathrel{\widehat{=}} sender(t)$.
6. $\widehat{recipient}_{\hat{\mathcal{B}}}^{\mathcal{UP}^{S\to C}}(t) \mathrel{\widehat{=}} (\mathsf{username\_for}_{\hat{\mathcal{B}}}(\psi, end(sender(t))), \psi)$, where $\psi = end(recipient(t)))$.
7. $\mathcal{UP}^{S\to C}$ satisfies $\mathcal{AC}$.

We now prove that $\mathcal{UP}^{S\to C}$ is abstractly correct. We elide the full proof since it, and the required subsidiary lemmas, are almost identical to those that prove the correctness of $\mathcal{UP}^{C\to S}$.

Proposition 8.44. Let Σ be a low-level strand space that satisfies $v_{B\to M}$-layer-disjoint encryption, c be a channel that is an explicit layering of $\mathcal{UP}^{S\to C}$ on an abstractly correct channel c' that satisfies the appropriate properties. Then, c is abstractly correct in Σ.

Proof. This can be proven by trivially adapting the proofs of Lemma 8.40, Lemma 8.41 and Proposition 8.42 to $\mathcal{UP}^{S\to C}$. □

8.6 Summary

In this chapter we have developed a model in which protocols that consist of an arbitrary number of layers can be proven correct. In particular, we have developed a way of proving that a middle transport-layer protocol is abstractly correct, whilst abstracting away from the transport-layer that it is layered on. Therefore, it follows that the composition of an arbitrary number of layers can be proven by inductively applying the results of this chapter.

In order to do this, we firstly defined, in Section 8.1, the concept of a high-level channel, and what it means for a high-level channel to be abstractly correct in a high-level bundle. This can be used to analyse the correctness of the middle transport layer in the high-level strand spaces model, thus abstracting away from the underlying transport layer. In Section 8.2 we defined what it means for a low-level channel to be the composition of a low-level channel and a high-level channel.

In Section 8.3 we proved the soundness of our approach, subject to a semantic condition. In particular, we proved that if a high-level channel is abstractly correct in a high-level strand space, and it is layered on an abstractly-correct transport layer, then any low-level composition of the two transport layers is abstractly correct. In Section 8.4 we extended the results of Chapter 5 and proved that our approach is sound subject to a statically-checkable condition.

In Section 8.5 we demonstrated the effectiveness of our technique by proving the correctness of two simple examples of multi-layer protocols. The first is a transport-layer protocol that adds sequence numbers to an underlying transport-layer protocol that does not have sequence numbers. The second example proved that a username and password layered on a unilateral-TLS-like transport-layer provides a channel that is bilateral-TLS-like, by authenticating the user to the server. This example is particularly important, given the number of websites and services that make use of this protocol.

Related Work Whilst there has been a reasonable amount of work on verifying when the composition of precisely two layers is secure, there has been surprisingly little on verifying the correctness of an arbitrary number of layers. In particular, relatively few authors have considered the problem of trying to prove that a transport-layer protocol that depends on a second transport-layer protocol is secure, other than by considering their explicit concatenation. We outline the only related work that we are aware of as follows.

In [GM11] Groß and Mödersheim consider how to verify arbitrary stacks of protocols. In this they differentiate between three different types of protocol. *Concrete application protocols* are application-layer protocols that have been explicitly composed with a transport-layer protocol, and are verified independently of the framework. This is essentially equivalent to an application-layer protocol that implements its own transport-layer security and therefore sends all messages over $\perp$. *Abstract application protocols* are like our regular application-layer protocols (i.e. of the top strand space), and require a transport-layer in order to ensure security. Lastly, *channel protocols* are equivalent to our high-level channels, and correspond to the middle transport-layer. As noted in Section 7.4, they only permit transport-layer protocols that establish symmetric keys, and thus do not permit the full range of transport-layers that we do. Our proof approaches also share some similarities. We define what it means for a low-level channel to be the combined channel in Definition 8.6, whilst [GM11] defines the *realise composition* of an abstract application protocol and a channel protocol.

In [MV11], as discussed in Section 7.4, the authors define a statically-checkable condition for when protocols can be securely layered. The authors do not explicitly state that their model supports arbitrary stacks of protocols, but there is no obvious reason why it could not be extended if it does not. The authors do consider the

example of turning a unilateral TLS channel into a bilateral TLS channel using a username and password, but only sketch how it could be considered in their model. As discussed in Section 7.4, their model does not actually permit TLS to be analysed in full.

Chapter 9

Conclusions

In this thesis we have presented the high-level strand spaces model that models the guarantees provided by a wide variety of transport-layer protocols. In particular, it supports both unilateral and bilateral transport-layer protocols that, optionally, can group messages into sessions and may also prevent message reordering. We have also given and proven the correctness of a number of proof rules that allow application-layer protocols to be easily proven correct by abstracting away from the underlying transport-layer protocol. We illustrated the effectiveness of these proof rules by proving the correctness of WebAuth.

The second contribution of this thesis is the soundness proof of the high-level strand spaces model. We have proven that, subject to certain assumptions, whenever there exists a low-level bundle that does not satisfy a protocol correctness property, then there exists a high-level bundle that also does not satisfy the property. Therefore, if a protocol can be proven correct at the high-level, then it is guaranteed to be correct at the low-level. We have also proven how TLS can be transformed to satisfy our assumptions. Thus, given the results of this thesis, it is now possible to deduce that WebAuth, when layered on top of TLS, is secure in the sense of the propositions of Section 3.3.

The soundness result was proven by firstly, in Chapter 4, proving that all interference-free bundles can be abstracted. In Chapter 5 we proved that a bundle $\mathcal{B}$ of a strand space Σ that satisfies disjoint-layered encryption can be transformed to an interference-free bundle $\mathcal{B}'$ of a strand space Σ_E such that $\mathcal{B} \trianglerighteq \mathcal{B}'$. In Chapter 6 we defined a logic of protocol-correctness properties, before proving that whenever $\mathcal{B}$ does not satisfy a correctness property ϕ and $\mathcal{B} \trianglerighteq \mathcal{B}'$, then $\mathcal{B}'$ also does not satisfy ϕ. Lastly, we proved that whenever $\hat{\mathcal{B}}'$ abstracts $\mathcal{B}'$ and $\mathcal{B}'$ does not satisfy a property ϕ, then $\hat{\mathcal{B}}'$ also does not satisfy ϕ.

As a result of this contribution, it is now possible to assert the correctness of application-layer protocols when explicitly layered on TLS, or other protocols for which formal results have been deduced, providing the application-layer protocol has been proven correct in the high-level strand spaces model.

We have also extended the high-level strand spaces model to allow protocols that consist of an arbitrary number of layers to be proven correct. In particular, we defined how to model secondary transport-layer protocols in the high-level strand spaces model. We then defined high-level penetrator subpaths that correspond to each of the transport-layer penetrator strands. We also proved the soundness of the

approach by adapting the results from Chapter 4 and Chapter 5. In particular, we proved that if a channel has been proven correct in the high-level model, then its explicit combination with any channel in the low-level model is guaranteed to also be secure. Lastly, we demonstrated the effectiveness of this technique by proving the correctness a transport-layer protocol that turns a unilaterally authenticating secure transport protocol into a bilaterally authenticating secure transport protocol by authenticating the user to login using a username and password.

One particular benefit of the multi-layer model that we have defined in this thesis is that each layer is abstracted. This means that the correctness proofs for each layer are completely independent and therefore arbitrary stacks can be composed together without any extra proof. Further, the correctness proofs are simpler to construct since only one layer has to be proven correct at a time.

As discussed in Section 3.4, the high-level strand spaces model appears to support the guarantees provided by a wider range of transport-layer protocols than any other technique that proves protocols correct by abstraction. As mentioned in Section 7.4, our model appears to be the first model that proves layered protocols correct by abstraction, and has a formally proven soundness result that depends only on statically-checkable conditions. Lastly, as discussed in Section 8.6, relatively few techniques have explicit support for verifying protocols that consist of an arbitrary number of layers. Further, we are not aware of any techniques that have formally proven soundness results that both permit TLS and can be checked statically.

9.1 Future Work

Whilst the high-level strand spaces model is able to analyse a wide variety of application-layer protocols, there are several extensions that could be made to enhance the range of protocols that it can model. For example, consider an application-layer protocol that allows two users to login to a server and then request a secure channel be setup between them, to allow them to exchange messages. One way of modelling the above protocol would be to have the server send each of the participants in the session a channel end for a secure channel. At the moment, the high-level strand spaces model cannot model such a protocol, since it has no way of communicating channel ends in messages and it provides no way of migrating channel ends from one strand to another. In particular, recall that Equation 3.1 explicitly requires that no channel ends are shared between distinct regular strands. Thus, it would be interesting to extend the high-level strand spaces model to allow channel ends to be *mobile* (in the sense of the pi-Calculus [Mil99]), in that they can be sent between different agents. This might be an interesting way of verifying some peer-to-peer protocols in which the users authenticate to a central sever, but communicate data directly from one user to another.

Whilst the proof rules that we have developed in this thesis make proofs of correctness relatively easy to construct, the amount of required knowledge is still a barrier to wider use. Further, the proofs require lots of small details to be tracked (such as which parts of which messages are equal), which makes them difficult to construct. Therefore, it would be useful to develop a tool that automates the construction of these proofs. Gavin Lowe has developed a prototype of such a tool that is able to construct the correctness proofs for WebAuth automatically. The tool

proceeds by firstly attempting to prove the safety of atoms, along the lines of Definition 3.17. It then applies the authentication proof rules in order to deduce how the strands are related to each other. Currently, the tool requires user input in order to decide which rule to apply. More work is required in order to allow the tool to automatically apply the correct proof rule.

It would be useful to expand the above tool to allow the correctness of protocols that consist of arbitrarily many layers to be analysed. This tool would be composed of two parts. Firstly, the tool would need to be extended to allow high-level channels to be specified and to allow the tool to prove the absence of the required high-level penetrator subpaths. Further, the tool could also be extended to automatically prove the correctness of an arbitrary number of layers by automatically applying the rules inductively.

Given the complexity of the assumptions of Section 5.2, it would be useful to construct a tool that would check if the assumptions are satisfied. In particular, the tool should take as input the list of low-level transport-layer protocols that are being used, and the list of high-level application-layer protocols that are being abstracted, and then check if the various disjointness conditions are satisfied. This will likely require certain domain-specific assumptions to be specified, in order for the disjointness conditions to be satisfied. In fact, it would be helpful if the tool were able to suggest sufficient disjointness conditions for the assumptions to be satisfied. The development of this tool would require a sophisticated way of reasoning about protocols syntactically; this would likely be the most challenging aspect of the research.

It would be useful if the tools that are developed above could each produce a *certificate* of correctness, in the form of a machine-checkable proof. For example, suppose we have a tool that essentially proves the correctness of a application-layer protocol by applying the proof rules of this thesis. One way of producing a certificate is to formalise the high-level strand spaces model and the proof techniques we have developed for it in a theorem prover (e.g. Coq [Tea04] or Isabelle [NWP02]). Then, it should be possible to make the tool output a script that details which proof rules were applied when, which could then be checked by a theorem prover. This approach would be similar to [MCB13] where the authors develop a tool to automatically construct proofs for Isabelle [NWP02] based on a custom protocol security model that supports secrecy and authentication properties.

Bibliography

[ACC07] Alessandro Armando, Roberto Carbone, and Luca Compagna. LTL Model Checking for Security Protocols. In *Computer Security Foundations Symposium*, pages 385–396, 2007.

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *Formal Methods in Security Engineering*, 2008.

[ACG+08] Suzana Andova, Cas J F Cremers, Kristian Gjøsteen, Sjouke Mauw, Stig F Mjølsnes, and Saša Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206(2–4):425–459, 2008.

[BF08] Michele Bugliesi and Riccardo Focardi. Language Based Secure Communication. In *Computer Security Foundations Symposium*, pages 3–16, 2008.

[Bla01] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE*, pages 82–96, 2001.

[Bla09] Bruno Blanchet. Automatic Verification of Correspondences for Security Protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[BLP03] Giampaolo Bella, Cristiano Longo, and Lawrence C Paulson. Verifying Second-Level Security Protocols. In *Theorem Proving in Higher Order Logics*, pages 352–366. 2003.

[BM10] A D Brucker and S A Mödersheim. Integrating Automated and Interactive Protocol Verification - Springer. *Formal Aspects in Security and Trust*, 2010.

[CC10] Stefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *Computer Security Foundations Symposium*, 2010.

[CD08] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2008.

[Cre08a] Cas J F Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. *Computer Aided Verification*, 2008.

[Cre08b] Cas J F Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128, 2008.

[DDMR07] Anupam Datta, Ante Derek, John C Mitchell, and Arnab Roy. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[DGT07] S Doghmi, Joshua Guttman, and F Javier Thayer. Searching for Shapes in Cryptographic Protocols. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2007.

[Dil11] Christopher Dilloway. *On the Specification and Analysis of Secure Transport Layers*. DPhil Thesis, University of Oxford, 2011.

[DL08] Christopher Dilloway and Gavin Lowe. Specifying Secure Transport Channels. In *Computer Security Foundations Symposium*, pages 210–223, 2008.

[DR08] T Dierks and E Rescorla. The TLS Protocol: Version 1.2, 2008.

[DY83] D Dolev and A Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FRH+] Brad Fitzpatrick, David Recordon, Dick Hardt, Johnny Bufu, and Josh Hoyt. OpenID Authentication 2.0.

[GM11] Thomas Groß and Sebastian Mödersheim. Vertical Protocol Composition. In *Computer Security Foundations Symposium*, 2011.

[GRL11] Thomas Gibson-Robinson and Gavin Lowe. Analysing Applications Layered on Unilaterally Authenticating Secure Transport Protocols. In *Formal Aspects in Security and Trust*, 2011.

[GT00] Joshua Guttman and F Javier Thayer. Protocol independence through disjoint encryption. In *Computer Security Foundations Workshop*, pages 24–34, 2000.

[GT02] Joshua Guttman and F Javier Thayer. Authentication Tests and the Structure of Bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[Hoa85] Tony Hoare. *Communicating Sequential Processes*. Prentice Hall International, University of Oxford, 1985.

[Hoy12] Jonathan Hoyland. Analysing The OAuth Protocol. BA Thesis, University of Oxford, 2012.

[JH12] Michael B Jones and Dick Hardt. The OAuth 2.0 Authorization Framework, 2012.

[Kam10] Allaa Kamil. *The Modelling and Analysis of Layered Security Architectures in Strand Spaces*. DPhil Thesis, University of Oxford, 2010.

[KL09] Allaa Kamil and Gavin Lowe. Specifying and Modelling Secure Channels in Strand Spaces. In *Formal Aspects in Security and Trust*, 2009.

[KL10] Allaa Kamil and Gavin Lowe. Understanding Abstractions of Secure Channels. In *Formal Aspects in Security and Trust*, 2010.

[KL11] Allaa Kamil and Gavin Lowe. Analysing TLS in the Strand Spaces Model. *Journal of Computer Security*, 2011.

[KR05] Eldar Kleiner and A W Roscoe. Modelling unbounded parallel sessions of security protocols in CSP. *Journal of Computer Security*, 2005.

[LBLM+04] Paul J Leach, Tim Berners-Lee, Jeffrey C Mogul, Larry Masinter, Roy T Fielding, and James Gettys. Hypertext Transfer Protocol – HTTP/1.1, 2004.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2007.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low97] Gavin Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *CSFW*, pages 18–30, 1997.

[Low98] Gavin Lowe. Casper: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998.

[MCB13] Simon Meier, Cas J F Cremers, and David Basin. Efficient Construction of Machine-Checked Symbolic Protocol Security Proofs. *Journal of Computer Security*, 2013.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[MV09a] Sebastian Mödersheim and Luca Viganò. Secure pseudonymous channels. In *European Symposium on Research in Computer Security*, 2009.

[MV09b] Sebastian Mödersheim and Luca Viganò. The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In *Foundations of Security Analysis and Design V*, pages 166–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[MV11] Sebastian Mödersheim and Luca Viganò. Sufficient Conditions for Vertical Composition of Security Protocols. Technical report, IMM-TR-2011-18, DTU Informatics, 2011.

[NS78] R Needham and M Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 1978.

[NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. Isabelle/HOL: a proof assistant for higher-order logic. *Isabelle/HOL: a proof assistant for higher-order logic*, January 2002.

[Pau98] Lawrence C Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 1998.

[Res00] E Rescorla. HTTP Over TLS, 2000.

[Ros10] Bill Roscoe. *Understanding Concurrent Systems*. Springer-Verlag New York Inc, May 2010.

[SA09] Roland Schemers and Russ Allbery. WebAuth Technical Specification v.3, 2009.

[SMCB12] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *Computer Security Foundations Symposium*, pages 78–94, 2012.

[Tea04] The Coq Development Team. *The Coq proof assistant reference manual*. LogiCal Project, 2004.

[TFHG99] F Javier Thayer Fábrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 1999.

Appendix A

Index of Notation

$t_1 \sqsubseteq_{es} t_2$ Equivalent to $t_1 = \mathit{extract}(t_2, es)$ (Definition 4.2)..

$\mathcal{B} \trianglerighteq \mathcal{B}'$ Denotes that the two bundles are equivalent and $\preceq$ relates no-more regular nodes (Definition 5.5)..

$\mathcal{A}$ Indicates that a channel is authenticated (Definition 3.10)..

$\mathcal{AC}$ Indicates that a channel is both authenticated and confidential..

$\mathcal{A}^c_{app}$ The subterm-closure of $\mathcal{T}^c_{app}$ together with the set of all encryption keys (both the key and its inverse) for terms in $\mathcal{T}^c_{app}$ (Definition 5.1)..

$\mathcal{A}_{\mathcal{P}}$ The set of all terms initially known to the penetrator (Definition 2.1)..

p_d app-extracts n Denotes that the destructive penetrator path p_d extracts the application-layer message sent from n (Definition 4.17)..

$\trianglerighteq_{app}$ $\mathcal{B} \trianglerighteq_{app} \mathcal{B}'$ iff the bundles are application-layer-equivalent and $\preceq$ relates no-more regular application-layer nodes (Definition 7.3)..

$\mathit{appmsg}(t)$ Gives the application-layer message of the high-level term t (Definition 3.2)..

p_c app-packages n Denotes that the constructive penetrator path p_c packages the application-layer message received at n (Definition 4.17)..

$\mathcal{C}$ The set of all channel ends (Definition 3.1)..

$\mathcal{C}$ Indicates that a channel is confidential (Definition 3.9)..

$\mathit{chan}(t)$ Gives the channel of the high-level term t (Definition 3.2)..

Channels The set of all transport-layer channels (Definition 3.1)..

$\mathcal{E}$ The set of all encryptions (Definition 5.2)..

$\mathcal{E}^c_{app}$ The set of encryptions that occur within the application-layer in messages sent along channel c (Definition 5.2)..

$endpoints(st)$ Returns the set of channel endpoints that the regular strand st uses (Definition 3.4)..

$ends(st)$ Returns the set of channel ends that the regular strand st uses (Definition 3.4)..

$\mathcal{EP}$ The set of all extraction paths (Definition 4.2)..

$\mathcal{E}^c_{trpt}$ The set of transport-layer encryptions that enclose application-layer messages sent on channel c (Definition 5.2)..

$\mathcal{E}^c_{trpt\text{-}non\text{-}msg}$ The set of transport-layer encryptions that do not enclose application-layer messages on channel c (Definition 5.2)..

$\mathsf{expath}\,(p_d)$ The extraction path corresponding to the destructive penetrator path p_d (Lemma 4.3)..

$\mathsf{expath}^{\sim}\,(p_c)$ The extraction path corresponding to the constructive penetrator path p_c (Lemma 4.3)..

$\mathsf{expath}^{\mathsf{c}}\,(c)$ The extraction path that extracts the application-layer message from a transport-layer payload term (Assumption 4.5)..

$extract(t, es)$ Extracts the term t using the extraction path es (Definition 4.2)..

$height_{\hat{\mathcal{B}}}(st)$ The $\hat{\mathcal{B}}$-height of the strand st..

$\mathcal{I}$ The set of all endpoints (Definition 3.1)..

$\mathsf{ingredient}\ t_1\ \mathsf{ingredient}\ t_2$ iff t_1 is required in order to construct t_2..

$\mathcal{N}_{payload}$ The set of regular nodes that send application-layer messages (Assumption 4.4)..

$\mathcal{N}_{trpt}$ The set of regular nodes that send application-layer messages over a channel other than $\perp$ (Assumption 4.4)..

$\mathcal{N}_{\perp}$ The set of regular nodes that send application-layer messages over $\perp$ (Assumption 4.4)..

$recipient(t)$ Gives the recipient of the high-level term t (Definition 3.2)..

$\mathcal{S}$ The set of all sequence numbers (Definition 3.1)..

$sender(t)$ Gives the sender of the high-level term t (Definition 3.2)..

$seqno(t)$ Gives the sequence number of the high-level term t (Definition 3.2)..

$\mathcal{T}^c_{app}$ The set of application-layer messages sent along channel c (Definition 5.1)..

$TLS^{C\to S}$ The type of a unilateral TLS channel from client to server (Definition 3.4)..

$TLS^{S\to C}$ The type of a unilateral TLS channel from server to client (Definition 3.4)..

$\mathcal{T}^c_{non\text{-}msg}$ The subterm-closed set of terms that appear in the transport-layer packaging of channel c (Definition 5.1)..

$\mathcal{T}^c_{non\text{-}payload}$ The set of transport-layer non-payload (e.g.handshake terms) terms for channel c (Assumption 4.5)..

$\mathcal{T}^c_{payload}$ The set of transport-layer terms that contain application-layer messages for channel c (Assumption 4.5)..

Appendix B

Index of Assumptions

Assumption 3.3 This requires that there is a format for high-level terms on a particular high-level channel. For example, two high-level terms on a channel c must either both have a sequence number, or neither must have.

Assumption 3.5 Regular agents in high-level bundles must *correctly* use sequence numbers, in that they must send and receive consecutively numbered messages.

Assumption 3.12 All channels in a high-level strand space must be *sensible*, in that if they prohibit a certain combination of strands, they must prohibit all equivalent combinations.

Assumption 3.21 This details the assumptions necessary to prove the correctness of WebAuth.

Assumption 4.4 The set of regular nodes is partitioned between nodes that send and receive transport-layer payload messages, and those that manipulate transport-layer non-payload messages (e.g.handshake terms).

Assumption 4.5 For each low-level channel, this requires the existence of various sets of terms, such as $\mathcal{T}_{payload}$.

Assumption 4.6 This ensures the existence of various functions, such as *sender*, that extract components of low-level transport-layer messages.

Assumption 4.8 This assumption ensures that all low-level channels have disjoint sets of transport-layer payload terms.

Assumption 4.10 This assumption is a low-level analogue of Assumptions 3.3 and 3.5, along with Equation 3.1. In particular, it ensures that low-level transport-layer terms have a well-defined and consistent format.

Assumption 4.15 All channels permit TX and HJRN strands iff they permit equivalent combinations of HJ and RN strands..

Assumption 5.9 Formalises the main statically checkable assumption that Σ satisfies layer-disjoint encryption.

Assumption 5.18 This ensures that all channel types are *sensible* in that whenever a HJ strand is prohibited, then so to must the equivalent LN and FK combination).

Assumption 5.32 Assumes that any innocuous non-expanded subpath in a low-level bundle can be expanded.

Assumption 6.10 The sets of regular strands $Role_k$ and $\widehat{Role}_k$ are well-defined, in that they consist of strands related using an abstraction function.

Assumption 8.26 Regular agents in a high-level strand space use sequence numbers correctly when using the high-level channel $\mathcal{S}$.

Assumption 8.36 Defines the assumptions that are required to prove the username/password channel correct.