On the Protocol Composition Logic PCL

Cas Cremers*
ETH Zurich
Switzerland
cremersc@inf.ethz.ch

ABSTRACT

A recent development in formal security protocol analysis is the Protocol Composition Logic (PCL). We identify a number of problems with this logic as well as with extensions of the logic, as defined in [9, 13, 14, 17, 20, 21]. The identified problems imply strong restrictions on the scope of PCL, and imply that some claimed PCL proofs cannot be proven within the logic, or make use of unsound axioms. This includes the proofs of the CR protocol from [13, 14] and the SSL/TLS and IEEE 802.11i protocols from [20, 21]. Where possible, we propose solutions for these problems.

Categories and Subject Descriptors

C.2.2 [**Computer-communication Networks**]: Network Protocols—*Protocol verification*; F.3 [**Logics and meanings of programs**]: Specifying and Verifying and Reasoning about Programs

General Terms

Security, Theory

Keywords

Security protocol analysis, logic, composition

1. INTRODUCTION

Formally establishing properties of security protocols has been investigated extensively during the last twenty years, but a definite model and a corresponding method for security protocol analysis has remained elusive thus far.

The most successful approaches to security protocol analysis have been focused on tools for finding attacks, which are often based on bounded model checking or constraint solving, e.g. [1, 22]. When such tools find an attack, one can easily verify manually whether or not the attack actually exists on the protocol. Some tools even allow for unbounded verification of protocols, e.g. [4, 7]. If no attack is found with such tools, correctness of the protocol follows. However, this provides little insight into why a protocol is correct.

An alternative approach is to develop a logic for reasoning about security protocols. When a protocol is proven correct in such a logic, the derivation steps can provide insight into the mechanisms that make the protocol work. Despite the obvious promise of such an approach, several attempts have failed, most notably the BAN logic from [6]. One of the stumbling points seems to be to provide a logic that is sound with respect to the complicated semantics of security protocol execution in the presence of an active intruder, and is able to provide concise formal proofs.

A recent attempt to develop such a logic is the Protocol Composition Logic (PCL) from e.g. [14]. This logic has evolved from a protocol model to express protocol composition and refinement, into a model with an associated logic that can be used to formally prove security properties of protocols [9, 13, 17]. Variants of PCL have been applied to many case studies and offer several interesting features. For example, one can reason about security protocols without explicitly reasoning about the (complex) intruder by means of a special kind of invariant reasoning captured by the *honesty rule*. This kind of reasoning also allows the protocol logic to deal with protocol composition and refinement, where proofs can be reused. PCL has been extended with several features in further work, such as an extension for hash functions in [21] that was used for a modular correctness proof of IEEE 802.11i and TLS.

In this paper, we identify a number of problems with PCL as defined in [9, 13, 14, 17, 20, 21]. They have implications for the scope of PCL, a number of claimed formal proofs, and several extensions to the base model. In particular, we show that in contrast with the claims in e.g. the introduction of [14], PCL as defined in [9, 13, 14, 17] cannot be used to prove common authentication properties of protocols that do not include signatures. We show that a number of claimed proofs in PCL cannot be correct because (a) there is no way to establish preceding actions in a thread, and (b) there is no way to express type restrictions in PCL. With respect to existing PCL extensions, we identify two problems: the Diffie-Hellman extension from [9, 13, 14, 17] does not correctly capture the algebraic behaviour of Diffie-Hellman-like protocols, and the extension for hash functions from [20, 21] is not sound. Some of these problems can be resolved by minor modifications to PCL, but other problems

*This work was supported by the Hasler Foundation, under ManCom project 2071.

To appear in the Third ACM Symposium on Information, Computer and Communications Security (ASIACCS'08), March 2008, Tokyo.

require further investigation. Our observations suggest that it is at least required to make changes to existing axioms, to introduce new axioms, and to add a mechanism for a type system.

The purpose of this paper is to identify some of the challenges that need to be addressed in order to make a logic like PCL work. We hope it will contribute to the improvement of PCL, and will lead to a better understanding of some of the pitfalls of designing a compact and usable formal logic for security protocols.

The scope of this paper. The presentation of this paper is inherently difficult, not least because there are a number of different papers on PCL, which vary in notation and technical details. Many ideas were already present in precursors of PCL, e.g. [18,19], but these variants use different concepts than later versions of PCL. These early variants in [18, 19] have no notion of thread (a.k.a. process, run, or role instance), and events are bound to agents. More recent versions of PCL bind events to threads of agents, and therefore distinguish between several threads of the same agent. PCL versions of the latter type can be found in [10–12,16]. Subsequently, [10–12,16] have been claimed to be either subsumed, or revised and extended, by more recent works [9,13,14,17]. Hence we choose here to focus on [9, 13, 14, 17], which contain similar descriptions of PCL. Throughout this paper we write *basic PCL* to refer to [9, 13, 14, 17]. The publications on basic PCL describe the fundamental part of PCL that focusses on authentication. In general, the comments in this paper apply to basic PCL. The comments in Section 4.2 apply only to the extensions found in [20, 21]. Our comments here do not cover the recent extensions to basic PCL for the analysis of secrecy, as found in [24], nor the computational variants of PCL, as found in e.g. [15].

Syntax and page references. In order to pinpoint our observations to specific formulas, we select a specific version of PCL to refer to. We have chosen the most recent description of PCL from 2007 as found in [14]. In particular, we will use [14] as a reference for the syntax of PCL formulas, and to provide specific page references. Hence we use [14] as the reference paper to present the problems with basic PCL from the papers [9, 13, 14, 17].

For the technical details, in particular the formulas, we assume the reader is at least somewhat familiar with one of the papers from [9, 13, 14, 17] or [20, 21]. However, the main points should be clear to readers familiar with formal security protocol analysis.

The remainder of the paper is structured in the following way. We start off by recalling some PCL notation and concepts in Section 2. Then, in Section 3 we discuss problems with the basic definition of PCL. In Section 4 we identify two problems with existing PCL extensions. We conclude in Section 5.

Acknowledgements.

The author would like to thank David Basin, Anupam Datta, Felix Klaedtke, Sjouke Mauw, Simon Meier, John C. Mitchell, Arnab Roy and the anonymous referees for useful discussions and feedback on earlier versions of this paper.

2. PRELIMINARIES

The purpose of this section is to recall some PCL notions that are required to interpret the forthcoming sections. We use the syntax from [14]. This partial summary of PCL is incomplete, and we encourage the reader to use the original papers for reference.

The structure of PCL is as follows: first notation is introduced to define terms, which in turn are used to define protocols. For such protocols, an execution model is defined, assigning to each protocol a set of possible execution histories, called *runs*. Then, a protocol logic is defined in order to reason about (sets of) runs of a protocol. This logic is proven sound with respect to the execution model. This means that if one proves a property in terms of the protocol logic, such as $\mathsf{Receive}(\ldots, m)$, then a similar property should hold for the corresponding set of runs in the execution model, such as "`receive` $\ldots, m$ has occurred in the protocol run". We touch upon these elements below.

Terms. A term system is introduced that contains constants (nonces, names, keys, etc.), variables, and compound terms such as tuples, encryptions, and signatures.

Protocols. In PCL, a protocol Q is defined as a set of roles. Each role $\rho \in Q$ is defined as a list of actions. Examples of possible actions can be found in the *actions* column of Table 1, and correspond respectively to sending terms, receiving terms, generating fresh terms, encryption, decryption, and signature verification. Each role is partitioned into a set of *basic sequences*. A basic sequence BS_i of a role ρ is a contiguous subsequence of ρ that starts with either the first action of ρ or a `receive` action, and of which the other actions are not `receive` actions. The basic sequences intuitively represent the idea that agents only block at `receive` actions, and execute all other actions immediately, allowing one to regard basic sequences as atomic actions in some respects. The notion of basic sequences therefore roughly corresponds to the notion of *step-compression* in other models, e.g. [2]. The basic sequences of a protocol are defined as the union of the basic sequences of each of its roles, and hence each action of a role of a protocol occurs in exactly one of its basic sequences.

Observe that although some versions of PCL have two distinct actions `receive` (for reading a term from the network without parsing) and `match` (for parsing a term and blocking if it is not as expected), these two actions are in many proofs collapsed together into a single `receive` action that receives and matches terms at the same time.

Execution model. An execution model is defined for protocols (in terms of *cords*). Actions are executed by *agents*, with names like $\hat{X}, \hat{Y}$. A subset of these agents, defined as $\mathit{HONEST}(C)$ (where C is the initial configuration of the system), are called the *honest agents*, and execute their protocol roles as expected. The agents may execute each protocol role any number of times; each such instance of a role is called a *thread* (in other formalisms, this notion is known as a strand, or a process, or a run). Threads are usually denoted by symbols such as X, Y, Z: a sequence of actions P executed within a thread X is written as $[P]_X$. The notation $\hat{X}$ is often[1] used to denote the agent executing the

[1]The hat notation $\hat{}$ is used in at least two different interpretations in both [13] and [14]. In some cases X is used to denote a particular thread, and $\hat{X}$ is then interpreted as the agent that executes that thread, as e.g. can be seen in the usage of Y in the **SEC** axiom [14, page 327], replicated here in Section 3.1. Thus, in one interpretation $\hat{}$ can be regarded as a function from a thread to an agent. However, in other cases $\hat{X}$ is used to denote a particular agent, and X is then interpreted as "any thread executed by the agent $\hat{X}$",

Action	Associated term structure	Predicate (in thread X)
`send` $\hat{X}, \hat{Y}, m$		$\mathsf{Send}(X, m)$
`receive` $\hat{X}, \hat{Y}, m$		$\mathsf{Receive}(X, m)$
`new` x		$\mathsf{Gen}(X, x)$
`enc` m, K	$t = ENC_K\{\!\|\, m \,|\!\}$	$\mathsf{Encrypt}(X, t)$
`dec` t, K	$t = ENC_K\{\!\|\, m \,|\!\}$	$\mathsf{Decrypt}(X, t)$
`verify` t, m, K	$t = SIG_K\{\!\|\, m \,|\!\}$	$\mathsf{Verify}(X, t)$
		$\mathsf{Has}(X, m)$
		$\mathsf{Honest}(\hat{X})$
		$\mathsf{Contains}(t, t')$

Table 1: Some examples of PCL actions, action predicates, terms, and their relations. Here $\hat{X}, \hat{Y}$ denote agents, m, x, t, t' denote terms, and X denotes a thread.

thread X. Informally, if the agent $\hat{X}$ is honest, $[P]_X$ is a sequence of actions from a protocol role. The agents that are not part of $HONEST(C)$ can execute so-called intruder roles, in line with the common Dolev-Yao intruder model. In the context of this execution model, the protocol description Q gives rise to a set of *runs* denoted by $\mathsf{Runs}(Q)$. A run is typically denoted by R, and corresponds to a sequence of actions as executed by threads. A run represents a possible execution history of the system.

Protocol logic. For each of the actions that can occur in a run, a corresponding *action predicate* is defined. Some examples are given in the right column of Table 1. The protocol logic is extended with logical connectives, a predicate to reason about the ordering of actions in a run ($<$), as well as predicates to reason about the knowledge of threads.

One of the predicates of the logic is the Honest predicate, which is closely related to the set $HONEST(C)$. For a run R, $\mathsf{Honest}(\hat{X})$ is used to denote that $\hat{X} \in HONEST(C)$ and "all threads of $\hat{X}$ are in a 'pausing' state in R" [14, page 325]. This is a technicality[2] that is needed to make the honesty rule (which is not described here) work.

One predicate of interest for this paper is the $\mathsf{Contains}$ predicate, which is used to reason about the relation between two terms. In particular, $\mathsf{Contains}(t_1, t_2)$ is defined by means of the subterm relation $\subseteq$ in [14, page 323], stating that t_1 contains t_2 as a subterm. The subterm relation $\subseteq$ is never formally defined. Here we assume that the subterm relation is defined syntactically[3]. In particular, we assume that a signature such as $SIG_{\hat{X}}\{\!\|\, m \,|\!\}$ contains m, i.e. $\mathsf{Contains}(SIG_{\hat{X}}\{\!\|\, m \,|\!\}, m)$ holds. Note that this assumption only plays a role in the construction of a particular counterexample in Section 3.3, and does not influence any of our general observations.

Using the protocol logic, one aims to establish properties of all runs of a protocol. A run R of a protocol Q that satisfies a property ϕ is denoted as $Q, R \models \phi$. If all runs of a protocol Q satisfy ϕ, we write $Q \models \phi$. If a formula ϕ is provable using the logic by using any PCL axioms except the honesty rule, we write $\vdash \phi$, which expresses that ϕ holds for all protocols. For formulas provable using the axioms and the honesty rule for a protocol Q, we write $\vdash_Q \phi$, which expresses that ϕ holds for the protocol Q.

In the remainder of this paper we use the following convention: All formulas that are numbered are ours. All unnumbered formulas, including the named axioms, are copied from PCL papers, in which case the source paper and page number is given.

3. PROBLEMS WITH BASIC PCL

In this section, we address two problems with basic PCL as defined in [9, 13, 14, 17]. First, we identify in Section 3.1 a strong restriction on the scope of basic PCL. Then, we identify two missing proof mechanisms that seem to be necessary to prove simple protocols correct in Sections 3.2 and 3.3.

3.1 Authentication properties only provable for signature protocols

In basic PCL [9, 13, 14, 17], the possibility of proving authentication properties is limited to protocols that use signatures.

Proving an authentication property such as aliveness or agreement requires a proof of existence of actions of another agent. In basic PCL, there is only one axiom that allows for a conclusion of that type. The precondition of this axiom can only be met by protocols that use signatures. As a consequence, if a protocol does not use signatures, the existence of a communication partner cannot be proven within the logic.

This is surprising, as PCL was "initially designed as a logic of authentication" [14, page 313], and it is stated in the abstract of the paper that "PCL [...] has been applied to a number of industry standards including SSL/TLS, IEEE 802.11i [...]" [14, page 311, abstract]. These protocols consist of many subprotocols that do not rely only on signatures, but also rely on symmetric key cryptography and symmetric MACs.

The problem occurs for all authentication properties that imply the existence of a communication partner. This includes *aliveness* or any form of *agreement* from [23], *matching conversations* from [3], or *synchronisation* from [8]. The matching conversations property is the authentication property used within basic PCL [14, page 331]. All these properties are of the form:

$$\phi(X) \supset \exists Y.\psi(Y) \tag{1}$$

where typically, $\phi(X)$ denotes that thread X executes a role that authenticates another role. Such a property states that if a thread X executes actions of a certain role (e.g. all actions of an initiator role), then there exists a thread Y that has executed some actions of another role (e.g. the first two actions of a responder role), and possibly some further condition holds. This is captured in $\psi(Y)$. Note that both

as in the honesty rule **HON** [14, page 329], and the **VER** axiom [14, pp. 329–330], replicated here in Section 3.1. In this second interpretation, X is a variable that ranges over all threads executed by the agent $\hat{X}$, and $\hat{X}$ effectively expresses a domain restriction on X.

[2]The Honest predicate serves within the honesty rule as an encoding of the atomic nature of basic sequences.

[3]Observe that if we would alternatively assume that the subterm relation involves only tuple projection, as seems to be suggested by the **CON** axiom [13, page 444], i.e. $t \subseteq t' \equiv (t = t') \vee (\exists t''.t' = (t, t'') \vee t' = (t'', t))$, then e.g. the **P2** and **VER** axioms in [13, 14] are unsound, because respectively the fresh value and the signature might be sent as part of an encrypted term, and decrypted by an agent that knows the key.

the weak and strong authentication properties in the example of [14, page 331] belong to this class of authentication properties.

In order to prove such a property, it is required to prove the existence of a thread. Examination of all axioms of the logic reveals that only one axiom allows for the conclusion of the existence of a thread, which is the **VER** axiom [14, page 327]:

$$\textbf{VER}\ \ \mathsf{Honest}(\hat{X}) \wedge \mathsf{Verify}(Y, SIG_{\hat{X}}\{\!|\, x \,|\!\}) \wedge \hat{X} \neq \hat{Y} \supset \exists X.\mathsf{Send}(X, m) \wedge \mathsf{Contains}(m, SIG_{\hat{Y}}\{\!|\, x \,|\!\})$$

Because this axiom has the signature verification predicate Verify in the precondition, it can only be used for signature protocols. Hence there is no way to prove the existence of another thread for protocols without signatures.

Other comments regarding non-signature protocols.

Whilst introducing new axioms for establishing thread existence for non-signature protocols is non-trivial, there is a related problem with the inconsistent use of symmetric/asymmetric cryptography. In basic PCL, there is only one type of encryption operator, and only a single key set, in the sense that the rules for encryption and decryption do not distinguish between different types of keys. This suggests that either only symmetric, or only asymmetric encryption is supported.

The definitions of the reaction rules [14, page 318], the action formulas [14, page 324] and the Has predicate [14, pp.324–325] all indicate that one encrypts and decrypts a message using the same key, e.g.:

$$\mathsf{Has}_{i+1}(A, ENC_K\{\!|\, m \,|\!\}) \quad \text{if} \quad \mathsf{Has}_i(A, m) \text{ and } \mathsf{Has}_i(A, K)$$
$$\mathsf{Has}_{i+1}(A, m) \quad \text{if} \quad \mathsf{Has}_i(A, ENC_K\{\!|\, m \,|\!\}) \text{ and } \mathsf{Has}_i(A, K)$$

The same assumption, that one encrypts and decrypts with the same key, can be found in axiom **AR3** [14, page 327]. Without explaining the full details of the notation of this axiom, we want to point out that the key K is used as the key to decrypt a message encrypted with K:

$$\textbf{AR3}\quad \mathsf{a}(x)\ [y := \mathsf{dec}\ x, K]_X\ \mathsf{a}(ENC_K\{\!|\, y \,|\!\})$$

However, the idea that symmetric encryption is intended seems to be contradicted by the **SEC** axiom [14, page 327], which states that there is only one agent that can decrypt a message encrypted with a key, along the lines of asymmetric encryption:

$$\textbf{SEC}\ \ \mathsf{Honest}(\hat{X}) \wedge \mathsf{Decrypt}(Y, ENC_{\hat{X}}\{\!|\, x \,|\!\}) \supset (\hat{Y} = \hat{X})$$

Combined with the definition of the Has predicate, one is lead to conclude that the only one who can create the encrypted message that occurs in the **SEC** axiom, is the agent that can decrypt it.

3.1.1 Implications

In order to prove any authentication property of the form of (1) for a protocol that does not use signatures, one needs to introduce consistent machinery for symmetric and asymmetric cryptography and several new axioms.

In basic PCL, protocols like Needham-Schroeder-Lowe[4] and many key establishment protocols cannot be modeled, and even if they could be, no authentication proofs could be given for them. Similarly, it is impossible to use basic PCL as defined in [9, 13, 14, 17] to prove the authentication properties of the protocols "SSL/TLS, IEEE 802.11i" [14, page 311, abstract]. In order to prove authentication of these protocols, one is required to introduce additional axioms.

[4]Observe that in a precursor of PCL in [19] there is a proof of Needham-Schroeder-Lowe, but this logic has only asymmetric cryptography, and has no notion of threads or processes.

3.1.2 Resolving the problem

The problem can be split into three subparts. First, symmetric and asymmetric cryptography must be dealt with consistently. Second, basic PCL must be extended with axioms that enable proving authentication properties (existence of a thread) for symmetric cryptography. Third, resolving the problem for public-key encryption protocols (which includes many key agreement protocols, and the well-known Needham-Schroeder-Lowe protocol) requires the introduction of additional theory. We first address the two easier problems before turning to public-key encryption.

Dealing consistently with symmetric and asymmetric cryptography.

The first requirement for resolving this problem is to split symmetric and asymmetric encryption either by having two types of encrypt/decrypt actions as in e.g. [1], or by splitting the key set as in e.g. [7]. Either choice impacts the action sets, the action formulas, the Has predicate, and requires the addition of further **ARx** axioms and an alternative for the **SEC** axiom. Most of the additions are trivial, but introduce additional complexity into the logic and possibly also the execution model.

Proving authentication for protocols using symmetric encryption or keyed hashes.

The axiom for signature protocols deals with the simplest possible case of authentication by cryptography: the signature of an honest agent can be used immediately to conclude that there exists a thread executed by this agent. If that agent differs from any agent executing the currently known threads, this implies the existence of another thread. This conclusion is based on the fact that only one agent has the private key needed to perform the signature.

For symmetric encryption and keyed hashes there is usually a notion of symmetry: in most cases, two agents share a symmetric key. Thus, if a symmetric key K shared by two honest agents $\hat{X}$ and $\hat{Y}$, occurs in a message of a thread X, there are two candidates for the agent who created the message. If we can exclude that the message was generated in thread X executed by $\hat{X}$, we can derive the existence of another thread Y that must have created it.

The authors of PCL have explored a variant of this route for e.g. the extensions needed for the keyed-hash based protocols in [21]. We discuss the merits of those extensions in detail in Section 4.2.

Proving authentication for public-key encryption protocols.

If we assume that basic PCL is extended to consistently deal with public-key encryption, the authentication properties still cannot be proven, as we lack an axiom for establishing thread existence as pointed out previously. Contrary to signatures or symmetric-key cryptography, there is no easy solution with an axiom based on honest agents only, because the public keys are also known to the intruder. If a message

encrypted with the public key of an honest agent $\hat{X}$ occurs, no conclusion can be drawn about the sender or creator of the message. Thus we must also consider the possibility that the message was sent by the adversary.

A first step towards a solution would be to introduce an axiom in the protocol logic, along the lines of Lemma 2.3 [14, page 321] of the execution model. In fact, such an axiom was present in a precursor of basic PCL. In [19, page 701] one can find axiom **AR1** which, when recast in the basic PCL model, would read

$$\textbf{AR1} \quad [\texttt{receive}\ t]_X \exists Z.\mathsf{Send}(Z,t) \qquad \text{(This axiom is not in basic PCL)} \quad (2)$$

A second axiom that might be adapted for proving these properties is the **SRC** axiom in [19, page 703].

Unfortunately, for either of these axioms, authentication proofs would require either further machinery to prove that Z is executed by an honest agent, or explicit reasoning about the intruder, as the sender thread Z need not be executed by an honest agent. This type of reasoning is not supported by basic PCL, and would require a significant amount of additional machinery.

3.2 No means to establish preceding actions in a thread

By the definition of the PCL execution model, threads of honest agents start at the beginning of a role description, and always execute the actions in the order given by the role. Thus, if one can establish that a thread of an honest agent has executed the nth action of a role, it should also be possible to conclude a similar result about preceding actions within the logic: in particular, one should be able to conclude that the preceding $n-1$ actions have also occurred in the same thread. However, there is no mechanism in the logic of basic PCL [9, 13, 14, 17] to draw such a conclusion.

Note that one can use the honesty rule to prove from, e.g., a `send` action that a `receive` action must have occurred previously, but only if both actions occur within the same basic sequence. However, if one wants to prove that given a basic sequence, an action has occurred from another basic sequence, there are no rules to enable this type of reasoning.

3.2.1 Implications

The consequence of not having a means of establishing preceding actions is that some claimed proofs do not seem to be correct. For example, observe the initiator role of the Q_{CR} protocol [14, page 315]. In order to prove invariants for the protocol using the honesty rule, one has to prove that the invariants hold for all basic sequences of the protocol. The initiator role consists of two basic sequences BS_1 and BS_2 [14, page 335]:

$$\mathrm{BS}_1 \equiv [\ \texttt{new}\ m;\ \texttt{send}\ \hat{X},\hat{Y},m;\ ]_X$$
$$\mathrm{BS}_2 \equiv [\ \texttt{receive}\ \hat{Y},\hat{X},y,s;\ \texttt{verify}\ s,(y,m,\hat{X}),\hat{Y};\ r := \texttt{sign}\ (y,m,\hat{Y}),\hat{X};\ \texttt{send}\ \hat{X},\hat{Y},r;\ ]_X$$

In this protocol, the basic sequence BS_1 defines m as a generated value, which determines the semantics of m in BS_2. Observe that there is no way to tell if m was received or generated from just investigating BS_2.

In order to show how this makes certain proofs impossible in basic PCL, consider a protocol Q', that contains a role ρ consisting of the basic sequences BS_1' followed by BS_2, where BS_1' is defined as:

$$\mathrm{BS}_1' \equiv [\ \texttt{receive}\ ENC_K\{\!|\, m \,|\!\};\ \texttt{send}\ \hat{X},\hat{Y},m;\ ]_X \quad (3)$$

Thus the protocol Q' shares the basic sequence BS_2 with Q_{CR}, but in Q', m is a received valued as opposed to a generated value.

Now, in order to prove invariant γ_1 [14, page 334] for basic sequence BS_2, we must prove that m is either generated, or received earlier as the main message, by the same thread that sends out the signature:

$$\gamma_1 \equiv \big(\mathsf{Send}(Y,t) \wedge \mathsf{Contains}(t, SIG_{\hat{Y}}\{\!|\, y,m,\hat{X} \,|\!\})\big) \supset \Big(\big(\mathsf{Gen}(Y,m) \vee (\mathsf{Receive}(Y,(\hat{X},\hat{Y},m)) < \mathsf{Send}(Y,(\hat{Y},\hat{X},y,SIG_{\hat{Y}}\{\!|\, y,m,\hat{X} \,|\!\})))\big)\Big)$$

In order to prove this invariant with respect to basic sequence BS_2, one needs to prove that m is generated in the thread Y. However, there is no mechanism to reason about any preceding actions, and thus it has to be dealt with in the same way for both protocols Q_{CR} and Q'. One must therefore also consider the case m was *first received*, and not generated, in a previous basic sequence as part of a bigger term, as happens in Q'. In fact, the invariant is false for protocol Q'.

Thus, in order to prove the invariant for Q_{CR}, we must be able to distinguish between the different semantics of BS_2 within the context of Q_{CR} and Q'. Such reasoning is not supported by basic PCL.

Without the ability to reason about preceding actions, the protocol descriptions are effectively cut up into independent basic sequences, which can then only be dealt with as separate protocols. This is evident from the formulation of the honesty rule [14, page 329], in which the reference to the protocol Q is only used for the definition of the basic sequences. Thus one is required to create much stronger proofs than would be strictly necessary. Put differently, in basic PCL one has to reason with an over approximation of protocol execution, in which basic sequences occur in no fixed order in a thread.

This phenomenon can also be observed in the following example. Let P be a protocol, and let P_1 be the protocol defined as P extended with a role consisting of the basic sequences $\mathrm{BS};\mathrm{BS}';\mathrm{BS}''$. Similarly, let P_2 be defined as P extended with a role $\mathrm{BS}'';\mathrm{BS}';\mathrm{BS}$, i.e., a role with the same basic sequences as the role in P_1, but composed in a different order. Then, any invariant γ that can be proven for protocol P_1 using the honesty rule, must also hold for P_2. In fact, the proof is identical. Conversely, any invariant that holds for P_1 but not for P_2, cannot be proven using the honesty rule. This puts a strong restriction on the type of invariants provable in basic PCL, because the structure (and hence the properties) of both protocols can be very different.

These observations lead to the following result:

THEOREM 1. *If basic PCL as defined in [9, 13, 14, 17] is sound, then the authentication properties of the challenge response protocol* CR *from [14] cannot be proven by using basic PCL without using protocol composition rules.*

A detailed proof of this theorem is given in Appendix A. Below we only provide a sketch of the proof.

PROOF SKETCH. We consider a specially crafted protocol CR′, which has the same basic sequences as the protocol CR. Because a proof in PCL that does not use composition rules only refers to basic sequences of a protocol, any such proof of $\vdash_{Q_{\mathrm{CR}}} \phi$ is also a proof of $\vdash_{Q_{\mathrm{CR}'}} \phi$. We show that the weak authentication formula ϕ of the initiator is false for $Q_{\mathrm{CR}'}$. Hence, there exists no proof of $\vdash_{Q_{\mathrm{CR}}} \phi$ that does not use composition rules. □

Ultimately, this problem seems to be a side effect of the weak link between the protocol description Q and the actions of honest agents in a run $R \in \mathsf{Runs}(Q)$ in the protocol logic. In basic PCL, the only way to make the link between a protocol description and the actions of honest agents, is by means of defining an invariant and then proving it with the honesty rule. This can be seen from inspecting the protocol logic rules: the only occurrences of a protocol name (e.g. Q) are in the honesty rule and the composition rules. As the honesty rule only reasons about isolated basic sequences of the protocol, the relations among the basic sequences of the protocol are inevitably lost.

3.2.2 Resolving the problem

Based on the semantics of PCL, it should be possible to extend the logic with a "precedence" inference rule, that would allow one to infer that given the nth action of a role ρ occurs in a thread X, also the $(n-1)$th action of the same role must have previously occurred in the same thread. Such an inference rule would allow for establishing preceding actions that must have occurred from the existence of other actions. This is particularly useful for proving invariants for basic sequences.

In general one could extend basic PCL with additional mechanisms to enable reasoning about the relation between protocol descriptions and the actions of honest agents in a run. However, the current weak link between the two has one major advantage: it eases compositional proofs. For example, because no constraints are put on any other (i.e. non-protocol) actions of honest agents, the sequential composition of protocols allows for re-use of proofs of invariants for basic sequences. Thus, one must be careful when introducing such links and introduce only links between the protocol and the actions of honest agent, such that the links are invariant under e.g. sequential composition. A "precedence" inference rule of the type sketched above should meet this condition.

3.3 No formal type system

It is mentioned that in PCL, "We assume enough typing to distinguish the keys K from the principals $\hat{A}$, the nonces n and so on." [14, page 316]. However, there are no constructs in PCL that allow for formal reasoning about the type of variables.

As a consequence, some claimed proofs of invariants are not correct within the logic, as they are only true under type assumptions, that cannot be expressed in PCL.

3.3.1 Implications

Many protocols have properties in a typed model, that do not hold for an untyped model. In particular, some protocols are correct in a typed model, but are vulnerable to so-called type flaw attacks in an untyped model. Such attacks exploit for example that an agent could mistake an encrypted session key for an encrypted nonce, and sends the key unencrypted in a next step, revealing it to the intruder. It is therefore often easier to prove properties for the typed model, but this requires that the logic supports drawing conclusions based on types. This is not possible in basic PCL.

As an example, we show that invariant γ_1 from [14, page 334] (reproduced in this paper in Section 3.2.1) is false when variables are untyped. The invariant γ_1 states that if a thread (of an honest agent) sends a message that contains a signature term S, with a subterm m, then either

1. m was generated in the thread Y, or
2. the thread Y executes a `receive` of m, and later a `send` of the message tuple (y, S).

Now consider the basic sequence BS_3 from [14, page 335]:

$$\mathrm{BS}_3 \equiv [\ \texttt{receive}\ \hat{X}, \hat{Y}, x;\ \texttt{new}\ n;\ r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y};\ \texttt{send}\ \hat{Y}, \hat{X}, n, r;\ ]_Y$$

This basic sequence corresponds to the responder receiving an unknown value, supposedly a nonce, and sending back a signed version that includes a freshly generated nonce.

In order to show the invariant is false, consider a thread Y' that is executed by agent $\hat{Y}$. If we assume that $x = SIG_{\hat{Y}}\{\!|\, y, m, \hat{X} \,|\!\}$, where m is generated by thread Y', the invariant is violated: we have that $x = SIG_{\hat{Y}}\{\!|\, y, m, \hat{X} \,|\!\} \supset \mathsf{Contains}(x, m)$ and by substitution in BS_3 and the $\mathsf{Contains}$ predicate, we find that $\mathsf{Contains}(r, m)$. As a result, the message sent at the end of BS_3 of a thread Y ($Y' \neq Y$) will contain m. However, m is neither generated by this thread nor is it the exact term that was received. Thus the invariant γ_1 does not hold for basic sequence BS_3. Hence the example authentication proof of the CR protocol in [14] is incorrect.

3.3.2 Resolving the problem

The model can be extended with typing information for the variables, and an axiom could be introduced that captures the fact that variables are only instantiated by their typing information. This would not introduce much new machinery, but requires additional reasoning steps in most of the proofs.

4. PROBLEMS WITH PCL EXTENSIONS

In this Section we discuss two additional mechanisms for PCL, in particular the extension for Diffie-Hellman exponentials as found in [9, 13, 14, 17] and the extension for hash functions from [20, 21].

4.1 Diffie-Hellman exponentials

In basic PCL [9,13,14,17], axioms are provided for reasoning about Diffie-Hellman (DH) exponentials. To that end, the logic is extended with four additional axioms and some changes are made to the language and execution model.

4.1.1 Capturing Diffie-Hellman behaviour

Below we recall the three elements of the DH extension (execution model, logic, proof system) and discuss their implications.

DH extension of the language and execution model.

The programming language and execution model are extended [14, page 354] with constructs $g(a)$ and $h(a, b)$, representing respectively $g^a \bmod p$ and $(g^a \bmod p)^b \bmod p$. In

the PCL papers, the mod p and brackets are usually omitted, resulting in the abbreviations g^a and g^{ab}.

This extension does not reflect the actual algebraic properties of the exponential. For Diffie-Hellman, the crucial property is that $g^{ab} = g^{ba}$. This equation must be included in the execution model: if it is not, the following sequence of actions (denoted by their action predicates), which is perfectly valid DH-type behaviour, does not correspond to a valid execution in PCL:

$$\mathsf{Send}(X, h(a,b)) < \mathsf{Receive}(Y, h(b,a)) \tag{4}$$

Just extending the Has predicate in the logic is not sufficient, as the equivalence still has no counterpart in the execution model of PCL.

DH extension of the protocol logic.

In basic PCL, the predicate $\mathsf{Fresh}(X, x)$ holds for any x generated by thread X (captured by $\mathsf{Gen}(X, x)$), as long as x is not sent as part of another term. The protocol logic is extended with an additional rule for the Fresh predicate, stating that $\mathsf{Fresh}(X, g(x))$ is true if and only if $\mathsf{Fresh}(X, x)$ is true, and Has is extended in [14, page 354] by

$$\mathsf{Has}(X,a) \wedge \mathsf{Has}(X, g(b)) \supset \mathsf{Has}(X, h(a,b)) \wedge \mathsf{Has}(X, h(b,a))$$

This rule is intended to capture the Diffie-Hellman equivalence relation. It is not sufficient, even with the addition of further axioms, as we show below.

DH extension of the proof system.

The proof system is extended in [14, page 354, Table A.1] by a definition and four new axioms. We reproduce them here:

Define **Computes (DH)**

$$\mathsf{Computes}(X, g^{ab}) \equiv \Big(\big(\mathsf{Has}(X,a) \wedge \mathsf{Has}(X, g^b)\big) \vee \big(\mathsf{Has}(X,b) \wedge \mathsf{Has}(X, g^a)\big) \Big)$$

This definition captures the intuition that a thread can compute the value of g^{ab} if and only if it has the required components. Note that for a thread that just has g^{ab} but not a or b, this predicate is false, and therefore we have that $\mathsf{Has}(X, g^{ab})$ does *not* imply $\mathsf{Computes}(X, g^{ab})$.

Note that the definition of Computes suggests that $g^{ab} \neq g^{ba}$, because of the explicit listing of both sides of the disjunction. If the equality would hold, one could just define $\mathsf{Computes}(X, g^{ab}) \equiv \mathsf{Has}(X,a) \wedge \mathsf{Has}(X, g^b)$ to achieve the same result.

Using the definition of Computes, the first axiom is given as:

DH1 $\quad \mathsf{Computes}(X, g^{ab}) \supset \mathsf{Has}(X, g^{ab})$

and corresponds to the extension of the Has predicate, which can be seen by unfolding the definition of Computes. The second axiom states

DH2 $\quad \mathsf{Has}(X, g^{ab}) \supset \Big(\mathsf{Computes}(X, g^{ab}) \vee \exists m. \big(\mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m, g^{ab})\big) \Big)$

This axiom captures the notion that one can only possess such a term g^{ab} by computing it from its parts, or receiving it. Effectively this restricts any agent (including the intruder) from knowing a term of the form g^{ab} at the start of each run, but it also excludes that g^{ab} is used as a parameter for a protocol.

For completeness, we also give the remaining DH axioms

DH3 $\quad \big(\mathsf{Receive}(X,m) \wedge \mathsf{Contains}(m, g^{ab})\big) \supset \exists Y, m'. \big(\mathsf{Computes}(Y, g^{ab}) \wedge \mathsf{Send}(Y, m') \wedge \mathsf{Contains}(m', g^{ab})\big)$

DH4 $\quad \mathsf{Fresh}(X,a) \supset \mathsf{Fresh}(X, g^a)$

which capture the assumption that g^{ab} must have been either computed or received (but was not given as a parameter). The fourth axiom echoes the extension of the Fresh predicate.

Summarizing, even with the extension of the Has predicate and the additional axioms, the behaviour of the equivalence for Diffie-Hellman is not captured in PCL, for two reasons. First, with respect to the execution model, the sequence of actions represented by Formula (4) cannot be enabled in the execution model by just changing the protocol logic. Second, we observe that given $\mathsf{Has}(X, h(a,b))$, the protocol logic does not allow us to conclude that $\mathsf{Has}(X, h(b,a))$ as one would expect (especially in the case where $\neg\mathsf{Computes}(X, h(a,b))$). Hence, the essential equivalence for any Diffie-Hellman type protocol is not captured in the execution model, nor in the protocol logic.

4.1.2 Implications

The Diffie-Hellman extension does not correctly capture the algebraic properties of Diffie-Hellman exponentials. As a result, certain possible behaviours of DH-like protocols are not considered within the logic and its execution model. The extension is therefore not a faithful representation of DH-like protocols.

4.1.3 Resolving the problem

The essential feature to be captured is the equality $g^{ab} = g^{ba}$. If this is introduced at the term level, e.g. by having the equality $h(a,b) = h(b,a)$ in the term system, this solves at the same time the problem in the execution model, protocol logic and proof system. Some further modifications to the axioms are required, and the current proofs have to be modified to take the term equality into account. Alternatively, one might consider introducing generalizations of the Diffie-Hellman assumption, as e.g. in [5].

4.2 Keyed hashes

In [20] and [21] basic PCL is extended and applied to a case study of protocols that do not rely solely on signatures. As observed previously, such an application requires additional axioms. In particular, to prove authentication, we need to introduce an axiom that allows us to conclude the existence of a thread.

Appendix A of [21] gives such additional axioms and definitions for keyed hash functions: one definition and four hashing axioms. We reproduce them here for convenience.

Define **Computes (HASH)** :

$$\mathsf{Computes}(X, HASH_K(a)) \equiv \mathsf{Has}(X,K) \wedge \mathsf{Has}(X,a)$$

As we will see in the following, the intention of this predicate seems to be to express which thread computed the hashed value from its components. However, this intuition is not correctly captured by the definition: a typical use pattern of a keyed hash would be to provide an integrity check for a message m, as in

$$\hat{X} \rightarrow \hat{Y} \ : \ m, HASH_K(m) \ , \tag{5}$$

where K is a key shared between $\hat{X}$ and $\hat{Y}$. In the typical use pattern, the $HASH_K(m)$ is received by an agent which at that point can construct the message m himself. In this use case, the message hash is received with (or after) the message, and is then used to verify the integrity of the message.

Assume we have that m (or some subterm of it) is freshly generated by X, and we have that $\mathsf{Computes}(X, HASH_K(m))$ holds. However, we can also prove that once a thread Y of the recipient $\hat{Y}$ receives the message, Y not only has K, but also m. Thus the predicate $\mathsf{Computes}(Y, HASH_K(m))$ holds as well. Put differently, we have that for a typical use case, $\mathsf{Computes}(\ldots)$ holds for both the thread who creates as well as the thread who receives. In particular, this is also true for the protocols under consideration in [21]. The protocols in the paper assume typing information (no hash mistaken for a nonce) and receive the hashed values at a point where they know the hash components.

Let K be a key shared between $\hat{X}$ and $\hat{Y}$. Then, we can prove[5] the following property (invariant) for the protocols in [21]:

$$\mathsf{Honest}(\hat{X}) \wedge \mathsf{Has}(X, HASH_K(m)) \supset \mathsf{Computes}(X, HASH_K(m)) \tag{6}$$

Intuitively one can see that an honest agent either creates the hash by having the components, or receives it. In this last case, where an honest agent receives the hash, the message integrity is verified using the hash, which is possible only because the recipient also has both the key K and the message m. We will use this result in our discussion of the **HASH4** axiom later on.

The first axiom in [21, Appendix A] that uses the definition of $\mathsf{Computes}()$ is the following:

HASH1

$$\mathsf{Computes}(X, HASH_K(x)) \supset \mathsf{Has}(X, x) \wedge \mathsf{Has}(X, K)$$

If we unfold the definition of $\mathsf{Computes}$ in this axiom, it reduces to $\phi \supset \phi$. The second hash axiom states the following:

HASH2

$$\mathsf{Computes}(X, HASH_K(x)) \ \supset \ \mathsf{Has}(X, HASH_K(x))$$

This axiom informally states that whoever computes the hash value also has it. If we again unfold the definition of $\mathsf{Computes}$, we can see that this works as an extension of the closure properties of the Has predicate (as defined in [14]). Effectively, it introduces a new term structure $f_x(y)$ to the PCL syntax, and expresses the one-way property of the hash function.

[5]Note that this cannot be proven using only basic PCL, but it can be proven using PCL combined with the meta-reasoning required to capture the assumption that variables are typed, as pointed out in Section 3.3.

The third hash function axiom is

HASH3 $\mathsf{Receive}(X, HASH_K(x)) \supset$
$\exists Y.\mathsf{Computes}(Y, HASH_K(x)) \wedge \mathsf{Send}(Y, HASH_K(x))$.

This axiom is not sound. Consider the following protocol in Alice and Bob style notation, where m is a freshly generated nonce of the initiator, and K' is a symmetrical key shared by the initiator and responder:

$$\begin{aligned} Init \rightarrow Resp \ &: \ ENC_{K'}\{\!|\, HASH_K(m) \,|\!\} \\ Resp \rightarrow Init \ &: \ HASH_K(m) \end{aligned} \tag{7}$$

In a normal run of this protocol, the initiator sends the hash as part of a bigger term, but does not send m. Thus the responder cannot compute the hash itself, but simply decrypts the message, and sends the hash back. Thus, the preconditions are fulfilled by an initiator thread of this protocol, but the postcondition does not hold: only the initiator thread can compute it, but it does not send it out in the required form. (Observe that contrary to the protocols in [21], this protocol in (7) does not satisfy the typical usage pattern, and therefore Formula (6) does not hold here.)

The fourth axiom is

HASH4 $\mathsf{Has}(X, HASH_K(x)) \supset \mathsf{Computes}(X, HASH_K(x))$
$\vee \exists Y, m.\ \mathsf{Computes}(Y, HASH_K(x)) \wedge \mathsf{Send}(Y, m) \wedge \mathsf{Contains}(m, HASH_K(x))$.

This axiom seems to express: if someone has the hash value, she computed the hash herself, or somebody computed it and sent it (possibly as a subterm).

With a suitable restriction on the intruder knowledge (the intruder should not know some $HASH_K(x)$ initially without knowing the components) this axiom can be proven sound. However, there is a problem with the applications of the axiom in the proofs of [14]. Each time this axiom is applied in proofs in the paper, one assumes honesty of $\hat{X}$ and $\hat{Y}$, and that K is the key shared between them. Thus we can rewrite the application of axiom **HASH4** using Formula (6), as in the following. First we unfold the definition of the axiom

$$\mathsf{Honest}(\hat{X}) \wedge \mathbf{HASH4} = \mathsf{Honest}(\hat{X}) \wedge \big(\mathsf{Has}(X, HASH_K(x)) \supset \mathsf{Computes}(X, HASH_K(x)) \vee \Phi\big) \ , \tag{8}$$

where Φ is used to denote the right-hand disjunct of the axiom **HASH4**, starting from the existential quantifier. Now we use Formula (6) to get

$$\mathsf{Honest}(\hat{X}) \wedge \mathbf{HASH4} = \mathsf{Honest}(\hat{X}) \ . \tag{9}$$

In other words, if $\hat{X}$ is assumed to be honest when applying axiom **HASH4**, the left-hand disjunct of the conclusion is always true (rendering the right-hand disjunct inconsequential). Because the right-hand disjunct of the conclusion contains the required thread existence, but is rendered useless, the axiom cannot be used here for proving authentication properties, based on our observations in Section 3.1.

4.2.1 Implications

The authentication proofs of the four-way handshake and group key handshake protocols in [20, 21] are not correct in their current form. The reason for this is that these protocols

do not contain signatures, and based on the observations in Section 3.1, any authentication proofs for such protocols must rely on newly introduced axioms. In this case, the only candidates are the axioms **HASH3** and **HASH4**. As shown above, **HASH3** is not sound, and for the protocols in [20,21], **HASH4** cannot be used to prove the existence of a thread. Hence the authentication proofs of the handshakes are incorrect, and therefore also the compositional proof of authentication of IEEE 802.11i is incorrect.

4.2.2 Resolving the problem

The properties of a keyed hash function like the ones occurring here are similar to the properties of symmetric-key cryptography. Thus, once sufficient reasoning infrastructure is in place for symmetric-key cryptography, one can devise a straightforward definition of a non-keyed hash function. Combining these two elements should lead to a natural definition and extension of the logic for keyed hashes.

Alternatively, it might be possible to reinstate rules that were used in older works, as e.g. found in [11, page 15].

5. CONCLUSIONS

In this paper, we have first shown that basic PCL as defined in [9,13,14,17] cannot be used to prove authentication properties of protocols without signatures. We consider this to be a strong limitation on the scope of basic PCL. Next, we have pointed out two reasoning tools that are missing in basic PCL: reasoning about preceding actions of a role within a thread, and the lack of a formal type system. With respect to PCL extensions, we have shown that the PCL Diffie-Hellman extension from [9,13,14,17] does not capture the algebraic behaviour of Diffie-Hellman protocols correctly in the execution model and protocol logic. Finally, we have shown that the extension for protocols with keyed hash functions in [20,21] is not sound.

It follows that some claimed PCL proofs cannot be proven within the basic logic as defined in [9,13,14,17], or make use of unsound axioms. These proofs include the authentication proofs of the CR protocol from [13, 14] and the SSL/TLS and IEEE 802.11i protocols from [20,21].

In the papers on PCL the proofs of the invariants are often not given. This lack of proof details for the invariants is surprising, as the invariants themselves are the difficult part. Furthermore, many of the proofs seem impossible to be completed in PCL without resorting to meta-reasoning.

Some of the problems identified here can be solved easily, but for some problems more work is required. For example, straightforward solutions include adding a formal type system and adding a mechanism to reason about earlier actions. Other problems, such as establishing thread existence of honest agents for protocols based on public-key cryptography, do not seem to be solvable by straightforward fixes, and suggest that more extensive modifications to PCL are required.

It remains to be seen whether formal proofs in such a modified version of PCL can be concise.

6. REFERENCES

[1] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, L. Cuellar, P. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer-Verlag, 2005.

[2] D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Springer-Verlag, 2003.

[3] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag.

[4] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society, 2001.

[5] E. Bresson, Y. Lakhnech, L. Mazaré, and B. Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In A. J. Menezes, editor, *Proc. of Crypto '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 482–499. Springer-Verlag, August 2007.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[7] C. Cremers. *Scyther - Semantics and Verification of Security Protocols*. PhD thesis, Computer Science Department, Eindhoven University of Technology, November 2006.

[8] C. Cremers, S. Mauw, and E. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, 2006.

[9] A. Datta. *Security Analysis of Network Protocols: Compositional Reasoning and Complexity-theoretic Foundations*. PhD thesis, Computer Science Department, Stanford University, September 2005.

[10] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 109–125. IEEE Computer Society, 2003.

[11] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 30–45, Washington, DC, USA, June 2004. IEEE Computer Society.

[12] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. Secure protocol composition. *Electron. Notes Theor. Comput. Sci.*, 83, 2004. Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics.

[13] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[14] A. Datta, A. Derek, J. Mitchell, and A. Roy. Protocol Composition Logic (PCL). *Electron. Notes Theor. Comput. Sci.*, 172:311–358, 2007. Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin. Editors: L. Cardelli, M. Fiore, and G.

Winskel.
[15] A. Datta, A. Derek, J. Mitchell, and B. Warinschi. Computationally sound compositional logic for key exchange protocols. *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, 0:321–334, 2006.
[16] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (extended abstract). In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23, New York, NY, USA, 2003. ACM.
[17] A. Derek. *Formal Analysis of Security Protocols: Protocol Composition Logic*. PhD thesis, Computer Science Department, Stanford University, December 2006.
[18] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 241–272, 2001.
[19] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:667–721, 2003.
[20] C. He. *Analysis of Security Protocols for Wireless Networks*. PhD thesis, Department of Electrical Engineering, Stanford University, December 2005.
[21] C. He, M. Sundararajan, A. Datta, A. Derek, and J. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 2–15. ACM Press, 2005.
[22] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 18–30. IEEE Computer Society, 1997.
[23] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society, 1997.
[24] A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. Secrecy analysis in protocol composition logic. In M. Okada and I. Satoh, editors, *Proceedings of 11th Annual Asian Computing Science Conference*, Lecture Notes in Computer Science, December 2006. Preliminary version.

APPENDIX

A. PROOF OF THEOREM 1

In this section we give a detailed proof of Theorem 1 from Section 3.2 of this paper. The proof is by contradiction: We show that if a proof could be given for the authentication of the CR protocol, it would also be a proof for a similar protocol CR′. Next, we show that the protocol CR′ does not satisfy the authentication formula.

LEMMA 1. *Let ϕ be any PCL formula, and let Q and Q' be PCL protocols such that the basic sequences of Q are equal to the basic sequences of Q'. Then, a proof of $\vdash_Q \phi$ in PCL as defined in [9, 13, 14, 17] that does not use any composition rules is also a proof of $\vdash_{Q'} \phi$.*

PROOF. By inspection of the axioms and inference rules of the logic, the only references to a protocol are in the honesty rule, through the set of basic sequences of all roles of the protocol, and the composition rules. Hence a proof for a protocol that does not use the composition rules, is a proof for all protocols with the same basic sequences. □

Next we recall the definition of the basic sequences of the CR protocol as defined on [14, page 335]. For the initiator role we have:

$$\mathrm{BS}_1 \equiv [\ \texttt{new}\ m;\ \texttt{send}\ \hat{X}, \hat{Y}, m;\ ]_X$$

$$\mathrm{BS}_2 \equiv [\ \texttt{receive}\ \hat{Y}, \hat{X}, y, s;\ \texttt{verify}\ s, (y, m, \hat{X}), \hat{Y};\ r := \texttt{sign}\ (y, m, \hat{Y}), \hat{X};\ \texttt{send}\ \hat{X}, \hat{Y}, r;\ ]_X$$

and for the responder role we have:

$$\mathrm{BS}_3 \equiv [\ \texttt{receive}\ \hat{X}, \hat{Y}, x;\ \texttt{new}\ n;\ r := \texttt{sign}\ (n, x, \hat{X}), \hat{Y};\ \texttt{send}\ \hat{Y}, \hat{X}, n, r;\ ]_Y$$

$$\mathrm{BS}_4 \equiv [\ \texttt{receive}\ \hat{X}, \hat{Y}, t;\ \texttt{verify}\ t, (n, x, \hat{Y}), \hat{X};\ ]_Y$$

DEFINITION 1. *Let $Q_{\mathrm{CR}'}$ consist of three roles ρ_1, ρ_2, ρ_3. The first two roles are identical to the two roles of the* CR *protocol. Let ρ_1 be the initiator role of* CR*: BS_1 followed by BS_2. Let ρ_2 be the responder role of* CR*: BS_3 followed by BS_4. Now, let ρ_3 be defined as BS_2.*

Note that as ρ_3 consists of BS_2 only, we have that m is not generated nor read within this role, and must be considered a parameter of this role. Furthermore, as $Q_{\mathrm{CR}'}$ is defined by the basic sequences of the CR protocol, we have that The protocols Q_{CR} and $Q_{\mathrm{CR}'}$ have the same basic sequences, i.e. $\mathrm{BS}_1, \mathrm{BS}_2, \mathrm{BS}_3, \mathrm{BS}_4$.

Consider the weak authentication property $\phi_{\mathrm{weak-auth}}$ of the initiator from [14, page 331]:

$$\phi_{\mathrm{weak-auth}} \equiv \exists Y.\big(\mathsf{Receive}(Y, (\hat{X}, \hat{Y}, m)) < \mathsf{Send}(Y, (\hat{Y}, \hat{X}, y, SIG_{\hat{Y}}(y, m, \hat{X})))\big)$$

which should hold in the following context [14, page 331]:

$$\vdash_{Q_{CR}} \top[\ \mathrm{Init}_{\mathrm{CR}}\ ]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{\mathrm{weak-auth}}$$

Observe that because the strong authentication from [14] implies weak authentication, our results also hold for the strong authentication property.

We refer to the modal formula to be proven as $\Phi_{\mathrm{weak-auth}}$:

$$\Phi_{\mathrm{weak-auth}} \equiv \top[\ \mathrm{Init}_{\mathrm{CR}}\ ]_X \mathsf{Honest}(\hat{Y}) \wedge \hat{Y} \neq \hat{X} \supset \phi_{\mathrm{weak-auth}} \quad (10)$$

Informally, this formula states that if the initiator role of the CR protocol (which consists of the basic sequences BS_1 and BS_2) is executed by an agent $\hat{X}$, who tries to communicate with another agent $\hat{Y}$ that is honest, then $\phi_{\mathrm{weak-auth}}$ holds: there exists a thread of $\hat{Y}$ in which a particular `send` and `receive` have occurred.

We construct a run $R_{\mathrm{no-auth}}$ of the $Q_{\mathrm{CR}'}$ protocol, described by $R_{\mathrm{no-auth}} = R_a; R_b; R_c; R_d$. The run consists of a sequence of four basic sequences BS_1, BS_3, BS_2, BS_2. We define this run below by representing the four subsequences in terms of the basic sequence notation. We use this notation to clearly show the relation of the run to the basic sequences, even though it is not the standard PCL syntax for describing runs (one would use syntax from the cord space semantics).

DEFINITION 2. *Let $R_{\mathrm{no-auth}}$ be a run of* CR′ *that contains the sequences below in order, and no other actions of*

honest agents. Note that all variables are considered to be instantiated in the description of $R_{\mathrm{no-auth}}$*. In particular, let* $\hat{X}, \hat{Y}$ *be honest agents, where* $\hat{X} \neq \hat{Y}$*, let* X, X' *be threads executed by* $\hat{X}$ *and* Y *a thread executed by* $\hat{Y}$*, and let* n, m *be nonces.*

$$
\begin{aligned}
R_{a(\rho_1,\mathrm{BS}_1)}: &\ [\ \mathtt{new}\ m;\ \mathtt{send}\ \hat{X},\hat{Y},m;\ ]_X \\
R_{b(\rho_2,\mathrm{BS}_3)}: &\ [\ \mathtt{receive}\ \hat{Y},\hat{X},m;\ \mathtt{new}\ n; \\
&\qquad r := \mathtt{sign}\ (n,m,\hat{Y}),\hat{X};\ \mathtt{send}\ \hat{X},\hat{Y},n,r;\ ]_{X'} \\
R_{c(\rho_3,\mathrm{BS}_2)}: &\ [\ \mathtt{receive}\ \hat{X},\hat{Y},n,s;\ \mathtt{verify}\ s,(n,m,\hat{Y}),\hat{X}; \\
&\qquad r := \mathtt{sign}\ (n,m,\hat{X}),\hat{Y};\ \mathtt{send}\ \hat{Y},\hat{X},r;\ ]_Y \\
R_{d(\rho_1,\mathrm{BS}_2)}: &\ [\ \mathtt{receive}\ \hat{Y},\hat{X},n,s;\ \mathtt{verify}\ s,(n,m,\hat{X}),\hat{Y}; \\
&\qquad r := \mathtt{sign}\ (n,m,\hat{Y}),\hat{X};\ \mathtt{send}\ \hat{X},\hat{Y},r;\ ]_X
\end{aligned}
\tag{11}
$$

In the run $R_{\mathrm{no-auth}}$, the following happens. R_a and R_d together form a normal thread of an agent $\hat{X}$ that executes the role ρ_1, which corresponds to the initiator role of the CR protocol. R_b corresponds to a normal execution of the ρ_2 role, analogous with the responder role of the CR protocol, executed by $\hat{X}$. The difference between CR and CR′ is exploited in R_c. Whereas in the context of CR, m is the previously generated nonce of the initiator role, we have that in the context of CR′, m is a parameter of the role ρ_3. This allows for the instantiation of the parameter[6] with the previously generated nonce m. All messages that are received in R_a, R_b, R_c, R_d can be constructed on the basis of previously sent messages by using tupling and projection cords. Hence, we have that $R_{\mathrm{no-auth}}$ is a run of $Q_{\mathrm{CR}'}$.

LEMMA 2. $Q_{\mathrm{CR}'} \not\models \Phi_{\mathrm{weak-auth}}$.

PROOF. We first show that the run $R_{\mathrm{no-auth}}$ does not satisfy $\Phi_{\mathrm{weak-auth}}$. Within $R_{\mathrm{no-auth}}$, R_a and R_d together form ρ_1, which is identical to the initiator role of CR. Both agents $\hat{X} \neq \hat{Y}$ are honest. Therefore the precondition of the implication of Formula (10) is met, and thus the postcondition $\phi_{\mathrm{weak-auth}}$ should also hold. $\phi_{\mathrm{weak-auth}}$ expresses that $\hat{Y}$ has executed a thread of the responder role. This is not the case: The only actions performed by $\hat{Y}$ in $R_{\mathrm{no-auth}}$ are those of the role ρ_3, which consists of the basic sequence BS_2, which is not part of the responder role of CR. In particular, the weak authentication property requires that $\hat{Y}$ received the term m, which does not happen in $R_{\mathrm{no-auth}}$.

Because the run $R_{\mathrm{no-auth}}$ is a run of $Q_{\mathrm{CR}'}$, $\Phi_{\mathrm{weak-auth}}$ does not hold for all runs of $Q_{\mathrm{CR}'}$. □

We are now able to prove Theorem 1 from Section 3.2.

PROOF OF THEOREM 1. We prove the theorem by contradiction. Assume there exists a proof of $\vdash_{Q_{\mathrm{CR}}} \Phi_{\mathrm{weak-auth}}$ in PCL as defined in [9,13,14,17] that does not use the composition rules. Then, by Lemma 1 and Definition 1 this is also a proof of $\vdash_{Q_{\mathrm{CR}'}} \Phi_{\mathrm{weak-auth}}$. By soundness, we have that $\models_{Q_{\mathrm{CR}'}} \Phi_{\mathrm{weak-auth}}$. This contradicts Lemma 2. □

[6]In PCL, this parameter mechanism is used to facilitate sequential composition of protocols.