Automated Proof of Authentication Protocols in a Logic of Events

Mark Bickford (Cornell University and ATC-NY)

Abstract

Using the language of event orderings and event classes, and using a type of atoms to represent nonces, keys, signatures, and ciphertexts, we give an axiomatization of a theory in which authentication protocols can be formally defined and strong authentication properties proven. This theory is inspired by *PCL*, the protocol composition logic defined by Datta, Derek, Mitchell, and Roy.

We developed a general purpose *tactic* (in the NuPrl theorem prover), and applied it to automatically prove that several protocols satisfy a strong authentication property. Several unexpected subtleties exposed in this development are addressed with new concepts—*legal protocols*, and a *fresh signature criterion*—and reasoning that makes use of a well-founded causal ordering on events.

This work shows that proofs in a logic like *PCL* can be automated, provides a new and possibly simpler axiomatization for a theory of authentication, and addresses some issues raised in a critique of *PCL*.

1 Introduction

For several years we have been building a general theory for constructing and reasoning about distributed systems using a logic of events [4]. We have successfully used the theory to *extract* algorithms, such as consensus algorithms, from constructive proofs that their specifications are *realizable*. In order to extend this approach to algorithms and protocols that involve concepts of security, we added *atoms* to our theory as an abstraction of *unguessable* secrets[3]. John Mitchell suggested a connection with the Protocol Composition Logic (*PCL*)[7], an axiomatic logic for security proofs, including authentication protocols, designed to

> Carry out proofs by induction only over steps of the protocol without requiring explicit reasoning over possible actions of a malicious attacker.

A proof of an authentication protocol in such a theory shows that the protocol is immune to "man-in-the-middle" attacks with one, two, or any number of attackers. The only attacks on the protocol are those that invalidate one or more of the axioms of the logical theory, so the assumptions needed to establish the authentication properties are explicit in the axioms of the theory.

Mitchell also conjectured that *PCL* could be embedded in a more general theory that allows reasoning about the relative ordering of protocol actions—in which actions at a single agent are totally ordered but actions at different agents are partially ordered. This is exactly the structure of our event logic, which is based on Lamport's concept of causal ordering on events [9], and we have also developed a general concept called an *event class*, which is very helpful in structuring logical theories. In 2009, visiting fellow Meihua Xiao prompted us to begin a true implementation of our version of *PCL*, and he also brought to our attention the critique by Cremers [6] of one version of *PCL*, the one presented in [7]. We were able to use the language of events, event classes, and atoms to implement a theory for authentication, which we name Authentication Event Logic, and to fully automate the proofs of strong authentication properties for some protocols.

The formalization of Authentication Event Logic has made some "foundational" clarifications by finding some unexpected subtleties and bringing to the surface things that are not so clear in other treatments. In particular, while developing an automated proof method, a *tactic*, for proving strong authentication properties of protocols that succeeds on variants of the Challenge/Response (*CR*) protocol,

we discovered several surprising subtleties that led us to new concepts—*legal protocols*, and a *fresh signature criterion*. We checked that the issues raised by Cremers in [6] are addressed in this formal implementation.

2 Overview

Section 3 is a self-contained description of Authentication Event Logic. This part of the theory does not mention protocols, but it is the general framework for the semantics of authentication protocols. It provides a language and axioms for reasoning about partially ordered events and the information associated with these events. Special classes of events model encryption, decryption, nonces, etc., and atoms are used to represent their associated information.

In section 4 we formally define protocols and the authentication properties we want them to have. *PCL* uses a dynamic logic to express properties of protocols, but we express a protocol as one kind of constraint on event orderings, and an authentication property as another kind of constraint. The goal is then to prove (automatically) that the first constraint (the protocol) implies the second constraint (the authentication property). Authentication properties have the following form: if a sequence of events at one location form an instance of the protocol then there must exist events at another location that form a *matching conversation*. This means that corresponding sends and receives in the two sequences have the same information and each send causally precedes the corresponding receive. When such a property holds, an agent who completes the protocol "knows" that a given remote agent has received and sent matching messages in the proper order—in particular an adversary can not disrupt the synchronization of the protocol by replaying messages from past instances.

In section 5 we begin the formal proof of the Challenge/Response (CR) protocol defined in [7]. Our aim is to understand the general procedure used to prove the matching conversation property so that we can automate the procedure. We encounter our first obstacle when attempting to prove the causal ordering relation between the sends and receives. To overcome this obstacle we define the *legal* protocols and prove the *nonce release lemma* for legal protocols. This allows use to completely automate the proof of a variant of the CR-protocol that includes one extra nonce.

In section 6 we formulate a *fresh signature criterion* and prove a *signature release lemma* for protocols that satisfy the criterion. This allows us to completely automate the proof of a second variant CR_2 that does not use an extra nonce, but differs from the original CR in order to satisfy the fresh signature criterion[1].

In section 7 we show how our formal theory addresses the critique by Cremers of [7].

3 Formal Theory

We need only three types of primitive values from which to build Authentication Event Logic. Two of these, booleans and identifiers, which we write $\mathbb{B}$ and Id are standard. The third type, $Atom$, provides a crucial simplifying abstraction.

3.1 Atoms

The atom type, $Atom$, and associated proof rules is described in [3]. Our authentication theory depends on only part of the theory of atoms and we give a brief, informal, overview of what we need.

[1] A longer version of this paper, available on the NuPrl website discusses the proof tactics needed to prove the CR_2-protocol, why reasoning about the well-founded causal ordering is needed in some cases, and how the proof is sped up by running the tactics in parallel.

Intuitively, members of type $Atom$ are *urelements* that have no structure and cannot be generated. This intuition is formalized by providing only a single primitive computation on atoms—an equality test, and adding a *permutation rule* that says that any provable judgment remains provable after a one-to-one renaming of its atoms. Stuart Allen [1] has proved the consistency of the permutation rule relative to NuPrl computational type theory [2].

In addition to the permutation rule, the theory of atoms introduces a proposition which we write as $(t\colon T \parallel a)$ and read as "term t of type T is independent of atom a". This means that there is a term t' that does not mention atom a and that represents the same value in type T as t does. So, for example, any term t of type $\mathbb{Z}$—even one like $(\lambda x.\lambda y.y+1)\ a\ 3$—is independent of any atom, because its normal form—for the example, 4—contains no atom. The rules for proving $(t\colon T \parallel a)$, given in [3], are simple and easy to automate with tactics. When the type T is clear from context we write just $(t \parallel a)$.

A consequence of these rules is that if a function $f \in \mathbb{Z} \to Atom$ can generate an atom a, say $f(0) = a$, then $\neg(f \parallel a)$. This means that the atom a is mentioned in the "code" for function f.

Our authentication theory does not explicitly use the permutation rule but depends on the concept of independence. We will use atoms to represent everything we wish to regard as *unguessable*—nonces, signatures, ciphertexts, and encryption keys. We associate information with *events* and $\neg(info(e) \parallel a)$ asserts that the information associated with event e contains atom a.

Authentication protocols exchange messages that contain tuples of data such as nonces, signatures, and names. It turns out that the type

$$Data \equiv_{def} Tree(Id + Atom)$$

of finite, binary trees with identifiers or atoms at the leaves, suffices to represent all the messages and plaintexts we need. We use capital letters A, B, ... for identifiers and lower case a, b, ... for atoms, so a term like $\langle a, A\rangle$ is in type $Data$. We abbreviate $\langle x, \langle y, z\rangle\rangle$ as $\langle x, y, z\rangle$.

For some types, T, there may be no computable test for independence $(t\colon T \parallel a)$, but for well-behaved types like $Data$, independence is computable. We define a function $atms : Data \to Atom\ List$ that computes, from a term of type $Data$, the list of atoms occurring at its leaves. Using the rules for independence we can prove

$$\forall d\colon Data.\ \forall a\colon Atom.\ \neg(d \parallel a) \Leftrightarrow a \in atms(d)$$

3.2 Event Orderings

From any formal model of distributed computing one can define the *runs* of a distributed system and can identify, within a run, those points where information is transferred. We call such points, e, *events*, and the information transferred at e the *primitive information*, $info(e)$, associated with the event. The events are points in "space-time" since each event occurs at some *location*—an abstraction of the process, thread, or agent, at which the event occurs. If the information transfer is an atomic operation, then the events at a single location do not overlap in time, so they will be totally ordered. However information is transferred—message passing or writing to shared memory—there is a natural *causal-ordering* on the set of events, the ordering first considered by Lamport [9].

This allows us to define an *event-ordering*, a structure, $\langle E,\ loc,\ <,\ info\rangle$, in which the causal ordering $<$ is a transitive relation on E that is well-founded, and locally-finite (each event has only finitely many predecessors). The events at a given location are totally ordered by $<$, and $info(e)$ (usually the message delivered to $loc(e)$ when the event occurred) is the primitive information associated with event e. In the authentication theory, the location $loc(e)$ of event e is the agent at which it occurs, so $loc(e) = A$ for some identifer A.

We abbreviate $(e' < e\ \wedge\ loc(e') = loc(e))$ by $e' <_{loc} e$.

$New : EClass(Atom)$
$Send, Rcv : EClass(Data)$
$Encrypt, Decrypt : EClass(Data \times Key \times Atom)$
$Sign, Verify : EClass(Data \times Id \times Atom)$

Figure 1: Event classes of the authentication theory

3.3 Event Classes

Our formal theory of authentication uses the language of event-orderings and another key concept—event classes. We describe protocols by classifying the events in the protocol. In authentication protocols there are send, receive, nonce, sign, verify, encrypt and decrypt events. Events in each class have associated information, and the type of this information depends on the class of the event. For example, a nonce event e will be a member of class New and its associated information $New(e)$ will have type $Atom$ because it is the nonce chosen at event e. A send or receive event e' is a member of class $Send$ or Rcv and its associated information, $Send(e')$ or $Rcv(e')$ is the message sent or received at event e' and has type $Data$.

In general, an *event class* X of type T (a member of type $EClass(T)$) is a function on events in an event ordering that partitions the events into two sets, $E(X)$ and $E - E(X)$, and assigns a value $X(e)$ of type T to events $e \in E(X)$.

To relate events in a class to atoms, we define

$$X(e)\ \mathbf{has}\ a \equiv_{def} (e \in E(X)\ \wedge\ \neg(X(e){:}\ T \parallel a))$$

This says that e is an X-event whose associated information is not independent of atom a, so it "has" the atom.

The formal authentication theory is based on the seven special event classes listed, with their types, in figure 1. We have already discussed the classes New, $Send$, and Rcv. Note that classes $Send$ and Rcv share the same type. The remaining four classes consist of pairs $Sign$, $Verify$ and $Encrypt$, $Decrypt$ that also share types.

The information associated with an event e in class $Sign$ or $Verify$ has type $Data \times Id \times Atom$ and is the triple, $\langle signed(e), signer(e), signature(e)\rangle$. An event $e \in E(Sign)$ with information $Sign(e) = \langle x, A, s\rangle$ is an event where agent A signs plaintext x to generate[2] signature s. If A is an honest agent, then $loc(e) = A$ (because honest A does not release its private key). An event $e' \in E(Verify)$ with the same information $Verify(e') = \langle x, A, s\rangle$ is an event where agent $B = loc(e')$ successfully verifies that s is the signature of A on plaintext x.

Similarly, the information associated with an event e in class $Encrypt$ or $Decrypt$ has type $Data \times Key \times Atom$ and is the triple, $\langle encrypted(e), key(e), ciphertext(e)\rangle$. An event $e \in E(Encrypt)$ with information $Encrypt(e) = \langle x, k, c\rangle$ is an event where agent $A = loc(e)$ encrypts plaintext x using key k to generate ciphertext c. Agent A can be any location that *has* the key k and the information in x. Our theory defines (as the negation of independence) only the concept of *having an atom*, so Authentication Event Logic constrains only the *atoms* that an agent may have, but these include nonces, private keys, signatures, and ciphertexts. An event $e' \in E(Decrypt)$ with the information $Decrypt(e') = \langle x, k', c\rangle$ is an event where agent $B = loc(e')$ successfully decrypts ciphertext c using key k' to produce plaintext x. We will assert, in $AxiomD$, that this can occur only when c was generated in an encryption event with a *matching key*.

[2]In our logic, atoms can not be generated so each agent must take them from its private store of atoms. This is not reality, but rather an abstraction that models the assumption that signatures, ciphertexts, and nonces are unguessable.

$Honest : Id \to \mathbb{B}$
$MatchingKeys : Key \to Key \to \mathbb{B}$
$PrivKey : Id \to Atom$

Figure 2: Additional operators of the authentication theory

$$
\begin{aligned}
&\forall A, B : Id.\ \forall k, k' : Key.\ \forall a\colon Atom \\
&MatchingKeys(k; k') \Leftrightarrow MatchingKeys(k'; k)\ \wedge \\
&MatchingKeys(Symm(a); k) \Leftrightarrow k = Symm(a)\ \wedge \\
&MatchingKeys(PrivKey(A); k) \Leftrightarrow k = A\ \wedge \\
&MatchingKeys(A; k) \Leftrightarrow k = PrivKey(A)\ \wedge \\
&PrivKey(A) = PrivKey(B) \Leftrightarrow A = B
\end{aligned}
$$

Figure 3: *Axiom*K

3.4 Key axiom

We represent unguessable keys—symmetric keys or private keys—as atoms, and public keys as identifiers. Private keys and symmetric keys are different, so the type Key is

$$Key \equiv_{def} Id + Atom + Atom$$

In addition to the seven event classes in figure 1 Authentication Event Logic needs three additional functions listed in figure 2. The function $PrivKey$ assigns an atom to each agent (and we abuse notation slightly and call this also a key). The function $MatchingKeys$ provides a relation on keys. Our theory includes several axioms that we will discuss one by one, beginning with *Axiom*K, in figure 3.

It says that matching keys is a symmetric relation, that a symmetric key $Symm(a)$ matches only itself, and that the private key assigned to agent A matches only the public key for A (in the current theory this is defined to be the identifier A itself). Also, no two agents have the same private key.

3.5 Causal axioms

Three axioms that we call $AxiomR$, $AxiomV$, and $AxiomD$ relate events in classes Rcv, $Verify$, and $Decrypt$ to corresponding, causally earlier, events in classes $Send$, $Sign$, and $Encrypt$. $AxiomR$ and $AxiomV$ are similar and say that for any receive or verify event there must be a causally prior send or sign event with the same associated information.

$$
\begin{aligned}
AxiomR: \quad & \forall e\colon E(Rcv).\ \exists e'\colon E(Send). \\
& (e' < e) \wedge\ Rcv(e) = Send(e') \\
AxiomV: \quad & \forall e\colon E(Verify).\ \exists e'\colon E(Sign). \\
& (e' < e) \wedge\ Verify(e) = Sign(e')
\end{aligned}
$$

$AxiomD$ is similar except that for a decrypt event, the prior encrypt event has the same associated information except for the key, which, rather than being the same is a matching key.

$$\begin{aligned} &AxiomD : \forall e : E(Decrypt).\ \exists e' : E(Encrypt).\\ &e' < e\ \wedge\ DEMatch(e, e')\\ &DEMatch(e, e') \equiv_{def} plaintext(e) = plaintext(e')\\ &\qquad\qquad \wedge\ ciphertext(e) = ciphertext(e')\\ &\qquad\qquad \wedge\ MatchingKeys(key(e); key(e')) \end{aligned}$$

3.6 Disjointness axioms

The axiom, $ActionsDisjoint$, simply says that an event in one of the seven special classes is not in any of the other special classes. A second disjointness axiom, $NoncesCiphersAndKeysDisjoint$, says that a nonce is not the same atom as an agent's private key, a signature, or a ciphertext. And, similarly, private keys, signatures, and ciphertexts are disjoint. The formal statements are obvious, so we omit them to save space.

One of these assumptions may deserve comment. A signature may be an encryption of a cryptographic hash of a plaintext, while a ciphertext is an encryption of a plaintext. Thus, as long as the hash of a well-formed member of type $Data$ is not a well-formed member of type $Data$ a signature will not be equal to a ciphertext.

3.7 Honesty axiom

Our theory includes a function $Honest\colon Id \to \mathbb{B}$ that allows us to express assumptions about honest agents. In particular, honest agents do not release their private keys, so sign events with an honest signer; and encryption or decryption events that use the private key of an honest agent must occur at that agent. We call this axiom $AxiomS$ (because it includes the properties of honest signers).

$$\begin{aligned} &AxiomS :\ \forall A\colon Id.\ \forall s\colon E(Sign).\\ &\qquad \forall e\colon E(Encrypt).\ \forall d\colon E(Decrypt).\\ &Honest(A) \Rightarrow\\ &\{signer(s) = A \Rightarrow (loc(s) = A) \wedge\\ &\ key(e) = PrivateKey(A) \Rightarrow (loc(e) = A) \wedge\\ &\ key(d) = PrivateKey(A) \Rightarrow (loc(d) = A)\} \end{aligned}$$

3.8 Flow relation

The final axiom of Authentication Event Logic concerns the causal ordering between events that contain nonces. This is the most complex axiom, and to state it we need some auxiliary definitions.

The type Act contains the events in any of the seven special classes—we call these *actions*. The relation $(e\ \mathbf{has}\ a)$ is true when action e has atom a. Its definition has the seven obvious cases:

$$\begin{aligned} &e\ \mathbf{has}\ a \equiv_{def}\\ &(e \in E(New)\ \wedge\ New(e)\ \mathbf{has}\ a)\ \vee\\ &(e \in E(Send)\ \wedge\ Send(e)\ \mathbf{has}\ a)\ \vee\ \ldots \end{aligned}$$

We define the *flow relation* $e_1 \xrightarrow{a} e_2$ to mean that atom a flows from action e_1 to action e_2. This can happen only in limited ways; either the actions e_1 and e_2 are at the same location, or there are intervening

send and receive events that send atom a "in the clear", or atom a is in the plaintext of an encryption event, and the ciphertext flows to a matching decryption event. The formal, recursive definition of the flow relation is

$$\begin{array}{l} e_1 \xrightarrow{a} e_2 =_{rec} \\ (e_1 \ \mathbf{has}\ a \ \wedge \ e_2 \ \mathbf{has}\ a \ \wedge \ e_1 \leq_{loc} e_2) \\ \qquad\qquad \vee \\ (\exists s\colon E(Send).\ \exists r\colon E(Rcv).\ e_1 \leq s < r \leq e_2 \\ \quad \wedge\ Send(s) = Rcv(r) \ \wedge \ e_1 \xrightarrow{a} s \ \wedge \ r \xrightarrow{a} e_2) \\ \qquad\qquad \vee \\ (\exists e\colon E(Encrypt).\ \exists d\colon E(Decrypt). \\ \ e_1 < e < d \leq e_2 \ \wedge \ DEMatch(d,e) \ \wedge \\ \ key(d) \neq Symm(a) \ \wedge \\ \ e_1 \xrightarrow{a} e \ \wedge \ e \xrightarrow{ciphertext(e)} d \ \wedge \ d \xrightarrow{a} e_2) \end{array}$$

The restriction $key(d) \neq Symm(a)$ is included in the last case because if atom a is encrypted using itself as a key it does not constitute a flow because it can not be decrypted unless the key a is first obtained some other way.

Some consequences of the flow relation are proved by induction

Lemma 1. *If* $e_1 \xrightarrow{a} e_2$ *then* $e_1 \leq e_2$ *and* e_2 **has** a*.*

We also make use of another relation that follows from the flow relation. We say that actions are related by the relation, $\rightsquigarrow$, when the first action is an encryption and the second action *has* the ciphertext of the first.

$$e' \rightsquigarrow e \equiv_{def} e' \in Encrypt \wedge e \ \mathbf{has}\ ciphertext(e')$$

An action e *potentially has* an atom a if it is in the transitive closure under $\rightsquigarrow$ of an event that has a. We write this $e\ \mathbf{has}^*\ a$ and the definition is

$$e\ \mathbf{has}^*\ a \equiv_{def} \exists e' : E.\ (e'\ \mathbf{has}\ a) \wedge (e' \ \rightsquigarrow^* \ e)$$

A send event s that potentially has atom a *releases* the atom because an agent or group of agents that receives the sent message and has all the necessary decryption keys could get the atom. If event e_1 at location A generates a nonce n and an event e_2 has n, then e_1 must causally precede e_2; and if e_2 takes place at a location other than A, it must be preceded by an event at A that sends n or an encrypted version of n. To express this, we define the $release$ relation:

$$\begin{array}{l} release(n, e_1, e_2) \equiv_{def} \\ e_1 \leq_{loc} e_2 \vee \\ \exists s\colon E(Send).\ (e_1 <_{loc} s < e_2) \wedge s\ \mathbf{has}^*\ n \end{array}$$

Lemma 2. *If* $e_1 \xrightarrow{a} e_2$ *then* $release(a, e_1, e_2)$*.*

3.9 Nonce axiom

The assertion about nonces, one part of the axiom we call $AxiomF$ (the *flow* property), is

$$\begin{array}{rl} AxiomF_1: & \forall e_1\colon E(New).\ \forall e_2 : E. \\ & e_2\ \mathbf{has}\ New(e_1) \Rightarrow e_1 \xrightarrow{New(e_1)} e_2 \end{array}$$

This part of $AxiomF$ implies that nonces are associated with unique events:

Lemma 3 (unique nonces)**.** *If* $e_1, e_2 \in E(New)$ *and* $New(e_1) = New(e_2)$ *then* $e_1 = e_2$.

Proof. e_1 **has** $New(e_2)$ so by $AxiomF_1$ and lemma 1, $e_2 \leq e_1$. Similarly, $e_1 \leq e_2$. Therefore, $e_1 = e_2$. □

The two other parts of $AxiomF$ assert a similar relation between signatures and ciphertexts and events that have them. The difference is that we do not assume that signatures and encryptions are always associated with unique events, so if an action has a signature or ciphertext we can only infer that for some sign or encrypt action with the same information, the flow relation holds:

$$
\begin{array}{l}
AxiomF_2 : \forall e_1 \colon E(Sign).\forall e_2 \colon E. \\
e_2 \ \mathbf{has}\ signature(e_1) \Rightarrow \\
\exists e' : E(Sign).\ Sign(e') = Sign(e_1) \\
\qquad \wedge \ \ e' \xrightarrow{signature(e_1)} e_2
\end{array}
$$

$$
\begin{array}{l}
AxiomF_3 : \forall e_1 \colon E(Encrypt).\forall e_2 \colon E. \\
e_2 \ \mathbf{has}\ ciphertext(e_1) \Rightarrow \\
\exists e' : E(Encrypt).\ Encrypt(e') = Encrypt(e_1) \\
\qquad \wedge \ \ e' \xrightarrow{ciphertext(e_1)} e_2
\end{array}
$$

3.10 Authentication Event Logic

To recap, Authentication Event Logic, the authentication theory in which we work, is an event ordering $\langle E,\ loc,\ <,\ info\rangle$, that satisfies the generic axioms—transitive, well-founded, locally-finite, events at one location totally ordered.[3] The event ordering is extended with seven special event classes New, $Send$, Rcv, $Sign$, $Verify$, $Encrypt$, and $Decrypt$, three additional functions, $Honest$, $PrivKey$, and $MatchingKeys$, and the axioms $AxiomK$, $AxiomR$, $AxiomV$, $AxiomD$, $AxiomS$, $AxiomF$, $NoncesCiphersAndKeysDisjoint$, and $ActionsDisjoint$.

We package all the items listed above into a single type SES (a *security event structure*). All the definitions that follow (threads, protocols, etc.) have a parameter ses of type SES, but for readability we use that parameter implicitly.

4 Protocols

We are now ready to formally define protocols and the authentication properties they should satisfy.

4.1 Threads

A thread is an ordered list of actions at single location.

$$Thread \equiv_{def} \{thr : Act\ List \mid \forall i.thr[i] <_{loc} thr[i+1]\}$$

Define $thr_1 \preceq thr_2$ to mean that thread thr_1 is an initial segment of thread thr_2, and define $thr_1 \simeq thr_2$ to be $thr_1 \preceq thr_2 \ \vee \ thr_2 \preceq thr_1$.

[3] There are two more axioms, needed for constructive proofs, that state that $<$ and equality on E are *decidable*.

The messages of a thread are just the $Send$ and Rcv actions in the thread.

$$isMsg(e) \equiv_{def} \ e \in E(Send) \ \vee \ e \in E(Rcv)$$
$$messages(thr) \equiv_{def} \ filter(isMsg, thr)$$

Two messages, s and r, are a weak match $s \sim r$ if the first is a send and the second is a receive with the same information. They form a strong match $s \mapsto r$ if, in addition, s is causally before r.

$$\begin{aligned} s \sim r \quad &\equiv_{def} \quad s \in E(Send) \ \wedge \ r \in E(Rcv) \\ &\qquad \wedge \ Send(s) = Rcv(r) \\ s \mapsto r \quad &\equiv_{def} \quad s \sim r \ \wedge \ s < r \end{aligned}$$

4.2 Matching conversations

Two threads, thr_1 and thr_2, form a *matching conversation of length* n if they both contain at least n messages and when the first n messages from each thread are paired, each pair $\langle m_1, m_2 \rangle$ satisfies $m_1 \mapsto m_2 \ \vee \ m_2 \mapsto m_1$. In this case, we have a *strong* matching conversation and write $thr_1 \overset{n}{\approx} thr_2$ (the definition is straightforward, so we omit it to save space). If each pair $\langle m_1, m_2 \rangle$ satisfies only $m_1 \sim m_2 \ \vee \ m_2 \sim m_1$, we have a *weak* matching conversation, and write $thr_1 \overset{n}{\sim} thr_2$.

A protocol that guarantees a strong matching conversation between two threads at different locations is said to satisfy a *strong authentication property*. The strong property prevents replay attacks and is much harder to prove than the corresponding weak property that leaves out the causal ordering requirement.

4.3 Protocol actions

A protocol is described by a collection of *basic sequences* of actions. To define these, we need a type $ProtocolAction$ of the allowed actions. Members of this type are pair of a tag and a value (which we write as $\mathbf{tag}(value)$), where the tags are seven constant strings, $\mathbf{new}$,...,$\mathbf{decrypt}$, and the values have the corresponding type.

A protocol action pa *matches* an event e, written $pa(e)$, if the obvious things hold:

$$e \in E(New) \ \wedge \ pa = \mathbf{new}(New(e)) \ \vee$$
$$e \in E(Send) \ \wedge \ pa = \mathbf{send}(Send(e)) \ \vee \ldots$$

A list, pas, of protocol actions *matches* a thread thr, written $pas(thr)$, if they have the same length ($\|pas\| = \|thr\|$) and corresponding members of each list match.

$$\forall i < \|thr\|.pas[i](thr[i])$$

4.4 Basic sequences

A basic sequence is essentially a parameterized list of protocol actions. In the protocols we consider, two of the parameters are identifiers, the names A and B of agents. An agent A obeying a protocol can participate in any number of threads, each of which is an "instance" of one of the basic sequences of the protocol, and each with possibly different locations playing the role of B. Our general definition allows basic sequences with more than two participants, but for simplicity, we restrict to the case of two-locations-per-instance in this paper. Before proceeding with the formal definition, an example is in order. We use the example of the CR (challenge-response) protocol, shown in figure 4, from the

Initiator		Responder
$\mathbf{new}(m)$		
$\mathbf{send}(m)$	$\rightarrow$	$\mathbf{rcv}(m)$
		$\mathbf{new}(n)$
		$\mathbf{sign}(\langle\langle n, m, A\rangle, B, s\rangle)$
$\mathbf{rcv}(\langle n, s\rangle)$	$\leftarrow$	$\mathbf{send}(\langle n, s\rangle)$
$\mathbf{verify}(\langle\langle n, m, A\rangle, B, s\rangle)$		
$\mathbf{sign}(\langle\langle n, m, B\rangle, A, s'\rangle)$		
$\mathbf{send}(s')$	$\rightarrow$	$\mathbf{rcv}(s')$
		$\mathbf{verify}(\langle\langle n, m, B\rangle, A, s'\rangle)$

Figure 4: The Challenge/Response (*CR*) Protocol

paper [7] on *PCL*. The complete initiator and responder sequences have six actions each. There are two location parameters, A and B, and four additional parameters, m,n,s, and s', that stand for two nonces and two signatures. A thread thr is an instance of the initiator sequence for locations A and B, if it has length six, and location A, and for some assignment of atoms to the parameters m,n,s, and s', it matches the given sequence of protocol actions.

An agent A is said to *obey* a protocol if every action at location A is a member of an instance of one of its basic sequences. The verify and decrypt actions in a sequence do not occur if the signature or ciphertext are not correct. Similarly, the next receive action in a sequence may never occur if the other agents fail or misbehave. So that an agent can obey the protocol even so, we include all prefixes of the complete basic sequences that end just before a receive, verify, or decrypt action. For the CR-protocol the basic sequences are

$$\begin{aligned}
I_1 &= \mathbf{new}(m), \mathbf{send}(\langle A, m\rangle)\\
I_2 &= I_1,\ \mathbf{rcv}(\langle n, s\rangle)\\
I_3 &= I_2,\ \mathbf{verify}(\langle\langle n, m, A\rangle, B, s\rangle),\\
&\quad\ \mathbf{sign}(\langle\langle n, m, B\rangle, A, s'\rangle),\ \mathbf{send}(s')\\
R_1 &= \mathbf{rcv}(\langle A, m\rangle),\ \mathbf{new}(n),\\
&\quad\ \mathbf{sign}(\langle\langle n, m, A\rangle, B, s\rangle),\ \mathbf{send}(\langle n, s\rangle)\\
R_2 &= R_1,\ \mathbf{rcv}(s')\\
R_3 &= R_2,\ \mathbf{verify}(\langle\langle n, m, B\rangle, A, s'\rangle)
\end{aligned}$$

Formally, a basic sequence is a relation between two locations and a thread. The relation is true when the thread is an instance of the basic sequence with the given location parameters. Thus, the basic sequence is a member of the type[4]

$$\mathit{Basic} \equiv_{\mathit{def}} \mathit{Id} \rightarrow \mathit{Id} \rightarrow \mathit{Thread} \rightarrow \mathbb{P}$$

A basic sequence is defined by a list of protocol actions with free variables, all of them, except for A and B, interpreted as atoms. Since each instance of the basic sequence may generate different nonces, signatures, etc. the atom parameters are existentially quantified in the relation defined by the sequence. For example the basic sequence, I_1, is intended to occur at location A and "talk to" location B and has one additional free variable, so it defines the basic sequence relation

$$\lambda A, B, \mathit{thr}.\ \exists m\colon \mathit{Atom}.\ I_1[m](\mathit{thr})$$

[4] $\mathbb{P}$ means proposition. In constructive logic propositions are not the same as booleans, but the difference is not important for this paper.

Similarly, the basic sequence, R_1, is intended to occur at location B and "talk to" location A and has three additional free variables, so it defines the basic sequence relation

$$\lambda B, A, thr.\ \exists m, n, s\colon Atom.\ R_1[A, B, m, n, s](thr)$$

We say that a thread thr is one of a given list bss of basic sequence relations, at location A, and write $thr = oneOf(bss, A)$, if

$$loc(thr) = A\ \wedge \exists B\colon Id.\ \exists b \in bss.\ b(A, B, thr)$$

The relation $inOneOf(e, thr, bss, A)$, used in the formal definition of protocols, is defined by

$$e \in thr\ \wedge\ thr = oneOf(bss, A)$$

4.5 Formal protocols

A list bss of basic sequence relations of type $Basic$ defines a *protocol*— formally, a predicate on locations.

The protocol $Protocol(bss)$ is

$$\begin{aligned}&\lambda A.\ \forall e\colon Act.\ loc(e) = A \Rightarrow\\ &\quad (\exists thr.\ inOneOf(e, thr, bss, A)) \wedge\\ &\quad \forall thr_1, thr_2.(inOneOf(e, thr_1, bss, A) \wedge\\ &\qquad inOneOf(e, thr_2, bss, A))\\ &\qquad \Rightarrow thr_1 \simeq thr_2\end{aligned}$$

This says that every action at location A is a member of an instance of one of the basic sequences and if it is a member of two (or more) instances, then those instances are compatible, i.e. one is an initial segment of the other.

For example, if agent A satisfies the CR-protocol, then an action $e \in E(Send)$ could be the second event in an instance of sequence I_1. But it could also be the second event in an instance of the complete sequence I_3. The compatibility condition then implies that choice of parameter m in the two instances must agree.

4.6 Authentication

Consider an honest agent A obeying the CR-protocol in figure 4. If A performs an instance of the full initiator sequence with parameter B, then provided that B is honest and also obeys the CR-protocol, there should be an instance of the responder sequence at B that forms a matching conversation for the first two messages in the protocol (we can not be sure that the third message, sent by A, will be received by B). Similarly, if B performs an instance of the full responder sequence, then there should be a matching conversation with A of length three.

This motivates the following formal definition stating that in protocol Pr, the basic sequence bs authenticates n messages.

$$\begin{aligned}&Pr \models auth(bs, n) \equiv_{def}\\ &\forall A, B.\ \forall thr_1.\\ &(Honest(A)\ \wedge\ Honest(B)\ \wedge\ Pr(A)\ \wedge\ Pr(B)\\ &\quad \wedge A \neq B\ \wedge\ loc(thr_1) = A\ \wedge\ bs(A, B, thr_1))\\ &\Rightarrow \exists thr_2.\ loc(thr_2) = B\ \wedge\ thr_1 \overset{n}{\approx} thr_2\end{aligned}$$

5 Proof of *CR*-protocol

We define CR to be $Protocol([I_1, I_2, I_3, R_1, R_2, R_3])$ for the basic sequence relations defined earlier. The (strong) authentication properties we wish to verify are

$$CR \models auth(I_3, 2) \ \wedge \ CR \models auth(R_3, 3)$$

We start with $CR \models auth(I_3, 2)$. Part of the proof is very easy, but the rest requires a new idea.

Suppose $A \neq B$ are both honest and obey CR, and suppose that thread thr_1 is an instance of I_3. Let $e_0 <_{loc} e_1 <_{loc} \cdots <_{loc} e_5$ be the actions in thr_1. Then, $e_0, \ldots, e_5$ all have location A, and for some atoms, m,n,s,and s' we have

$$\begin{aligned} &New(e_0) = m \ \wedge \ Send(e_1) = m \ \wedge \\ &Rcv(e_2) = \langle n, s\rangle \ \wedge \ Verify(e_3) = \langle\langle n, m, A\rangle, B, s\rangle \end{aligned}$$

By $AxiomV$ and $AxiomS$, there is an event e' such that

$$e' < e_3 \ \wedge \ Sign(e') = Verify(e_3) \ \wedge \ loc(e') = B$$

Because B obeys CR, action e' must be a member of an instance of one of the basic sequences of CR. The only ones that include a $\mathbf{sign}(\_)$ action are I_3, R_1, R_2, and R_3. We rule out I_3, and show that R_1 (and hence R_2 and R_3) implies a matching conversation.

If e' is in an instance of I_3, then for some atoms m_1,n_1,s_1,s'_1, and some location C, there is an $e'_0 <_{loc} e'$ such that

$$\begin{aligned} &New(e'_0) = m_1 \ \wedge \\ &Sign(e') = \langle\langle n_1, m_1, C\rangle, B, s'_1\rangle \end{aligned}$$

But we also have

$$Sign(e') = Verify(e_3) = \langle\langle n, m, A\rangle, B, s\rangle$$

and this implies that $m_1 = m$. But then, $New(e_0) = New(e'_0)$, hence, by Lemma 3, $e_0 = e'_0$ and hence $A = loc(e_0) = loc(e'_0) = B$, contrary to assumption, so I_3 is ruled out.

If e' is in an instance of R_1, then for some atoms m_2,n_2,s_2, and some location D, there are events e'_0, e'_1, and e'_3 at location B such that

$$\begin{aligned} &e'_0 < e'_1 < e' < e'_3 \ \wedge \ Rcv(e'_0) = \langle D, m_2\rangle \\ &\wedge New(e'_1) = n_2 \ \wedge \ Sign(e') = \langle\langle n_2, m_2, D\rangle, B, s_2\rangle \\ &\wedge Send(e'_3) = \langle n_2, s_2\rangle \end{aligned}$$

Therefore, $\langle\langle n_2, m_2, D\rangle, B, s_2\rangle$ = $\langle\langle n, m, A\rangle, B, s\rangle$ and hence [5] $n_2 = n$, $m_2 = m$, $D = A$, and $s_2 = s$. So we have

$$\begin{aligned} &e'_0 < e'_1 < e' < e'_3 \ \wedge \ Rcv(e'_0) = \langle A, m\rangle \\ &\wedge New(e'_1) = n \ \wedge \ Sign(e') = \langle\langle n, m, A\rangle, B, s\rangle \\ &\wedge Send(e'_3) = \langle n, s\rangle \end{aligned}$$

[5] If responder B signs only the pair of nonces $\langle n, m\rangle$ and does not include the name of the initiator, we can not conclude $D = A$ at this step and the proof fails. There is, in fact, a man-in-the-middle attack against this faulty version of the *CR*-protocol.

The first two messages of the original thread are e_1 and e_2, and the first two messages of the instance of R_1 are e'_0 and e'_3. We have established that $Send(e_1) = \langle A, m\rangle = Rcv(e'_0)$ and $Send(e'_3) = \langle n, s\rangle = Rcv(e_2)$. So we have a (weak) matching conversation of length two.

All of the reasoning used to establish the weak matching conversation is simple and easily automated. From the original hypotheses, we "forward-chain" using the axioms of Authentication Event Logic. This may imply the existence of some additional events. The honesty axiom may imply that the location of some of these events is known. Then we can do a case-analysis on the additional events whose location is known to satisfy the protocol. In each case, we use equality reasoning to deduce that certain parameters are equal. If nonces are shown equal, then we try to use Lemma 3 to rule out a case. When all possible progress along these lines has been made, we look for the matching conversation.

Returning to the proof of CR, to show that we have a (strong) matching conversation, we must prove that $e_1 < e'_0$ and $e'_3 < e_2$. To prove such orderings, we must use $AxiomF$, but there is a problem. For example, $Rcv(e'_0) = \langle A, m\rangle$, so e'_0 **has** $New(e_0)$. By $AxiomF$, lemma 2, and the assumption $A \neq B$, it follows that there is a send event s between e_0 and e'_0 that releases the nonce m. If $e_1 \leq s$ then we have the desired ordering $e_1 < e'_0$, but how can we rule out the case $e_0 <_{loc} s <_{loc} e_1$?

If $e_0 <_{loc} s <_{loc} e_1$ then s would have to be a member of some other thread at A, but, in the general case, there could be other basic sequences in the protocol that could "accidentally" release nonce m before it should be released. For example, suppose that the protocol included a basic sequence that began with the action $\mathbf{send}(k)$ where k is a free variable. Then, an instance of that sequence would begin with an event $s \in E(Send)$ with $Send(s) = k$ *for some atom* k, and there would be no way to show that $k \neq m$.

We call such basic sequences *illegal* because they allow atoms to be chosen non-deterministically and hence to potentially leak information from another thread of the protocol. Since the CR-protocol does not contain illegal sequences, it does satisfy the strong authentication property $CR \models auth(I_3, 2)$, but we need to define the legal sequences and prove a stronger invariant about legal protocols.

5.1 Legal sequences

The intuition behind our definition of a legal sequence is that no action in the sequence should *use* an atom before it is *useable*. The useable atoms are the ones that are generated or received by the action while the used atoms are the parts of the information of the action that are not generated by the action.

A basic sequence at location A is *legal* if a used atom of any action is either the private key for A or is a useable atom of a prior action in the sequence. Formally, this is a syntactic condition that is automatically checkable for any list of protocol actions. A protocol is legal if all of its basic sequences are legal.

The following *nonce release lemma* states a stronger invariant for legal protocols.

Lemma 4 (nonce release lemma)**.** *If protocol $Protocol(bss)$ is legal, A is honest and obeys Pr, and thr is an instance of one of the basic sequences bss, $n = thr[j]$, $n \in E(New)$, $e = thr[i]$, and $j < i$, then if for no k between j and i, is $thr[k]$ in $E(Send)$, the nonce, $New(n)$, is not released before e.*

Proof. A formal proof has been carried out in NuPrl (using only the axioms of Authentication Event Logic). It is too long to include here, but the key idea is to strengthen the invariant even further to show that any event that potentially has the nonce is after e if its location is not A, and is not a send if it is locally before e. The stronger invariant is then proved by induction on the well-founded causal order. So the proof amounts to considering a minimal counterexample and showing that it can not exist. □

Using this lemma, we can finish the proof of $CR \models auth(I_3, 2)$. We first check that CR is indeed a legal protocol. Then, because there are no send actions (or any actions at all) between event e_0 and e_1 in thread thr_1, Lemma 4 implies that nonce m is not released before e_1 and this was all that was needed

for $AxiomF$ to imply that $e_1 < e'_0$. We must also show that $e'_3 < e_2$. This also follows from $AxiomF$ and Lemma 4 because e_2 **has** n and $n = \mathit{New}(e'_1)$ is not released before e'_3 (because there is only the sign-event e'_2 between them in the thread).

5.2 Proof of CR-Responder property

To finish the proof of the CR protocol we must prove that $CR \models auth(R_3, 3)$. We expected the proof to be similar to the one given for $CR \models auth(I_3, 2)$, but found that it is much harder—requiring more case analysis and another new idea.

Suppose $A \neq B$ are both honest and obey CR, and suppose that thread thr_1 is an instance of R_3 at location B. Let $e_0 <_{loc} e_1 <_{loc} \cdots <_{loc} e_5$ be the actions in thr_1. Then, $e_0, \ldots, e_5$ all have location B, and for some atoms, m,n,s,and s' we have

$$\begin{aligned}
&\mathit{Rcv}(e_0) = m \ \wedge\ \mathit{New}(e_1) = n \ \wedge \\
&\mathit{Sign}(e_2) = \langle\langle n, m, A\rangle, B, s\rangle \ \wedge\ \mathit{Send}(e_3) = \langle n, s\rangle \\
&\mathit{Rcv}(e_4) = s' \ \wedge\ \mathit{Verify}(e_5) = \langle\langle n, m, B\rangle, A, s'\rangle
\end{aligned}$$

By $AxiomV$ and $AxiomS$, there is an event e' such that

$$e' < e_5 \ \wedge\ \mathit{Sign}(e') = \mathit{Verify}(e_5) \ \wedge\ \mathit{loc}(e') = A$$

Because A obeys CR, action e' must be a member of an instance of one of the basic sequences of CR. The only ones that include a $\mathbf{sign}(\_)$ action are I_3, R_1, R_2, and R_3. R_1, R_2, and R_3 are ruled out in the same way that cases were ruled out in the proof of $CR \models auth(I_3, 2)$. So we must show that if e' is in an instance of I_3, then there is a matching conversation.

In this case, for some atoms m_1,n_1,s_1,s'_1, and some location C, we have $e'_0 <_{loc} e'_1 <_{loc} \cdots <_{loc} e'_5$ with $e' = e'_4$ where

$$\mathit{Sign}(e') = \langle\langle n_1, m_1, C\rangle, A, s'_1\rangle$$

Thus, $\langle\langle n_1, m_1, C\rangle, A, s'_1\rangle = \langle\langle n, m, B\rangle, A, s'\rangle$ so $n_1 = n$, $m_1 = m$, $C = B$, and $s'_1 = s'$, but note that we have not established that $s_1 = s$. Thus we have

$$\begin{aligned}
&\mathit{New}(e'_0) = m \ \wedge\ \mathit{Send}(e'_1) = m \\
&\mathit{Rcv}(e'_2) = \langle n, s_1\rangle \\
&\mathit{Verify}(e'_3) = \langle\langle n, m, A\rangle, B, s_1\rangle \\
&\mathit{Sign}(e') = \langle\langle n, m, B\rangle, A, s'\rangle \wedge\ \mathit{Send}(e'_5) = s'
\end{aligned}$$

To show that we have a (strong) matching conversation of length three, we must establish that $e'_1 < e_0$, $s_1 = s$ and $e_3 < e'_2$, and $e'_5 < e_4$. Both $e'_1 < e_0$ and $e_3 < e'_2$ follow from the nonce release lemma, Lemma 4.

To establish that $s_1 = s$ we need another round of case analysis. This time we consider the sign event e'' that matches the verify event e'_3. Event e'' must have

$$\mathit{Sign}(e'') = \langle\langle n, m, A\rangle, B, s_1\rangle$$

so it must occur at location B and be in an instance of R_1, R_2, R_3, of I_3. Again, the case I_3 is ruled out, and in each of the other cases, the thread, thr_2, that contains e'' has parameters m_2,n_2,s_2,s'_2, and C that satisfy $m_2 = m$, $n_2 = n$, $s_2 = s_1$, $C = A$. Because of this, the nonce events for m_2 and m are the same event. The definition of a protocol says that two instance threads that share an event must be

Initiator		Responder
$\mathbf{new}(m)$		
$\mathbf{send}(m)$	$\rightarrow$	$\mathbf{rcv}(m)$
		$\mathbf{new}(n)$
		$\mathbf{sign}(\langle\langle n, m, A\rangle, B, s\rangle)$
$\mathbf{rcv}(\langle n, s\rangle)$	$\leftarrow$	$\mathbf{send}(\langle n, s\rangle)$
$\mathbf{verify}(\langle\langle n, m, A\rangle, B, s\rangle)$		
$\mathbf{new}(k)$		
$\mathbf{sign}(\langle\langle n, k, m, B\rangle, A, s'\rangle)$		
$\mathbf{send}(\langle k, s'\rangle)$	$\rightarrow$	$\mathbf{rcv}(\langle k, s'\rangle)$
		$\mathbf{verify}(\langle\langle n, k, m, B\rangle, A, s'\rangle)$

Figure 5: Extra nonce variant CR_1

compatible—one is an initial segment of the other. Therefore at least the first four events of the threads thr_2 and thr_1 are equal, so $e'' = e_2$, and this implies that $s_1 = s$.

To finish the proof, we must show that the send and receive of the signature s' are in the correct causal order, viz. $e'_5 < e_4$. This would follow from the nonce release lemma if s' were a nonce, but it is a signature. Faced with this difficulty, our first response was to define the variant CR_1 of the CR-protocol shown in figure 5. This variant includes an extra nonce in the final message. All the reasoning used so far goes through unchanged, and, because of the extra nonce, the ordering relation $e'_5 < e_4$ now follows from Lemma 4. We automated all of the reasoning in the proof using a NuPrl tactic (described in the long version of this paper). The tactic constructs the proof of the strong authentication properties for the variant protocol completely automatically (i.e. with no user interaction).

6 Unique signatures

We want to verify the original CR-protocol without the extra nonce. Our intuition is that if a protocol never signs the same data twice, then all the signatures it generates are unique and function like nonces. To formalize this we proved the following lemma:

Lemma 5 (signature release lemma)**.** *If protocol $Protocol(bss)$ is legal and generates unique signatures, A is honest and obeys Pr, and thr is an instance of one of the basic sequences bss, $sg = thr[j]$, $n \in E(Sign)$, $e = thr[i]$, and $j < i$, then if for no k between j and i, is $thr[k]$ in $E(Send)$, then $signature(s)$ is not released before e.*

Proof. A formal proof, carried out in NuPrl, is very similar to the proof of lemma 4. The assumption that the signatures generated by instances of the protocol are all unique replaces the use of Lemma 3. □

Before we can use lemma 5 in our proofs, we need a way to establish that a protocol generates unique signatures. Examining the CR-protocol in figure 4, we note that in every sign action, the data signed includes a nonce, generated earlier in its sequence. Is this enough to guarantee that all signatures generated by the protocol are unique? We expected this to be true, but it is not! The CR-protocol provides a counterexample. Suppose that agent A is both the initiator and responder in a conversation with itself. Then the sign actions in both the initiator and responder threads will sign the data $\langle n, m, A\rangle$ so the same signature could be generated twice.

We could simply assume that all signatures are randomized and therefore unique, but we wanted our formal theory to include as few axioms as possible. Requiring that an agent may not engage in

Initiator		Responder
$\mathbf{new}(m)$		
$\mathbf{send}(m)$	$\rightarrow$	$\mathbf{rcv}(m)$
		$\mathbf{new}(n)$
		$\mathbf{sign}(\langle\langle n, A, m\rangle, B, s\rangle)$
$\mathbf{rcv}(\langle n, s\rangle)$	$\leftarrow$	$\mathbf{send}(\langle n, s\rangle)$
$\mathbf{verify}(\langle\langle n, A, m\rangle, B, s\rangle)$		
$\mathbf{sign}(\langle\langle m, B, n\rangle, A, s'\rangle)$		
$\mathbf{send}(\langle k, s'\rangle)$	$\rightarrow$	$\mathbf{rcv}(\langle k, s'\rangle)$
		$\mathbf{verify}(\langle\langle m, B, n\rangle, A, s'\rangle)$

Figure 6: Fresh signature variant CR_2

a conversation with itself would rule out this particular counterexample, but it is not clear that such a restriction is desirable, or that it would suffice to prove that all signatures are unique.

Another way to rule out the counterexample, and which turns out to be a sufficient condition to guarantee unique signatures, is to require that every sign action in a thread includes a nonce, generated in that thread, in a *standard position* in the data it signs.

Definition 1. *A protocol satisfies the* fresh signature criterion *if there is a function f of type $Data \rightarrow (Atom + Unit)$ such that for every sign action $\mathbf{sign}(\langle d, A, s\rangle)$ in a basic sequence there is a nonce $\mathbf{new}(m)$ earlier in the sequence, such that $f(d) = m$ and there is no other sign action $\mathbf{sign}(\langle d', A', s'\rangle)$ between them in the sequence with $f(d') = m$.*

Lemma 6 (fresh signatures)**.** *If a protocol satisfies the fresh signature criterion, then all signatures generated by instances of the protocol are unique.*

Proof. Suppose two actions in instances of the protocol generate the same signature. Then, by $AxiomF$, they have the same signed data d. By the fresh signature criterion, in each instance there is a nonce $n = f(d)$, and hence the two instances generate the same nonce. By lemma 3, the two nonce events are the same. By the definition of protocol, the two instances are compatible—one is an initial segment of the other. This is enough to imply that the two sign actions are the same, because, if not, one would occur between the common nonce event and the other sign action, contrary to the fresh signature criterion. □

The variant CR-protocol in figure 6 reorders the data in the sign actions so that a nonce from earlier in the sequence is always the *first* component of the tuple. This variant then satisfies the fresh signature criterion.

Automatic proof of fresh signature variant For a given function f, the fresh signature criterion is automatically checkable, and for any protocol we can easily guess what the function f must be. We added this check of the freshness criterion to our authentication tactic, and we also added tactics to derive consequences of the signature release lemma, lemma 5. With these additions, we expected the tactic to automatically prove the fresh signature variant CR_2 of CR shown in figure 6, but we had one final surprise. The seemingly minor reordering of the data in the protocol made a big difference in the proof. In particular, some cases in the original that were ruled out because they implied $A = B$, no longer do so, and must be ruled out by showing that they lead to a loop in the causal ordering. Our tactics had to be strengthened to carry out this reasoning. The full story is in the longer version of the paper.

7 Related Work

The purpose of this paper is simply to show how a logic like *PCL* is expressed in the logic of events and proofs automated in NuPrl, so we will not compare the logic itself with the many other formal approaches to verification of security properties. Instead, we use this section to address the issues raised in the critique of [7] by Cremers [6].

Cremers raises three main issues with [7]. In the first, he asserts, based on a logical analysis, that the axioms of *PCL* are not powerful enough to prove the existence of threads at remote locations. We believe that this analysis may be flawed by a misunderstanding of the powerful "honesty rule" in *PCL* which is actually an inference scheme. However, Cremers' critique does show that this part of *PCL* is hard to understand, and would probably be hard to use automatically. In contrast, our definition of protocol

$$\lambda A.\, \forall e\colon Act.\, loc(e) = A \Rightarrow (\exists thr.\, inOneOf(e, thr, bss, A)) \wedge \ldots$$

shows clearly how the assumption that an honest agent B obeys a protocol, can be used to establish the existence of a thread at location B—we only have to establish the existence of an action at B. This can be established by showing that B created a signature or encrypted or decrypted using its private key.

The second issue raised in [6] is that the consistency between basic sequences in *PCL* can not be established. We believe that this is correct. The notion of basic sequence in [7] breaks the roles of the protocol into disjoint sections (that begin at the start of the role, or at a receive action). In our version, the basic sequences are not disjoint, but are rather initial segments of the full sequence for the protocol role [6]. Because of this, we cannot assert that every action is in an instance of a *unique* basic sequence, instead we have the consistency criterion:

$$\forall thr_1, thr_2.(inOneOf(e, thr_1, bss, A) \wedge inOneOf(e, thr_2, bss, A)) \Rightarrow thr_1 \simeq thr_2$$

This difference addresses the problem with consistency between basic sequences noted by Cremers. It does, however, increase the number of cases that the theorem prover must consider because a given action may be in an instance of several compatible basic sequences.

The third issue raised in the critique is the most substantial. It asserts that proving authentication protocols such as Needham-Schoeder-Lowe (NSL) [10] that use only encryption rather than digital signatures requires the development of more theory. The problem is that without the use of digital signature it is hard to establish the location of some remote event. In the case of NSL, messages are encrypted with the public keys of agents, and such actions can occur anywhere. We have defined the NSL protocol and its desired properties in our system and, indeed, our proof tactic does not prove the authentication properties automatically—so we do need to develop and implement additional theory.

The basis for the additional theory we need has been developed by several researchers. For instance, while it is correct that the formal system presented in [7] does not provide sufficient machinery to prove authentication for the NSL protocol, an earlier paper [8] using a different formulation of some concepts, but the same basic approach as other PCL papers did provide a proof for NSL.

A more recent work [12] introduces an inductive method to establish secrecy properties in *PCL* and proves properties of Kerberos and a variant of NSL. Several authors have used similar inductive methods to establish that actions occur at a restricted set of locations. The theorem proving community traces these ideas to the inductive method of Paulson [11], also used by Cortier, Millen and Rueß [5], so we adopt their terminology in our adaptation of the method to Authentication Event Logic. The method establishes the existence of a set of secrets (keys, nonces, etc.) and a set of agents, called the *cabal*, such that actions that have any of the secrets are confined to the cabal. The paper [5] shows that finding such

[6] The initial segment definition of basic sequence was also used in another version of *PCL*.

sets of secrets and cabals is computationally easy.

We have proved such a "cabal lemma" (described in the longer version of this paper on the NuPrl website.) Our planned approach to automatic verification of protocols like NSL is to automate a tactic for finding a cabal. Such a cabal usually consists of only two or three agents, and once it is established, the proof can proceed by case analysis as before.

8 Conclusions

We have shown that a logic like *PCL* can be expressed using the language of event orderings and event classes, and that using atoms to represent nonces, keys, signatures, and ciphertexts gives a reasonably simple axiomatization of a theory in which authentication protocols can be formally defined and strong authentication properties automatically proven.

The exercise of implementing a formal theory and automating proofs in the theory makes all assumptions and definitions explicit and imposes a certain clarity on subject. We hope that even readers who object to some aspects of our axiomatic theory will agree on the value of such an exercise.

We would like to thank visiting fellow Meihua Xiao for the Wednesday afternoon visits in 2009 that prompted this work; Robert Constable for supporting this research and lecturing on the results; John Mitchell, Anupam Datta, and Arnab Roy for discussions about *PCL*, and David Guaspari for advice and editing.

References

[1] Stuart F. Allen. *An Abstract Semantics for Atoms in Nuprl*. Cornell Tech Report TR2006-2032, 2006.

[2] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *J. Applied Logic*, 4(4):428–469, 2006.

[3] Mark Bickford. Unguessable atoms: A logical foundation for security. In *VSTTE '08: Proceedings of the 2nd international conference on Verified Software: Theories, Tools, Experiments*, pages 30–53, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Mark Bickford. Component specification using event classes. In *CBSE '09: Proceedings of the 12th International Symposium on Component-Based Software Engineering*, pages 140–155, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] Véronique Cortier, Jonathan Millen, and Harald Rueß. Proving secrecy is easy enough. In *14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 2001.

[6] Cas Cremers. On the protocol composition logic pcl. In *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 66–76, New York, NY, USA, 2008. ACM.

[7] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. Protocol composition logic (pcl). *Electron. Notes Theor. Comput. Sci.*, 172:311–358, 2007.

[8] Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11, 2003.

[9] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[10] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[11] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 2000.

[12] Arnab Roy, Anupam Datta, Ante Derek, John C. Mitchell, and Jean-Pierre Seifert. Secrecy analysis in protocol composition logic. *Formal Logical Methods for System Security and Correctness*, 2008.