State and Progress in Strand Spaces: Proving Fair Exchange[*]

Joshua D. Guttman

June 21, 2011

Abstract

Many cryptographic protocols are intended to *coordinate state changes* among principals. Exchange protocols, for instance, coordinate delivery of new values to the participants, i.e. additions to the set of values they possess. An exchange protocol is *fair* if it ensures that delivery of new values is balanced: If one participant obtains a new possession via the protocol, then all other participants will, too.

Understanding this balanced coordination of different principals in a distributed system requires relating (long-term) *state* to (short-term) protocol activities. Fair exchange also requires *progress* assumptions.

In this paper we adapt the strand space framework to protocols, such as fair exchange, that coordinate state changes. We regard the *state* as a multiset of facts, and we allow protocol actions to cause local changes in this state via multiset rewriting. Second, *progress* assumptions stipulate that some channels are resilient—and guaranteed to deliver messages—and some principals will not stop at critical steps. Our proofs of correctness cleanly separate protocol properties, such as authentication and confidentiality, from properties about progress and state evolution.

G. Wang's recent fair exchange protocol illustrates the approach.

1 Introduction

Many cryptographic protocols *coordinate state changes* between principals in distributed systems. For instance, electronic commerce protocols aim to coordinate state changes among a customer, a merchant, and one or more financial institutions. The financial institutions should record credits and debits against the accounts of the customer and the merchant. These state changes correlate with state changes at the merchant and the customer. The merchant's state changes should include sending a shipping order to its warehouse. The customer records a copy of the shipping order, and a receipt for the payment,

[*]Funded in part by MITRE-Sponsored Research, and in part by National Science Foundation grant number CNS-0952287. Email: `guttman@{wpi.edu}`.

issued by its financial institution. The designer of an application-level protocol must ensure that these changes occur in a coordinated, transaction-like way.

State changes should occur only when the participants have taken certain actions; for instance, any funds transfer requires the payer's authorization. Moreover, changes should occur only when the participants have certain joint knowledge, e.g. that they all agree on the identities of the participants in the transaction, and the amount of money involved. These are *authentication* goals in the parlance of protocol analysis. There may also be *confidentiality* goals that limit joint knowledge. In our example, the customer and merchant should agree on the goods being purchased—which should not be disclosed to the bank—while the customer and bank should agree on the account number or card number, which should not be disclosed to the merchant.

Goal of this paper. We develop a model connecting protocol execution with state and state change. We use our model to provide a proof of a clever fair exchange protocol due to Guilin Wang [21], modulo a slight correction.

We believe that the strength of the model is evident in the proof's clean composition of protocol-specific reasoning with state-specific reasoning. In particular, our proof modularizes what it needs to know about protocol behavior into the four authentication properties given in Section 2, Lemmas 5–6. If any protocol achieves these authentication goals and its roles obey simple conditions on the ordering of events, then other details do not matter: it will succeed as a fair exchange protocol.

A *two-party fair exchange* protocol is a mechanism to deposit a pair of values atomically into the states of a pair of principals. Certified delivery protocols are a typical kind of fair exchange protocol. A certified delivery protocol aims to allow A, the sender of a message, to obtain a digitally signed receipt if the message is delivered to B. B should obtain the message together with signed evidence that it came from A. If a session fails, then neither principal should obtain these values. If it succeeds, then both should obtain them. The protocol goal is to cause state evolution of these participants to be *balanced*.

The "fair" in "fair exchange" refers to the balanced evolution of the state. "Fair" does not have the same sense as in some other uses in computer science, where an infinitely long execution is *fair* if any event actually occurs, assuming that it is enabled in an infinite subsequence of the states in that execution. In some frameworks, fairness in this latter sense helps to clarify the workings of fair exchange protocols [3, 6]. However, we show here how fair exchange protocols can also be understood independent of this notion of fairness. When we formalize Wang's protocol [21], we use an extension of the strand space model [16] in which there are no infinite executions or fairness assumptions.

As has been long known [10, 19], a deterministic fair exchange protocol must rely on a trusted third party T. Recent protocols generally follow [1] in using the trusted third party optimistically, i.e. T is never contacted in the extremely common case that a session terminates normally between the two participants. T is contacted only when one participant does not receive an expected message.

Each principal A, B, T has a state. T uses its state to record the sessions in which one participant has contacted it. For each such session, T remembers the outcome—whether T aborted the session or completed it successfully—so that it can deliver the same outcome to the other participant. The states of A, B simply records the ultimate result of each session in which it participates. The protocol guides the state's evolution to ensure balanced changes.

Strand space extensions. Two additions to strand spaces are needed to view protocols as solving to coordinated state change problems.

A *strand* is a sequence of actions executed by a single principal in a single local session of a protocol. We enrich strands to allow them to *synchronize* with the projection of the joint state that is local to the principal P executing the strand. We previously defined the actions on a strand to be either (1) message transmissions or (2) message receptions. We now extend the definition to allow the actions also to be *(3) state synchronization events.* P's state at a particular time may permit some state synchronization events and prohibit others, so that P's strands are blocked from the latter behaviors. Thus, the state constrains protocol behavior. Updates to P's state reflect actions of P's strands.

We represent states by multisets of facts, and state change by multiset rewriting [4, 9], although with several differences from Mitchell, Scedrov et al. First, they use multiset rewriting to model protocol and communication behavior, as well as the states of the principals. We instead use strands for the protocol and communication behavior. Our multiset rewriting steps change only a single principal's local state. Hence, second, in our rules we do not need existentials, which they used to model selection of fresh values. Third, we use "big" states that may have a high cardinality of facts. However, the big states are generally sparse, and extremely easy to implement with small data structures.

We also incorporate *guaranteed progress* assumptions into strand spaces. Protocols that establish balance properties need guaranteed progress. Since principals communicate by messages, one of them—call it A—must be ready to make its state change first. Some principal (either A or some third party) must send a message to B to enable it to make its state change. If this message never reaches B, B cannot execute its state change. Hence, in the absence of a mechanism to ensure progress, A has a strategy—by preventing future message deliveries—to prevent the joint state from returning to balance.

Progress has two ingredients. One is that certain message *transmissions* are *resilient* in the sense that they are guaranteed to be received sooner or later. For instance, each party's transmissions to the trusted third party should be resilient. The other ingredient is that a *recipient* (in this case, the trusted third party) will *progress*, and prepare a response. This response should also be transmitted resiliently. These two elements together ensure that the original sender will receive a reply. However, no particular time bound is required. We will not assume that these events will occur before other, independent events have occurred.

The two augmentations—state synchronization events and a way to stipulate progress—fit together to form a strand space theory usable for reasoning about coordinated state change.

Structure of this paper. Section 2 describes the general approach of Wang's protocol. Section 3 specifies it in a more precise way, using the notation of strand spaces.

Section 4 develops the authentication properties that we will rely on, summarized in two lemmas (Lemmas 5–6). Any protocol whose message flow satisfies these two lemmas, and which synchronizes with state history at the same points, will meet our needs.

Section 5 introduces our multiset rewriting framework, proving a locality property. This property says that state synchronization events of two different principals are always concurrent in the sense that they commute. Hence, coordination between different principals can only occur by protocol messages, not directly by state changes. We also formalize the state facts and rules for Wang's protocol, inferring central facts about computations using these rules. These (very easily verified) facts are summarized in Lemma 13. Any system of rules that satisfies Lemma 13 will meet our needs.

Section 6 gives definitions for guaranteed progress, applying them to Wang's protocol. Lemmas 20–21, the key conclusions of Section 6, jointly entail that any compliant principals executing a session with a session label L can always proceed to the end of a local run, assuming only that the trusted third party is ready to handle sessions labeled L.

In Section 7 we put the pieces together to show that it achieves its balanced state evolution goal (Thm. 23). In particular, the balance property of Thm. 23 depends only on:

Lemmas 4, 5 and 6 about the protocol structure;

Lemma 13 about the state history mechanism; and

Lemmas 20–21 about progress.

These lemmas factor the verification into three sharply distinguished components.

2 The Gist of Wang's Protocol

Wang's fair exchange protocol [21] is appealing because it is short—only three messages in the main exchange (Fig. 1)—and uses only "generic" cryptography. By generic cryptography, Wang means hash functions, standard digital signatures, and probabilistic asymmetric encryption such that the random parameter may be recovered when decryption occurs. RSA-OAEP is such a scheme.

Some other protocols proposed for fair exchange use more specialized primitives. For instance, Anteniese [2] studied protocols that use an encryption that can be verified before having the decryption key; and Garay, Jakobsson, and MacKenzie [12] proposed a protocol that uses a primitive called "private contract signatures." A private contract signature is not verifiable by an outsider to the transaction, but it may be verified by the peer. Moreover, the trusted third party can convert to a standard digital signature to recover a failed transaction. See also [5].

In many situations, the advantages of Wang's protocol—that it is short and uses only standard signatures—will probably outweigh its disadvantage, namely one additional step in dispute resolution (see below in this section, p. 8).

Terminology. We write:

$t_1 \hat{} t_2$ for the concatenation of t_1 and t_2;

$\{|t|\}_k$ for t encrypted with the key k;

$\{|t|\}_k^r$ for t encrypted with the key k using the recoverable random value r;

$\mathsf{h}(t)$ for a cryptographic hash of t; and

$[\![t]\!]_k$ for a digital signature on t which may be verified using key k. By this, we mean t together with a cryptographic value v prepared from $\mathsf{h}(t)$ using the private signature key k^{-1} corresponding to k.

When we use a principal name A, B, T in place of k, we mean that a public key associated with that principal is used. Thus:

$\{|t|\}_T^r$ is an asymmetric encryption using T's public encryption key; and

$[\![t]\!]_A$ is a digital signature that may be verified using A's public verification key.

Message ingredients such as $\mathsf{keytag}, \mathsf{ab\_rq}, \mathsf{ab\_cf}$, etc., are distinctive bit-patterns used to tag data, indicate requests or confirmations, etc. Our notation differs somewhat from Wang's; for instance, his L is our $\mathsf{h}(L)$.

2.1 Main exchange

In the first message (Fig. 1), A sends the payload M to B encrypted with a key K, as well as K encrypted with the public encryption key of the trusted third party T. A also sends a digitally signed unit EOO asserting that the payload (etc.) originate with A. The value L serves to identify this session uniquely.

In the second message, B countersigns $\mathsf{h}(L), \mathsf{EK}$.

In the third message, A discloses K and the random value R used originally to encrypt K for T. B uses this information to obtain M, and also to reconstruct EK, and thus to validate that the hashes inside EOO are correctly constructed. At the end of a successful exchange, each party deposits the resulting values as a record in its state repository.

2.2 Abort and recovery subprotocols

What can go wrong? If the signature keys are uncompromised and the random values K, R are freshly chosen, only two things can fail:

$$\begin{aligned} A \to B: &\quad L \,\hat{}\, \mathsf{EM} \,\hat{}\, \mathsf{EK} \,\hat{}\, \mathsf{EOO} \\ B \to A: &\quad \mathsf{EOR} \\ A \to B: &\quad K \,\hat{}\, R \end{aligned}$$

where:

$$L = A \,\hat{}\, B \,\hat{}\, T \,\hat{}\, \mathsf{h}(\mathsf{EM}) \,\hat{}\, \mathsf{h}(K) \qquad \mathsf{EM} = \{\!|M|\!\}_K$$

$$\mathsf{EK} = \{\!|\, \mathsf{keytag} \,\hat{}\, \mathsf{h}(L) \,\hat{}\, K |\!\}_T^R$$

$$\mathsf{EOO} = [\![\, \mathsf{eootag} \,\hat{}\, \mathsf{h}(L) \,\hat{}\, \mathsf{EK} \,]\!]_A \qquad \mathsf{EOR} = [\![\, \mathsf{eortag} \,\hat{}\, \mathsf{h}(L) \,\hat{}\, \mathsf{EK} \,]\!]_B$$

Figure 1: Wang's protocol: A Successful Run

- A does not receive B's evidence of receipt EOR, and would like to abort this incomplete session. A prepares and signs an *abort request* AR containing L, which it transmits to the trusted third party T.
- B does not receive a correct K, R, and would like to recover them with the help of the trusted third party. B signs a value indicating a recovery request for L, and forwards that to T as a unit RR that also includes the signed evidence of receipt and other session data.

What can T do, if contacted in connection with a session L?

- T can abort L by countersigning an abort request AR. The resulting *abort confirmation token* AT certifies that A requested that the session be aborted, and T accepted that request.
- T can recover L by delivering K, R to B. T can extract them from the encrypted unit $\mathsf{EK} = \{\!|\, \mathsf{keytag} \,\hat{}\, \mathsf{h}(L) \,\hat{}\, K |\!\}_T^R$, using its private decryption key.

  If T's attempt to decrypt fails, or yields a value incompatible with the session information, then no harm is done: A will never be able to convince a judge that a valid transaction occurred. Wang's protocol returns an error message that we do not show here [21, Fig. 3].

However, a crucial condition is that if T receives both an abort request and a recovery request for the same session L, then T should provide compatible responses to both parties. If it provided AT to A and K, R to B, then B would succeed at decrypting M with K, adding the message and its EOO to its state repository. But A would not have received the balancing EOR.

Thus, when receiving the first request for any session L, T must record its action. It then responds to the other party to L in a matching way. If T receives an abort request for a new session L, then it records its abort token AT, to be delivered to B in lieu of recovery information in case a recovery request arrives

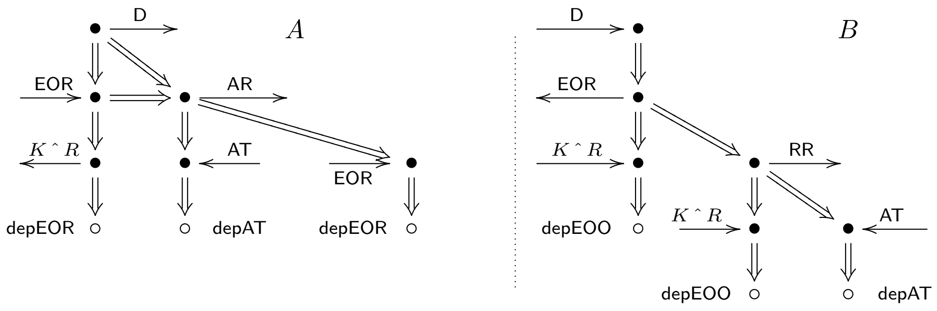


Figure 2: Initiator (A) and Responder (B) Behavior

subsequently. Conversely, before delivering K, R to B, it must receive EOR, and check that it is correctly formed, and record it, so that if A subsequently requests an abort for L, T can deliver EOR to provide the balancing outcome. This is the core reason why the trusted third party in a fair exchange protocol must keep long term state.

Thus, T has essentially four state-manipulating actions:

Abort When an abort request arrives for a session L, and no session information is stored for L, then T issues an abort token AT, which is stored also for L.

Recover When a recovery request arrives for a session L, and no session information is stored for L, then T extracts K, R, storing the EOR for L.

Forced recovery When an abort request arrives for a session L, but an EOR is already stored for L, then the EOR is transmitted in response, with no further state change.

Forced abort When a recovery request arrives for a session L, but an AT is already stored for L, then the AT is transmitted in response, with no further state change.

For completeness, we also allow an abort event to occur when an abort request arrives for a session L, and a matching AT is already stored for L. In this case, the same AT remains in the store. Similarly, when a recovery request arrives for a session L, and a matching EOR is already stored for L, then the same K, R should be retransmitted in response, with no further state change.

Hence, if A makes an abort request and B also makes a recovery request, perhaps because EOR was sent but lost in transmission, then T services whichever request is received first. When the other party's request is received, T reports the result of that first action. The local behaviors (strands) for A, B in this protocol are shown in Fig. 2. The local sessions (strands) are the paths from a

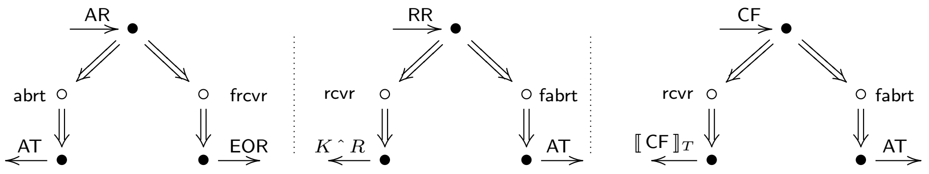


Figure 3: Trusted Third Party: Abort (left), Resolve (center), and Confirm (right) Requests

root to a terminal node; there are four paths for A and three paths for B. The solid nodes indicate messages to be sent or received, while the hollow nodes $\circ$ indicate events in which the participants deposit results into their state repositories. This figure is not precise about the forms of the messages, the parameters available to each participant at each point in its run, or the parameters to the state synchronization events. For instance, B does not know whether a claimed EM is really of the form $\{\!|M|\!\}_K$ when first receiving it, nor what M, K would produce the message received. We will clarify these details in Section 3.

A's abort request AR elicits an abort confirmation $[\![\mathsf{AR}]\!]_T$ if it reaches T first, but it elicits a recovery token $L\,\hat{}\,\mathsf{EOR}$ if B's recovery request was received first. Likewise, B's recovery request RR elicits $K\,\hat{}\,R$ if it is received first, but it elicits the abort confirmation $[\![\mathsf{AR}]\!]_T$ if A's abort request was received first. T must synchronize with its state to ensure that these different requests are serviced in compatible ways, depending on whichever arrived first. This compatibility of responses ensures that A, B will execute balanced state changes.

These behaviors of the trusted third party T, together with an additional behavior concerned with dispute resolution, are summarized in Fig. 3. We have indicated here that T's behavior, in response to an abort request AR may lead either to an abort token AT, or else to evidence of receipt EOR. Now, the hollow nodes $\circ$ *guard* the choice of branch. T transmits AR only after an abrt event, and EOR only after a frcvr event. In response to a recovery request RR from B, T may transmit $K\,\hat{}\,R$ or an abort token AT; however, the former occurs only after a rcvr event and the latter only after a fabrt event. Thus, the essential job for T's long term state in this protocol is to ensure that if an abrt event occurs for session L, then a rcvr never happens for L, and vice versa. This is easily accomplished by a state-based mechanism.

2.3 Dispute Resolution

A subtlety in this protocol concerns dispute resolution. Since A receives EOR before disclosing $K\,\hat{}\,R$, A could choose to abort at this point. A dishonest A could later choose between proving delivery via EOR and proving that this session aborted via the abort token AT. To prevent this, the protocol stipulates an extra step for a judge resolving disputes. If A presents the judge with EOR, then the judge queries B or T for an abort token. The judge does not accept A's presented EOR if the abort token is also available.

However, this is asymmetric. The abort token is used only by B (or T on B's behalf) to dispute receipt. A can never use it to dispute origin [21, Sec. 4.4], because of essentially the same abuse just mentioned.

For simplicity, we will assume that the judge is identical with T. When asked by A to confirm an EOR, T does so if the session has not aborted. When confirming an EOR, T must ensure that the session will never abort in the future, so that an EOR confirmation is handled similarly to a recovery request. If the session has already aborted, then T returns the abort token instead. This behavior is summarized in the behaviors starting from the reception of a confirmation request CF at the right side of Fig. 3.

This step may make Wang's protocol undesirable in some cases, where T may no longer be available for dispute resolution. It is also why Wang's protocol can use fewer messages than the four that Pfitzmann-Schunter-Waidner proved to be needed in a fair exchange protocol with asynchronous communication [18].

2.4 Our Correction to Wang's Protocol

We have adjusted Wang's protocol, so that B receives $[\![\, \mathsf{AR} \,]\!]_T$ in the forced abort case. In the original description, B received only AR.

However, in the original protocol, a dishonest B has a strategy to defeat the fairness of the protocol. Namely, after receiving the first message, B does not reply to A, but immediately requests resolution from T, generally receiving $K \mathbin{\hat{}} R$ from T. When A requests an abort from T, B attempts to read this abort request from the network. If successful, B has both AR and $K \mathbin{\hat{}} R$. Hence, it can subsequently choose whether to insist that the message was delivered, using the valid EOO, or whether to repudiate receipt, using the AR.

Whether this attack is possible depends on the nature of the channel between A and T. Under the usual assumption that the channel is resilient just in the sense of ensuring delivery, the attack is possible. If the channel offers both resilience and confidentiality, then the attack would be impossible. We have stipulated that B needs the countersigned $\mathsf{AT} = [\![\cdots \mathsf{AR} \cdots]\!]_T$ to make this attack infeasible on the standard assumption of resiliency only.

3 Wang's Protocol: a Specification

Participants cannot always immediately verify that a message they have received is of the form intended by the protocol. For instance, B cannot verify that a message component received where EK is expected is actually of the form $\{\!|\, \mathsf{keytag} \mathbin{\hat{}} \mathsf{h}(L) \mathbin{\hat{}} K |\!\}_T^R$. Nor can B validate that the last component of L is of the form $\mathsf{h}(K)$, since B has not yet received K.[1] Likewise, T, when receiving an

[1]Indeed, if the argument to h is a 160-bit value, and the symmetric keys are 128 bits, so that in fact h is applied to a key followed by padding of a prescribed form, then a given

abort request AR for a session L, will never validate that the last two components of L are of the form $\mathsf{h}(\mathsf{EM})$ and $\mathsf{h}(K)$.

For these and other messages, the sender and recipient do not always have the same view of how the message is constructed from its parameters. In this section, we provide a more detailed specification of the protocol, using the strand space formalism. This description clarifies exactly what messages a participant sends and receives, even in cases where a compromised peer may disobey the protocol with regard to message components that a recipient cannot immediately check. We start by summarizing the core strand space definitions.

3.1 Preliminaries: Messages and Protocols

In this section, we provide an overview of the current strand space framework; this section follows [13], except that it adds internal state synchronization events. See also [16, 8].

Message Algebra. Let A_0 be an algebra equipped with some operators and a set of homomorphisms $\eta\colon \mathsf{A}_0 \to \mathsf{A}_0$. We call members of A_0 *atoms*.

For the sake of definiteness, we will assume here that A_0 is the disjoint union of infinite sets of *nonces*, *atomic keys* (which we divide into *symmetric keys* and *asymmetric keys*), *names*, and *texts*. The operator $\mathsf{signk}(a)$ maps names to (asymmetric) signature keys, and $\mathsf{pubk}(a)$ maps names to (asymmetric) public encryption keys. K^{-1} maps an asymmetric atomic key to its inverse, and a symmetric atomic key to itself. Homomorphisms η are maps that respect sorts, and act homomorphically on $\mathsf{signk}(a)$ and K^{-1}.

Let X be an infinite set disjoint from A_0; its members—called *indeterminates*—act like unsorted variables.

A is freely generated from $\mathsf{A}_0 \cup X$ by two operations:

Encryption: The encryption of t_0 using t_1 as key is written $\{|t_0|\}_{t_1}$. In $\{|t_0|\}_{t_1}$, a non-atomic key t_1 is a symmetric key.

Tagged concatenation: The tagged concatenation of t_0 and t_1 using *tag* as tag is written *tag* $t_0 \hat{\ } t_1$. For a distinguished tag *nil*, we write *nil* $t_0 \hat{\ } t_1$ as $t_0 \hat{\ } t_1$ with no visible tag.

Members of A are called *messages*.

This algebra does not contain a separate hashing operator $\mathsf{h}(t_0)$. We regard the operator as a public key encryption using a public key K_h such that no principal knows the corresponding private key K_h^{-1}. Thus, any principal possessing t_0 can construct $\{|t_0|\}_{K_h}$. A principal who has received a value $\{|t_0|\}_{K_h}$ and has a hypothesis about the t_0 can test this hypothesis by re-encrypting using K_h and testing for equality. Thus, our encryption represents a non-probabilistic encryption.

bitstring may in fact not equal $\mathsf{h}(K)$ for any symmetric key K.

To represent a probabilistic encryption with recoverable randomness, such as the operator $\{\!|t_0|\!\}_K^R$, we use $\{\!|\mathit{prob}\ t_0\,\hat{}\,R|\!\}_K$, where the tag *prob* indicates the specific roles of t_0 and R. R is certainly recoverable from $\{\!|\mathit{prob}\ t_0\,\hat{}\,R|\!\}_K$.

A homomorphism $\alpha = (\eta, \chi)\colon \mathsf{A} \to \mathsf{A}$ consists of a homomorphism η on atoms and a function $\chi\colon X \to \mathsf{A}$. It is defined for all $t \in \mathsf{A}$ by the conditions:

$$\begin{array}{llr} \alpha(a) = \eta(a), & \text{if } a \in \mathsf{A}_0 & \alpha(\{\!|t_0|\!\}_{t_1}) = \{\!|\alpha(t_0)|\!\}_{\alpha(t_1)} \\ \alpha(x) = \chi(x), & \text{if } x \in X & \alpha(\mathit{tag}\ t_0\,\hat{}\,t_1) = \mathit{tag}\ \alpha(t_0)\,\hat{}\,\alpha(t_1) \end{array}$$

Thus, atoms serve as typed variables, replaceable only by other values of the same sort, while indeterminates x are untyped. Indeterminates x serve as blank slots, to be filled by any $\chi(x) \in \mathsf{A}$. Indeterminates and atoms are jointly *parameters.* The *instances* of a message t_0 are its images $\alpha(t_0)$ under homomorphisms α.

Messages are abstract syntax trees in the usual way:

1. Let ℓ and r be the partial functions such that for $t = \{\!|t_1|\!\}_{t_2}$ or $t = \mathit{tag}\ t_1\,\hat{}\,t_2$, $\ell(t) = t_1$ and $r(t) = t_2$; and for $t \in \mathsf{A}_0$, ℓ and r are undefined.

2. A *path* p is a sequence in $\{\ell, r\}^*$. We regard p as a partial function, where $\langle\rangle = \mathsf{Id}$ and $\mathsf{cons}(f, p) = p \circ f$. When the rhs is defined, we have: 1. $\langle\rangle(t) = t$; 2. $\mathsf{cons}(\ell, p)(t) = p(\ell(t))$; and 3. $\mathsf{cons}(r, p)(t) = p(r(t))$.

3. *p traverses a key edge* in t if $p_1(t)$ is an encryption, where $p = p_1{}^\frown\langle r\rangle{}^\frown p_2$.

4. *t_0 is an ingredient of* t, written $t_0 \sqsubseteq t$, if $t_0 = p(t)$ for some p that does not traverse a key edge in t.

5. t_0 *appears in* t, written $t_0 \ll t$, if $t_0 = p(t)$ for some p.

Strands. A *strand* is a (linearly ordered) sequence of nodes $n_1 \Rightarrow \ldots \Rightarrow n_j$, each of which represents either:

Transmission of some message $\mathsf{msg}(n_i) = t_i$, graphically $\bullet \xrightarrow{t_i}$;

Reception of some message $\mathsf{msg}(n_i) = t_i$, graphically $\xrightarrow{t_i} \bullet$; or

State synchronization labeled by some *fact* $E(a_1, \ldots, a_k)$, i.e. a variable-free atomic formula, graphically ${}^{E(a_1,\ldots,a_k)}\ \circ$.

We write $s \downarrow i$ for the i^{th} node of s, using 1-based notation. We show transmission and reception nodes by bullets $\bullet$ and state synchronization nodes by hollow circles $\circ$. In Figs. 2–3, the columns of nodes connected by double arrows $\Rightarrow$ are strands.

We lift homomorphisms α to strands s by mapping α through the nodes of s. The *instances* of a strand s are all of its images $\alpha(s)$ under homomorphisms α of the message algebra.

Origination. A message t_0 *originates* at a node n_1 if (1) n_1 is a transmission node; (2) $t_0 \sqsubseteq \mathsf{msg}(n_1)$; and (3) whenever $n_0 \Rightarrow^+ n_1$, $t_0 \not\sqsubseteq \mathsf{msg}(n_0)$.

Thus, t_0 originates when it was transmitted without having been either received, transmitted, or synchronized previously on the same strand. Values assumed to originate only on one node in an execution—*uniquely originating* values—formalize the idea of freshly chosen, unguessable values. Values assumed to originate nowhere may be used to encrypt or decrypt, but are never sent as message ingredients. They are called *non-originating* values. For a non-originating value K, $K \not\sqsubseteq t$ for any transmitted message t. However, $K \ll \{\!|t_0|\!\}_K \sqsubseteq t$ possibly, which is why we distinguish $\sqsubseteq$ from $\ll$.

A strand may represent the behavior of a principal in a single local session of a protocol, in which case it is a *regular* strand of that protocol, or it may represent a basic *adversary* activity. Basic adversary activities include receiving a plaintext and a key and transmitting the result of the encryption, and similar activities: An *adversary strand* has any of the following forms:

M_a: $\langle +a \rangle$ where a is basic value
M_g: $\langle +g \rangle$ where g is an indeterminate
C: $\langle -g \Rightarrow \cdots \Rightarrow -h \Rightarrow +tag\ g\,\hat{}\,\ldots\,\hat{}\,h \rangle$
S: $\langle -tag\ g\,\hat{}\,\ldots\,\hat{}\,h \Rightarrow +g \Rightarrow \cdots \Rightarrow +h \rangle$
E: $\langle -K \Rightarrow -h \Rightarrow +\{\!|h|\!\}_K \rangle$
D: $\langle -K^{-1} \Rightarrow -\{\!|h|\!\}_K \Rightarrow +h \rangle$

Since strands are linearly ordered sequences of events, there is no such thing as a branching strand. Each directed acyclic graph in Figs. 2–3 specifies a number of strands, not a strand that branches. The maximal paths through each DAG are distinct strands that share a common initial segment, namely the subpath that precedes the point at which they are distinguished. An execution that has only used this initial part is not yet "committed" to one path or the other. We therefore regard it as being the same, regardless of which path the not-yet-executed part would choose. We formalize this in Section 3.2, Definition 2.

Protocols. A *protocol* Π is a finite set of strands, called the *roles* of the protocol. These strands, the roles of Π, are like templates that define the behavior permitted for principals adhering to the protocol. Wang's protocol has, as its roles, the fourteen paths from roots to terminal nodes in Figs. 2–3.

The *instances* of a role $r \in \Pi$, are all the strands s such that $s = \alpha(r)$, i.e. s results from the role r by applying a homomorphism α to it. A set of atoms and indeterminates $\{a_1, \ldots a_k, x_1, \ldots, x_\ell\}$ *parametrize* a role $r \in \Pi$ iff

1. $\alpha(r \downarrow j) = \beta(r \downarrow j)$ for all j up to the length of r, when α, β are any pair of homomorphisms such that $\alpha(a_i) = \beta(a_i)$ for all $i \leq k$ and $\alpha(x_i) = \beta(x_i)$ for all $i \leq \ell$; and

2. Clause 1 is false for any proper subset of $\{a_1, \ldots a_k, x_1, \ldots, x_\ell\}$.

That is, the a_i, x_i are a minimal set which suffice to determine the instances of r under any homomorphism. There may be more than one choice of parametrization for a role (for instance, when $K \neq K^{-1}$, either could be used as the parameter), but there are always parametrizations. In the remainder of this section, we will clarify exactly the set of instances for each role, by describing parametrizations for them (Figs. 4–5, 7–8).

Protocols as defined elsewhere [8, 13] have some additional structure which is not relevant here and is therefore omitted.

Bundles. A *bundle* $\mathcal{B}$ is a finite directed acyclic graph whose vertices are strand nodes, and whose arrows are either strand edges $\Rightarrow$ or communication arrows $\rightarrow$. A bundle satisfies three properties:

1. If $m \rightarrow n$, then m is a transmission node, n is a reception node, and $\mathsf{msg}(m) = \mathsf{msg}(n)$.
2. Every reception node $n \in \mathcal{B}$ has exactly one incoming $\rightarrow$ arrow.
3. If $n \in \mathcal{B}$ and $m \Rightarrow n$, then $m \in \mathcal{B}$.

We write $m \preceq_{\mathcal{B}} n$ when there is a path from m to n in $\mathcal{B}$ using arrows in $\Rightarrow \cup \rightarrow$. Bundles, which may include both adversary strands and regular strands, represent possible protocol executions.

Clause 3 requires that each bundle contains an initial segment of the nodes lying on any one strand. However, it does not have to contain all of the nodes on a strand. In this case, the bundle represents a moment when a principal is only part way through some session. Indeed, this principal may have decided to terminate its involvement in the session. The *$\mathcal{B}$-height* of a strand is the number of nodes on it that are in $\mathcal{B}$. If all of the nodes of a strand s are in $\mathcal{B}$, we say that s has full height in $\mathcal{B}$.

We assume that strands, nodes, and bundles are disjoint from A. We say that a strand s is *in* $\mathcal{B}$ if s has at least one node in $\mathcal{B}$.

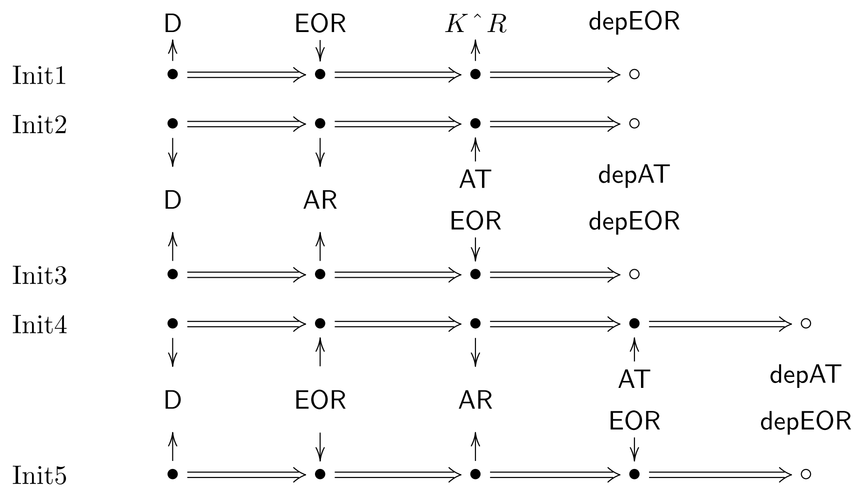
Proposition 1 *Let $\mathcal{B}$ be a bundle.*

1. *$\preceq_{\mathcal{B}}$ is a well-founded partial order.*
2. *Every non-empty set of nodes of $\mathcal{B}$ has $\preceq_{\mathcal{B}}$-minimal members.*
3. *If $a \sqsubseteq \mathsf{msg}(n)$ for any $n \in \mathcal{B}$, then a originates at some $m \preceq_{\mathcal{B}} n$.*

Recall that a originates at a transmission node m if $a \sqsubseteq \mathsf{msg}(m)$, and for all m_0 earlier on the same strand, a is neither sent nor received on m_0. I.e. $m_0 \Rightarrow^{+} m$ implies $a \not\sqsubseteq \mathsf{msg}(m_0)$.

Clause 3 justifies our use of non-origination as a way to express that a key is *uncompromised.* In particular, if in $\mathcal{B}$ the adversary ever uses K to encrypt or decrypt, then K is received on the first node n of an encryption or decryption strand. Thus, K originates on some $m \preceq_{\mathcal{B}} n$. By contraposition, keys that originate nowhere are used only on regular strands.

Values are *freshly chosen* in $\mathcal{B}$ if they originate at just one node. If a is uniquely originating on some regular node n, then the adversary does not guess the value a in the sense of originating the same value on an M_a strand. Moreover, no other regular strand has made a choice that unintentionally collides with a.



$\mathsf{AR} = [\![\, \mathsf{ab\_rq}\ \ L \,]\!]_A \qquad \mathsf{AT} = [\![\, \mathsf{ab\_cf}\ \ \mathsf{AR} \,]\!]_T$

Parameters: A, B, T, M, K, R

Figure 4: Initiator Strands

3.2 Specifying the Initiator

As an example of unfolding directed acyclic graphs, the DAG for the initiator (Fig. 2, left) unfolds to the five strands shown in Fig. 4, in which we use the same abbreviations as in Fig. 1, and for brevity write $\mathsf{D} = L\,\hat{}\ \mathsf{EM}\ \hat{}\ \mathsf{EK}\ \hat{}\ \mathsf{EOO}$. In this case, each of the five strands Init$i$ has the same parameters, namely A, B, T, M, K, R. These parameters are chosen by A at the beginning of the session, so nothing additional can be learnt about them. In this situation, the DAG notation of Fig. 2 leaves nothing to be desired.

Indeed, it has an advantage over the separate strand notation of Fig. 4. Namely, if a node is the first node of any of these strands, it can be continued into an execution of any of the other strands. Moreover, any node 2 on Init1, Init4, or Init5 may be continued into an execution of any of the others. A node 3 on either Init4, or Init5 may be continued into an execution of the other. A node 2 on Init2 or Init3 may be continued into an execution of the other.

Indeed, we regard a transformation on bundles as an isomorphism if it replaces the nodes of one strand by the nodes of another strand with the same parameters, and it is permitted by the rules we have just described. When the parameters are determined at the start, the DAG notation has the advantage that two initial segments may be interchanged (to within isomorphism) if and only if those initial segments may be written as the same partial path. We summarize this interchangeability by saying that the nodes are *similar*:

Definition 2 *Strands s, s' are* similar up to k *if, for each i where $1 \leq i \leq k$,*

1. *the directions (transmission, reception, or state synchronization) of their corresponding nodes $s \downarrow i$ and $s' \downarrow i$ agree; and*
2. *$s \downarrow i$ and $s' \downarrow i$ have the same message or state synchronization event.*

Nodes n, n' are similar*, written $n \sim n'$, if they are $s \downarrow i$ and $s' \downarrow i$ for strands s, s' that are similar up to some $k \geq i$.*

Bundles $\mathcal{B}, \mathcal{B}'$ are similar*, written $\mathcal{B} \sim \mathcal{B}'$, if one results from the other by replacing nodes with similar nodes.*

According to the notion of homomorphism of [8], if $\mathcal{B} \sim \mathcal{B}'$, then $\mathcal{B}, \mathcal{B}'$ are certainly isomorphic. Thus, we need not distinguish between similar bundles. This is convenient: It means that we can replace the part of s up to i with the part of s' up to i whenever s and s' will diverge only at some point after i. What has happened "so far" does not distinguish s from s'.

3.3 Specifying the Responder

However, it is worthwhile to expand the DAG representation when the parameters are not fully determined at the start. When the initial message reaches B, B cannot check that it is of the form D.

If we write L^* as an abbreviation for $A \,\hat{}\, B \,\hat{}\, T \,\hat{}\, \mathsf{h}(e_1) \,\hat{}\, x$, then the most B can check is that the first message is of the form D^*, where:

$$\mathsf{D}^* = L^* \,\hat{}\, e_1 \,\hat{}\, e_2 \,\hat{}\, [\![\, \mathsf{eootag} \,\hat{}\, \mathsf{h}(L^*) \,\hat{}\, e_1 \,]\!]_A. \tag{1}$$

However, B can say nothing about whether $x = \mathsf{h}(k)$ for any key k, nor whether e_1 and e_2 are really encryptions, or are really prepared from plaintexts of the right form. Thus, the parameters to the first node of a responder strand are essentially: A, B, T, e_1, x, e_2. However, on node 3 of the main protocol, values K and R are received. Now, B will accept them only if they disclose that the parameters e_1, x, e_2 are of suitable forms; namely, whether, for some M:

$$e_1 = \{\!|M|\!\}_K; \quad x = \mathsf{h}(K); \quad e_2 = \{\!|\, \mathsf{keytag} \,\hat{}\, \mathsf{h}(L^*) \,\hat{}\, K |\!\}_T^R. \tag{2}$$

Thus, if B succeeds at receiving $K \,\hat{}\, R$ as node 3 of a strand of the main protocol, B has succeeded in *refining* his knowledge of the parameters. He has learnt that the old parameters take the explicit forms shown, for suitable values of the newly introduced parameters K, R, M. A message $K' \,\hat{}\, R'$ for which Eqn. 2 does not hold must not be accepted as an instance of node 3, in a strand in which the initial message took the form shown in Eqn. 1. In a run of the responder role ending with an abort token, however, responder will never refine his knowledge of the parameters, and, thus, we have the strands shown in Fig. 5. Here, the strands Resp1 and Resp2 have the desired parameters A, B, T, M, K, R. However, Resp3 has the less informative parameters A, B, T, e_1, x, e_2.

$$\begin{tikzcd}
 & \mathsf{D} \arrow[d] & \mathsf{EOR} & K\,\hat{}\,R \arrow[d] & \mathsf{depEOO} \\
\text{Resp1} & \bullet \arrow[r, Rightarrow] & \bullet \arrow[u] \arrow[r, Rightarrow] & \bullet \arrow[r, Rightarrow] & \circ \\
\text{Resp2} & \bullet \arrow[r, Rightarrow] & \bullet \arrow[d] \arrow[r, Rightarrow] & \bullet \arrow[r, Rightarrow] & \circ \\
 & \mathsf{D} \arrow[u] & \mathsf{RR} & K\,\hat{}\,R \arrow[u] & \mathsf{depEOO} \\
 & \mathsf{D}^* \arrow[d] & \mathsf{RR}^* & \mathsf{AT}^* \arrow[d] & \mathsf{depAT} \\
\text{Resp3} & \bullet \arrow[r, Rightarrow] & \bullet \arrow[u] \arrow[r, Rightarrow] & \bullet \arrow[r, Rightarrow] & \circ
\end{tikzcd}$$

$$\mathsf{EOO}^* = \llbracket\, \mathsf{eootag}\,\hat{}\,\mathsf{h}(A\,\hat{}\,B\,\hat{}\,T\,\hat{}\,\mathsf{h}(e_1)\,\hat{}\,x)\,\hat{}\,e_2\,\rrbracket_A$$

$$\mathsf{EOR}^* = \llbracket\, \mathsf{eortag}\,\hat{}\,\mathsf{h}(L^*)\,\hat{}\,e_2\,\rrbracket_B$$

$$\mathsf{RR} = L\,\hat{}\,\mathsf{EK}\,\hat{}\,\mathsf{EOO}\,\hat{}\,\mathsf{EOR}\,\hat{}\,\llbracket\, \mathsf{rc\_rq}\;\; L\,\hat{}\,\mathsf{EK}\,\rrbracket_B$$

$$\mathsf{RR}^* = L^*\,\hat{}\,e_2\,\hat{}\,\mathsf{EOO}^*\,\hat{}\,\mathsf{EOR}^*\,\hat{}\,\llbracket\, \mathsf{rc\_rq}\;\; L^*\,\hat{}\,e_2\,\rrbracket_B$$

$$\mathsf{AT}^* = \llbracket\, \mathsf{ab\_cf}\;\; \llbracket\, \mathsf{ab\_rq}\;\; L^*\,\rrbracket_A\,\rrbracket_T$$

Parameters:	Resp1, Resp2:	A, B, T, M, K, R
	Resp3:	A, B, T, e_1, x, e_2

Figure 5: Responder Strands

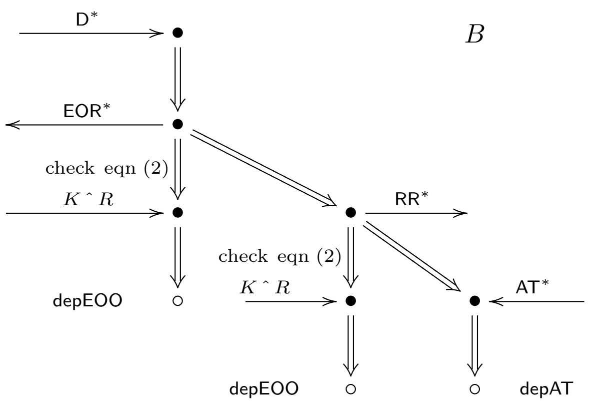


Figure 6: Responder Behavior

We may formalize this notion of refinement, and eliminate any reference to the psychology and knowledge of the principals, using the notion of similarity from Def. 2. The idea is that some but not all instances of Resp3 are similar to instances of Resp2 up to step 2. Specifically, an instance of Resp3 is similar to an instance of Resp2 up to step 2 if the parameters values for this instance satisfy the equations in Eqn 2.

Definition 3 *Let $r_1, r_2 \in \Pi$; let $\{a_1, \dots a_k, x_1, \dots, x_\ell\}$ be a parametrization of r_2; and let eqs be a set of equations of the forms $x_i = t$ or $a_i = t$.*
Role r_1 refines *role r_2* up to *j* under *eqs iff*

1. *every instance of r_1 similar to an instance of r_2 up to j, and*

2. *if α satisfies eqs, then $\alpha(r_2)$ is similar to an instance of r_1 up to j.*

Since some values of e_1, x, e_2 satisfy Eqn. 2, some prefixes of length 1 of Resp3 are also prefixes of Resp1 and Resp2. Likewise, some prefixes of length 2 of Resp3 are also prefixes of Resp2. However, B does not know which values of e_1, x, e_2 satisfy these conditions, nor for which values of M, K, R. If A's signature key is used only in accordance with Wang's protocol, then the conditions of Eqn. 2 will be satisfiable. However, if A's signature key is compromised, then B may receive messages for which these conditions are unsatisfiable.

We may express the responder strands in DAG form, as shown in Fig. 6. We use here the starred message forms of Fig. 5. We provide an accurate view of B's knowledge of the parameters by means of the edges annotated "check (2)". This annotation stipulates that the principal must check—before accepting a putative $K\,\hat{}\,R$—that the equations in Eqn. 2 are satisfied.

This check ensures that the branch it guards refines the alternative up to the index of their last common node. B can use K, R to refine his knowledge of

$$\begin{tikzcd}
\text{TtpAb1} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet \arrow[d] \\
 & & \mathsf{abrt} & \mathsf{AT}' \\
 & \mathsf{AR}' \arrow[uu] \arrow[d] & \mathsf{frcvr} & \mathsf{EOR}' \arrow[d] \\
\text{TtpAb2} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet
\end{tikzcd}$$

$$\mathsf{AR}' = [\![\, \mathsf{ab\_rq}\ \ L' \,]\!]_A \qquad \mathsf{AT}' = [\![\, \mathsf{ab\_cf}\ \ \mathsf{AR}' \,]\!]_T$$
$$\mathsf{EOR}' = [\![\, \mathsf{eortag} \,\hat{}\, \mathsf{h}(L') \,\hat{}\, e \,]\!]_B$$

Parameters:	TtpAb1:	A, B, T, y, x
	TtpAb2:	A, B, T, y, x, e

Figure 7: T's Strands for an Abort Request

the originally presented parameters e_1, x, e_2, confirming that they are messages of the expected forms. Hence, we may "substitute back" the expanded forms $\{\!|M|\!\}_K$, $\mathsf{h}(K)$, and $\{\!|\, \mathsf{keytag} \,\hat{}\, \mathsf{h}(L^*) \,\hat{}\, K |\!\}_T^R$ in place of e_1, x, e_2. This ensures that a local execution that traverses an edge marked "check (2)" in fact has the parameters demanded by a run of Resp1 or Resp2, even though B knew only, when executing the first two nodes, that its messages took the weaker starred forms.

The Cryptographic Protocol Programming Language CPPL, after the publication of [15], was equipped with a "match" statement with just the semantics of "check." As a consequence, the semantics of a CPPL procedure is exactly compatible with the semantics of DAGs with checks embedded.

It is a strength of the strand space framework that it allows a rigorous treatment of exactly what each participant knows about the parameters to a run at each point along it. This information was not present in [21], which effectively uses only the unstarred message forms. An accurate specification is crucial for providing reliable analyses of cryptographic protocols.

3.4 Specifying the Trusted Third Party

We will clarify the parameters and checks performed by the trusted third party T in the same way. In this subsection, we will annotate the messages with partially analyzed ingredients using primes, so that T's initial view of a session label L takes the form:

$$L' = A \,\hat{}\, B \,\hat{}\, T \,\hat{}\, y \,\hat{}\, x.$$

That is, neither of the hashes $\mathsf{h}(e_1), \mathsf{h}(K)$ can immediately be verified. We start first with the response to an abort request (Fig. 7). However, in handling the recovery request and the confirmation request (Fig. 8), T is given more

$$\begin{tikzcd}
\text{TtpCf1} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet \arrow[d] \\
& \mathsf{CF}'' \arrow[u] & \mathsf{rcvr} & \llbracket\, \mathsf{CF}'' \,\rrbracket_T \\
& \mathsf{RR}'' \arrow[d] & & K \,\hat{}\, R \\
\text{TtpRc1} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet \arrow[u] \\
\text{TtpRc2} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet \arrow[dd] \\
& \mathsf{RR}'' \arrow[u] & & \\
& & \mathsf{fabrt} & \mathsf{AT}'' \\
& \mathsf{CF}'' \arrow[d] & & \\
\text{TtpCf2} & \bullet \arrow[r, Rightarrow] & \circ \arrow[r, Rightarrow] & \bullet \arrow[uu]
\end{tikzcd}$$

$$L'' = A \,\hat{}\, B \,\hat{}\, T \,\hat{}\, y \,\hat{}\, \mathsf{h}(K) \qquad \mathsf{EK}'' = \{\!|\, \mathsf{keytag} \,\hat{}\, \mathsf{h}(L'') \,\hat{}\, K \,|\!\}_T^R$$

$$\mathsf{EOO}'' = \llbracket\, \mathsf{eootag} \,\hat{}\, \mathsf{h}(L'') \,\hat{}\, \mathsf{EK}'' \,\rrbracket_A \qquad \mathsf{EOR}'' = \llbracket\, \mathsf{eortag} \,\hat{}\, \mathsf{h}(L'') \,\hat{}\, \mathsf{EK}'' \,\rrbracket_B$$

$$\mathsf{RR}'' = L'' \,\hat{}\, \mathsf{EK}'' \,\hat{}\, \mathsf{EOO}'' \,\hat{}\, \mathsf{EOR}'' \,\hat{}\, \llbracket\, \mathsf{rc\_rq} \;\; L'' \,\hat{}\, \mathsf{EK}'' \,\rrbracket_B$$

$$\mathsf{CF}'' = L'' \,\hat{}\, \mathsf{EK}'' \,\hat{}\, \mathsf{EOO}'' \,\hat{}\, \mathsf{EOR}'' \,\hat{}\, \llbracket\, \mathsf{cf\_rq} \;\; L'' \,\hat{}\, \mathsf{EK}'' \,\rrbracket_A$$

$$\mathsf{AR}'' = \llbracket\, \mathsf{ab\_rq} \;\; L'' \,\rrbracket_A \qquad \mathsf{AT}'' = \llbracket\, \mathsf{ab\_cf} \;\; \mathsf{AR}'' \,\rrbracket_T$$

Parameters:	TtpCf1, TtpRc1:	A, B, T, R, K, y
	TtpCf2, TtpRc2:	A, B, T, R, K, y

Figure 8: T's Strands for a Recovery or Confirmation Request

information from which to recover the parameters, and looks more carefully at them.

4 Security Properties of Wang's Protocol

In this section, we will summarize the security properties that Wang's protocol achieves in three lemmas.

We will write $\text{Init1}_3(n_1, A, B, T, M, K, R)$, for instance, to mean that n_1 is a node of the form $s \downarrow 3$, where s is an Init1 strand with the parameters A, B, T, M, K, R. That is, the name indicates the role of the strand, and the subscript indicates the position of the node on the strand. The first argument to the predicate is the node being described, and the remaining arguments are the values of the parameters with which the role has been instantiated. As another example, $\text{Ttprc1}_3(n', A, B, T, y, K, R)$ means that n' is of the form $s' \downarrow 3$, where s' is a TtpRc1 strand with the parameters A, B, T, y, K, R.

Notice that some of these formulas are equivalent. Since each role Init$i$

begins by sending a message of the same form, $\mathrm{Init1}_1(n_1, A, B, T, M, K, R)$ is equivalent to

$$\mathrm{Init2}_1(n_1, A, B, T, M, K, R) \text{ and to } \mathrm{Init3}_1(n_1, A, B, T, M, K, R).$$

That is, if s and s' are instances of the roles Init1 and Init2 with the same parameters A, B, T, M, K, R, then $(s \downarrow 1) \sim (s' \downarrow 1)$. Bundles differing only in having one versus the other node are isomorphic, and therefore should certainly satisfy the same formulas. As another example, $\mathrm{Resp1}_1(n_1, A, B, T, M, K, R)$ holds iff

$$\mathrm{Resp3}_1(n_1, A, B, T, \{\!|M|\!\}_K, \mathsf{h}(K), \{\!| \mathsf{keytag} \mathbin{\hat{}} L \mathbin{\hat{}} K |\!\}_T^R),$$

since the first node of a Resp3 strand in which the last parameters take this particular form has received the same message as a Resp1 strand with parameters A, B, T, M, K, R. The nodes are similar.

We also write $\mathsf{Unq}(x)$ to say that x is uniquely originating, and $\mathsf{Non}(x)$ to say that x is non-originating. This provides a language for each protocol Π similar to the language $\mathcal{L}(\Pi)$ studied in [14], although slightly more expressive.

Disclosure only when authorized. Lemma 4 states that the message M is disclosed only if there has been node 3 either of an Init1 strand or of a TtpRc1 strand, with matching parameters. In particular, if B deposits M with evidence of origin into its state repository, then either A or T has reached node 3. This may be regarded as a confidentiality goal: It says that M is not disclosed unless it is *released* by one of the authorizing events, which are the third node of a Init1 or TtpRc1 strand.

Lemma 4 *Let $\mathcal{B}$ be a bundle in which:*

- $\mathrm{Init1}_1(n_1, A, B, T, M, K, R)$
- $\mathsf{Non}(\mathsf{pubk}(T)^{-1})$*, i.e. T's private decryption key is not compromised;*
- $\mathsf{Unq}(M)$ *and* $\mathsf{Unq}(K)$*, i.e. M, K are freshly chosen; and*
- *there is a node*[2] *m with either* $\mathsf{msg}(m) = M$ *or* $\mathsf{msg}(m) = K \mathbin{\hat{}} R$.

Then $\mathcal{B}$ contains a node n_3 where either

1. $\mathrm{Init1}_3(n_3, A, B, T, M, K, R)$ *and $n_1 \Rightarrow \ldots \Rightarrow n_3$, or else*
2. $\mathrm{TtpRc1}_3(n_3, A, B, T, y, K, R)$*, where in fact $y = \mathsf{h}(\{\!|M|\!\}_K)$.*

We could prove Lemma 4 using the *authentication test theorems* [16, 14]; and these proofs would be quite routine. Alternatively, we can allow the Cryptographic Protocol Shapes Analyzer CPSA [8] to search for the minimal, essentially different skeletons satisfying the hypotheses of the lemma. CPSA, starting from

[2]Possibly an adversary node.

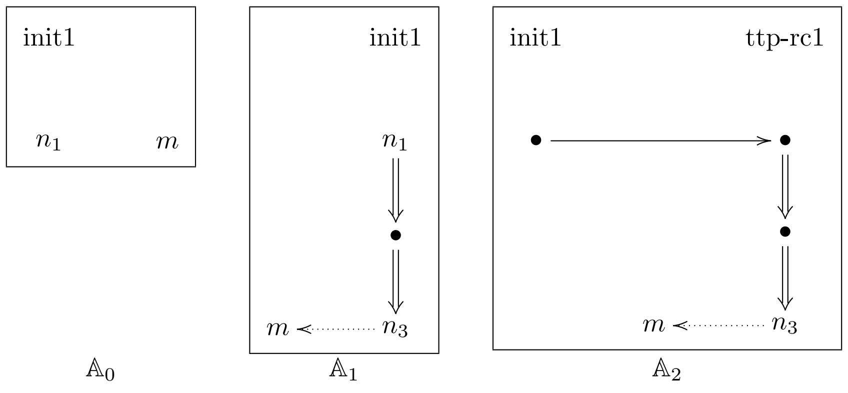


Figure 9: CPSA Run Verifying Lemma 4. $\mathsf{Non}(\mathsf{pubk}(T)^{-1})$, $\mathsf{Unq}(M), \mathsf{Unq}(K)$

the configuration described by the hypotheses, constructs the minimal, essentially different executions extending that starting point. In order to prove the Lemma, CPSA successively adds different items to the starting point $\mathbb{A}_0$, and finds that there are two different branches of the search that lead to possible executions $\mathbb{A}_1, \mathbb{A}_2$ (see Fig. 9). Each of these two executions satisfies one of the disjuncts of the conclusion.

Authenticity of evidence of origin and receipt. Lemma 5 states that the evidence of origin and receipt is sound. Specifically, if A deposits the evidence of receipt in his state repository, then B transmitted the signed evidence on node 2 of a responder strand with matching parameters. Conversely, if B deposits either evidence of origin or an abort token in his state repository, then A transmitted the initial message and signed evidence of origin, on node 1 of an initiator strand with matching parameters.

Lemma 5 *1. For all $n \in \mathcal{B}$, if:*

- *either* $\mathrm{Init1}_3(n, A, B, T, M, K, R)$*, or* $\mathrm{Init3}_3(n, A, B, T, M, K, R)$*, or* $\mathrm{Init5}_4(n, A, B, T, M, K, R)$*; and*
- $\mathsf{Non}(\mathsf{signk}(B))$*,*

then $\mathcal{B}$ contains a node m such that either:

- $\mathrm{Resp1}_2(m, A, B, T, M, K, R)$*; or*
- $\mathrm{Resp2}_2(m, A, B, T, M, K, R)$*.*

2. For all $n \in \mathcal{B}$, if:

- $\mathrm{Resp1}_3(n, A, B, T, M, K, R)$ *or* $\mathrm{Resp2}_4(n, A, B, T, M, K, R)$*; and*
- $\mathsf{Non}(\mathsf{signk}(A))$*,*

then there is an m such that $\mathrm{Init1}_1(m, A, B, T, M, K, R)$*.*

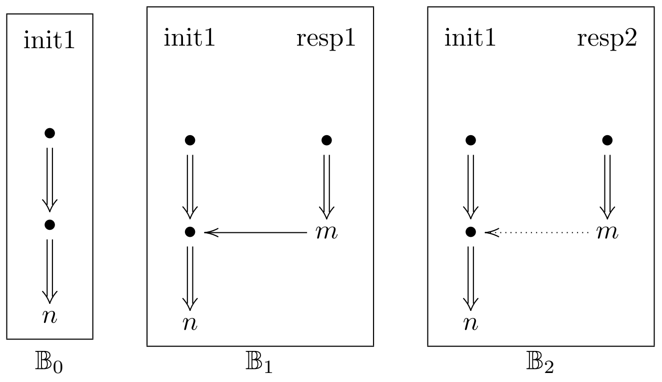


Figure 10: CPSA Analysis of Lemma 5, Clause 1, Init1. $\mathsf{Non}(\mathsf{signk}(B))$; all parameters match in $\mathbb{B}_1, \mathbb{B}_2$

```
(defskeleton wang
  (vars (a b t hash name) (m data) (r text) (k skey))
  (defstrand init1 3 (b b) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" b))
  (comment "First of two queries to prove Lemma 4.2, cl 1"))

(defskeleton wang
  (vars (a b t hash name) (m data) (r text) (k skey))
  (defstrand init3 3 (b b) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" b))
  (comment "Second of two queries to prove Lemma 4.2, cl 1"))
```

Figure 11: CPSA Inputs for Lemma 5, Clause 1

3. *For all* $n \in \mathcal{B}$*, if* $\text{Resp3}_4(n, A, B, T, e_1, x, e_2)$*, and* $\mathsf{Non}(\mathsf{signk}(A))$*, then there is an* $m \in \mathcal{B}$ *and* M, K, R *such that* $\text{Init1}_1(m, A, B, T, M, K, R)$*, and*

$$e_1 = \{\!|M|\!\}_K, \quad x = \mathsf{h}(K), \quad \textit{and } e_2 = \{\!|\,\mathsf{keytag} \,\hat{}\, L \,\hat{}\, K|\!\}_T^R.$$

We have checked this lemma by a succession of queries to CPSA. The result of the first, checking Clause 1 in the case where $\text{Init1}_3(n, A, B, T, M, K, R)$, yields the two skeletons shown in Fig. 10. Two more queries to CPSA check the cases of $\text{Resp2}_2(m, A, B, T, M, K, R)$ and $\text{Init5}_4(n, A, B, T, M, K, R)$. In all, the lemma requires six queries to CPSA.

Although this may seem a tedious way to prove theorems, it is in fact quite comfortable. The two queries for Clause 1 take the forms shown in Fig. 11. In each case, we declare the name of the protocol, some types for arguments, and associate the formal parameters for each role with its actual arguments in this query. The fact that *hash* is a principal name, as well as a, b, t, is an artifact of our representation of hashing; we represent $\mathsf{h}(t)$ as $\{\!|t|\!\}_{\mathsf{pubk(hash)}}$. Thus, $\mathsf{h}(t)$ is represented as the result of encrypting with the public encryption key of the principal *hash*. Since this principal never decrypts anything, if in addition $\mathsf{Non}(\mathsf{pubk}(\mathsf{hash})^{-1})$, then principals can create and compare hashes, but never extract values from them. In our figures here, we will not restate the assumption $\mathsf{Non}(\mathsf{pubk}(\mathsf{hash})^{-1})$.

These queries declare B's private signing key to be non-originating, and finally comment on the purpose of the query.

The twelve queries to verify the results of this section fit in 50 non-blank, non-comment lines, and CPSA executes them all in a total of 6.1 seconds on a 2008 vintage laptop with an Intel Core 2 Duo processor at 1.6 GHz, or 5.6 seconds with parallelism enabled for its two cores. Such small problems do not benefit much from CPSA's parallel facilities, although larger problems often do.

The protocol itself is represented by 190 non-blank, non-comment lines, including auxiliary definitions.[3] We have justified this as a method for proving security theorems in [14].

Authenticity of abort request and token. Lemma 6 concerns the trusted third party. It states that if A deposits an abort token into its state repository, then T issued that token on a run with matching parameters. It also asserts that if B deposits an abort token, then A has requested that this session abort, and T has transmitted the abort token.

A manual proof by routine applications of rules for digital signatures is again possible, though we have used CPSA instead.

Lemma 6 *1. For any $n \in \mathcal{B}$, if*

$$\mathsf{msg}(n) = \mathsf{AT} = [\![\, \mathsf{ab\_cf}\ [\![\, \mathsf{ab\_rq}\ \ A \,\hat{}\, B \,\hat{}\, T \,\hat{}\, e \,\hat{}\, x \,]\!]_A \,]\!]_T,$$

and $\mathsf{Non}(\mathsf{signk}(T))$, then for some $m \in \mathcal{B}$ and value R, either:

- $\mathrm{TtpAb1}_3(m, A, B, T, y, x)$*; or*
- $\mathrm{TtpRc2}_3(m, A, B, T, y, K, R)$*; or*
- $\mathrm{TtpCf2}_3(m, A, B, T, y, K, R)$*,*

where $y = \mathsf{h}(\{\!|M|\!\}_K)$ and $x = \mathsf{h}(K)$.

2. For any $n \in \mathcal{B}$, if $\mathsf{Non}(\mathsf{signk}(T))$ and $\mathsf{Non}(\mathsf{signk}(A))$ and either

- $\mathrm{TtpAb1}_3(n, A, B, T, y, x)$*; or*
- $\mathrm{TtpRc2}_3(n, A, B, T, y, K, R)$*; or*
- $\mathrm{TtpCf2}_3(n, A, B, T, y, K, R)$*,*

[3] The protocol definition and sequence of queries are available archived on the web as `http://web.cs.wpi.edu/~guttman/spiss`. The CPSA program is available as open source under a BSD license at `http://hackage.haskell.org/package/cpsa-2.0.5`.

then for some $m \in \mathcal{B}$ and M, either

$$\mathrm{Init2}_3(m, A, B, T, M, K, R') \text{ or } \mathrm{Init4}_4(m, A, B, T, M, K, R'),$$

and $y = \mathsf{h}(\{\!|M|\!\}_K)$ and $x = \mathsf{h}(K)$.

In the last clause, the earlier TTP strands are an artifact of our formalization, which cannot represent the way that these strands actually retrieve the abort request AR from the long term state.

Clause 1 would still be true in Wang's original protocol, without our adjustment. However, it would not be sufficient to justify the fair exchange property for the original protocol. Instead, the final theorem (Thm. 23) makes some assertions about conditions under which AT is available or not available to B. In Wang's original protocol, it is only A's request AR that is delivered to B, not T's signed version AT. We certainly cannot prove the counterpart to clause 1 where we infer a node m on a TTP strand from the weaker assumption that $\mathsf{msg}(n) = \mathsf{AR} = [\![\, \mathsf{ab\_rq} \;\; A\,\hat{}\,B\,\hat{}\,T\,\hat{}\,e\,\hat{}\,x \,]\!]_A$. In the original protocol, A and B cannot know whether T's state reflects the abort.

CPSA enabled us to correct one small error in an earlier, manual proof of these properties. In Lemma 6, Clause 2, the random value R that T obtains may differ from the random value R' that A has chosen. This may occur in the cases $\mathrm{TtpRc2}_3$ and $\mathrm{TtpCf2}_3$ in which T has received the session data including a component EK'', presumably from B. However, on the extremely spare assumptions that we make in Lemma 6, this EK'' does not have to be the one that A sent.

If we add the (reasonable) assumptions that $\mathsf{Unq}(M), \mathsf{Unq}(K)$ and $\mathsf{Non}(\mathsf{pubk}(T)^{-1})$, then $R = R'$. However, if M, K were transmitted on two separate strands by A, then A could choose different Rs on these two occasions, without T being able to tell which transmission lay on the same A-strand as the abort request. Moreover, if K were guessable by the adversary, then an adversary could synthesize a new encrypted key package with the same K and a different R'. If $\mathsf{pubk}(T)$ were compromised, then an adversary could decrypt the key package, obtain K, and regenerate the key package with R'. Naturally, these assumptions were in force in Lemma 4.

Our manual proofs, using the authentication test method [16, 14], led otherwise to the same results shown here.

5 Protocol Behavior and Mutable State

We formalize state change using multiset rewriting (MSR) [4, 9]. Strands contain special *state synchronization events* that synchronize them with the state of the principal executing the strands, as formalized in Definition 11.

5.1 Variables, Substitutions, and Facts

Let V be an infinite set of values disjoint from messages in the message algebra A. We generate a larger algebra from V and A in two steps:

- An *extended atom* is either (i) an atom $a \in \mathsf{A}$; (ii) a variable; or (iii) an application of one of the forms $\mathsf{pubk}(v)$, $\mathsf{signk}(v)$, v^{-1}, $\mathsf{pubk}(v)^{-1}$, or $\mathsf{signk}(v)^{-1}$ for $v \in V$.
- $\mathsf{A}[V]$, the *algebra of extended messages*, is the least set containing the extended atoms and the indeterminates X and closed under tagged concatenation and encryption.

An extended message in which no variables occur is a message in A, which we also call a *ground message*. The *free variables* $\mathsf{fv}(t)$ of an extended message t are defined as usual.

A *substitution* σ is a finite partial function $V \to \mathsf{A}$; thus, variables do not occur in the ground messages in the range of a substitution. If S_0 is the set of variables x such that some part of t is of the form $\mathsf{signk}(x)$ or $\mathsf{pubk}(x)$, then we call S_0 the *name variables* of t. If S_1 is the set of variables x such that some part of t is of the form x^{-1}, then we call S_1 the *key variables* of t. If σ is a substitution in which every name variable of t is mapped to a name, and every key variable is mapped to a key, then $t \cdot \sigma$ is well-defined, and is indeed the extended message built in the obvious way from members of the range of σ rather than the variables in its domain.

If $\mathsf{fv}(t) \subseteq \mathsf{dom}(\sigma)$ and $t \cdot \sigma$ is well-defined, then $t \cdot \sigma$ is a ground message.

We adopt a convention about assertions containing parts of the form $t \cdot \sigma$ that are possibly ill-defined. A positive atomic assertion about an extended message $t \cdot \sigma$ is true only if $t \cdot \sigma$ is well defined. A negative assertion such as "$t \cdot \sigma$ is not ground" or "If $t \cdot \sigma$ is ground, then …" is true if $t \cdot \sigma$ is not well defined [11].

We assume given a set of predicate symbols $F, G, \ldots$, each of a fixed arity. If the arity of F is k and $t_1, \ldots, t_k$ are extended messages, then $F(t_1, \ldots, t_k)$ is an *atomic formula*. If $t_1, \ldots, t_k$ are ground messages, then $F(t_1, \ldots, t_k)$ is also a *fact*, i.e. a ground atomic formula.

We write ϕ, ψ, etc. for atomic formulas; Γ, Δ, etc. for multisets of atomic formulas; and Σ, Σ_0, etc. for multisets of facts. We use the comma to form the multiset union Γ, Δ from Γ and Δ.

A *state* Σ is a multiset of facts.

5.2 Multiset rewriting to maintain state

A *rewrite rule* or simply a *rule* ρ takes the form:

$$\Gamma \xrightarrow{\phi} \Delta \qquad \text{where}\, \mathsf{fv}(\Gamma, \Delta) \subseteq \mathsf{fv}(\phi).$$

We also require:

- the name variables of Γ, Δ are included among those of ϕ;
- the key variables of Γ, Δ are included among those of ϕ.

Thus, if $\phi \cdot \sigma$ is well defined, so are $\Gamma \cdot \sigma$ and $\Delta \cdot \sigma$. Moreover, if $\phi \cdot \sigma$ is ground, so are $\Gamma \cdot \sigma$ and $\Delta \cdot \sigma$. We write $\mathsf{lhs}(\rho)$ for Γ; and $\mathsf{rhs}(\rho)$ for Δ; and $\mathsf{lab}(\rho)$ for ϕ.

Labeling the arrow with an atomic formula ϕ is in contrast with [9]. Also unlike [9], we will not require existential quantifiers in the conclusions of rules.

A rule stipulates that the state can change by consuming instances of the facts in its left-hand side, and producing the corresponding instances of the facts in its right hand side. These sets of facts may overlap, in which case the facts in the overlap are required for the rule to apply, but preserved when it executes. The corresponding instance of the label allows us to correlate the state change with some strand's state synchronization event labeled by the same fact.

When $\mathsf{lab}(\rho) \cdot \sigma$ is ground, ρ *applies to* a state Σ_0 *under* σ if

$$\Sigma_0 = \Sigma_0', (\Gamma \cdot \sigma);$$

i.e., Σ_0 is the multiset union of some Σ_0' with instances of the premises of $\mathsf{lhs}(\rho)$ under σ. Then the *result* of applying ρ to Σ_0, using σ, is

$$\Sigma_0', (\Delta \cdot \sigma).$$

By the definition of a rule ρ above, the result $\Sigma_0', \Delta \cdot \sigma$ is also a state, namely a multiset of *ground* atomic formulas.

Δ may contain variables that were not free in Γ. From the point of view of the prior state Σ_0, these variables take values nondeterministically. In an execution, they may be determined by protocol activities synchronized with the state. When a variable in ϕ *does* appear in a formula ψ in Γ, then the prior state Σ_0 is placing a constraint on the protocol activities: They can proceed only for labels $\phi \cdot \sigma$ such that $\psi \cdot \sigma \in \Sigma_0$. Thus, σ, by summarizing all choices of values for variables in ρ, determines which parameters of the protocol execution can help determine the next state Σ_1, and which parameters of facts in the current state Σ_0 can help constrain the protocol execution.

Definition 7 *Let* $\rho = \Gamma \xrightarrow{\phi} \Delta$.

1. $\Sigma_0 \xrightarrow{\rho,\sigma} \Sigma_1$ *is a* ρ, σ transition from Σ_0 to Σ_1 *iff*

   (a) Σ_0, Σ_1 *are ground, and*

   (b) there exists a Σ_0' *such that* $\Sigma_0 = \Sigma_0', (\Gamma \cdot \sigma)$*, and* $\Sigma_1 = \Sigma_0', (\Delta \cdot \sigma)$.

   A transition ρ, σ *is* enabled *in* Σ_0 *iff for some* Σ_1, $\Sigma_0 \xrightarrow{\rho,\sigma} \Sigma_1$.

2. *A* computation $\mathcal{C}$ *is a finite path through states via transitions; i.e.*

$$\mathcal{C} = \Sigma_0 \xrightarrow{\rho_0,\sigma_0} \Sigma_1 \xrightarrow{\rho_1,\sigma_1} \ldots \xrightarrow{\rho_j,\sigma_j} \Sigma_{j+1}.$$

3. $\mathcal{C}$ *is* over *a set of rules* R *if each* $\rho_i \in R$.

When no ambiguity results, we will also write $\mathcal{C}$ in the form:

$$\mathcal{C} = \Sigma_0 \xrightarrow{\phi_0 \cdot \sigma_0} \Sigma_1 \xrightarrow{\phi_1 \cdot \sigma_1} \ldots \xrightarrow{\phi_j \cdot \sigma_j} \Sigma_{j+1}.$$

We write $\mathsf{first}(\mathcal{C})$ *for* Σ_0 *and* $\mathsf{last}(\mathcal{C})$ *for* Σ_{j+1}.

Writing $\setminus, \cup, \subseteq$ for the multiset difference, union, and subset operators, we have:

Lemma 8 *1. A transition ρ, σ is enabled in Σ iff $\mathsf{lhs}(\rho) \cdot \sigma \subseteq \Sigma$.*

2. If $\Sigma_0 \xrightarrow{\rho,\sigma} \Sigma_1$, then $\Sigma_1 = (\Sigma_0 \setminus \mathsf{lhs}(\rho) \cdot \sigma) \cup \mathsf{rhs}(\rho) \cdot \sigma$.

3. If $(\mathsf{lhs}(\rho_1) \cdot \sigma_1) \cup (\mathsf{lhs}(\rho_2) \cdot \sigma_2) \subseteq \Sigma_0$, then

$$\Sigma_0 \xrightarrow{\rho_1,\sigma_1} \Sigma_1 \xrightarrow{\rho_2,\sigma_2} \Sigma_2 \quad \textit{and} \quad \Sigma_0 \xrightarrow{\rho_2,\sigma_2} \Sigma_1' \xrightarrow{\rho_1,\sigma_1} \Sigma_2$$

where

$$\begin{array}{rcl} \Sigma_1 & = & (\Sigma_0 \setminus \mathsf{lhs}(\rho_1) \cdot \sigma_1) \cup \mathsf{rhs}(\rho_1) \cdot \sigma_1 \\ \Sigma_1' & = & (\Sigma_0 \setminus \mathsf{lhs}(\rho_2) \cdot \sigma_2) \cup \mathsf{rhs}(\rho_2) \cdot \sigma_2 \\ \Sigma_2 & = & ((\Sigma_0 \setminus \mathsf{lhs}(\rho_1) \cdot \sigma_1) \setminus \mathsf{lhs}(\rho_2) \cdot \sigma_2) \cup \mathsf{rhs}(\rho_1) \cdot \sigma_1 \cup \mathsf{rhs}(\rho_2) \cdot \sigma_2 \end{array}$$

Proof. Immediate from the definitions. □

5.3 Locality to principals

In our manner of using MSR, all manipulation of state is local to a particular principal, and coordination among different principals occurs only through protocol behavior represented on strands.

Definition 9 *A set of rewrite rules R is* localized to principals via the variable *p iff, for every rule $\rho \in R$, every atomic formula in $\mathsf{lhs}(\rho)$, $\mathsf{lab}(\rho)$, or $\mathsf{rhs}(\rho)$ takes the form*

$$F(p, t_1, \ldots, t_k).$$

When R is localized to principals via p and $\rho \in R$, the principal *of a transition $\Sigma_0 \xrightarrow{\rho,\sigma} \Sigma_1$ is $p \cdot \sigma$.*

Thus, only the principal of a transition $\Sigma_0 \xrightarrow{\rho,\sigma} \Sigma_1$ is affected by it. Transitions with different principals are always concurrent. If $p \cdot \sigma_1 \neq p \cdot \sigma_2$ and $(\rho_1, \sigma_1), (\rho_2, \sigma_2)$ can happen, so can the reverse, with the same effect:

Corollary 10 *Let R be localized to principals via p, with $\rho_1, \rho_2 \in R$, and $p \cdot \sigma_1 \neq p \cdot \sigma_2$. If $\Sigma_0 \xrightarrow{\rho_1,\sigma_1} \Sigma_1 \xrightarrow{\rho_2,\sigma_2} \Sigma_2$, then $\Sigma_0 \xrightarrow{\rho_2,\sigma_2} \Sigma_1' \xrightarrow{\rho_1,\sigma_1} \Sigma_2$, for some Σ_1'.*

Proof. Since $p \cdot \sigma_1 \neq p \cdot \sigma_2$, the facts on the right hand side of $\rho_1 \cdot \sigma_1$ are disjoint from those on the left hand side of $\rho_2 \cdot \sigma_2$. Hence, ρ_2, σ_2 being enabled in Σ_1, it must also be enabled in Σ_0. Hence, $(\mathsf{lhs}(\rho_1) \cdot \sigma_1) \cup (\mathsf{lhs}(\rho_2) \cdot \sigma_2) \subseteq \Sigma_0$, and we may apply Lemma 8, Clause 3. □

The following definition connects bundles with computations.

Definition 11 *Let R be localized to principals via p.*

1. *An* eventful protocol Π *is a finite set of strands, called the* roles *of the protocol. They contain nodes of three kinds:*

   (a) transmission nodes $+t$, where t is a message;

   (b) reception nodes $-t$, where t is a message; and

   (c) state synchronization events, facts $E(p, t_1, \ldots, t_k) \cdot \sigma$, i.e. ground atomic formulas.

   We require that if $E(p, t_1, \ldots, t_k) \cdot \sigma$ and $F(p, t'_1, \ldots, t'_j) \cdot \sigma'$ lie on the same strand, then $p \cdot \sigma = p \cdot \sigma'$. If s contains event $E(p, t_1, \ldots, t_k) \cdot \sigma$, then p is the principal of *s.*

   The regular strands *of Π are all instances $\alpha(r)$ of roles $r \in \Pi$, using all homomorphisms α on the message algebra A.*[4]

2. *Suppose that $\mathcal{B}$ is a bundle over the eventful protocol Π; $\mathcal{C}$ is a finite computation for the rules R; and Φ is a bijection between state synchronization nodes of $\mathcal{B}$ and indices i of the computation $\mathcal{C}$.*

   $\mathcal{B}$ is compatible with $\mathcal{C}$ under Φ *iff*

   (a) $\mathsf{lab}(\rho_i) \cdot \sigma_i$ is the event $E(p, t_1, \ldots, t_k)$ at n, whenever (i) n is a state synchronization event; (ii) $\Phi(n) = i$, and (iii) the i^{th} transition of $\mathcal{C}$ is ρ_i, σ_i; and

   (b) $n_0 \preceq_{\mathcal{B}} n_1$ implies $\Phi(n_0) \leq \Phi(n_1)$.

3. *An* execution *of Π* constrained by *R is a triple $(\mathcal{B}, \mathcal{C}, \Phi)$ where $\mathcal{B}$ is compatible with $\mathcal{C}$ under Φ.*

If $(\mathcal{B}, \mathcal{C}, \Phi)$ is an execution, then it represents possible protocol behavior $\mathcal{B}$ for Π, where state-sensitive steps are constrained by the state maintained in $\mathcal{C}$. Moreover, the state $\mathcal{C}$ evolves as driven by state synchronizations occurring in strands appearing in $\mathcal{B}$. The bijection Φ makes explicit the correlation between events in the protocol runs of $\mathcal{B}$ and transitions occurring in $\mathcal{C}$. We do not expect the converse of Clause 2b always to be true, because $\mathcal{C}$ is linearly ordered, whereas $\preceq_{\mathcal{B}}$ may be only a partial ordering.

5.4 States and Rules for Wang's Protocol

Trusted Third Party State. Conceptually, the trusted third party T_0 maintains a status record for each possible transaction it could be asked to

[4]The homomorphisms α act on messages in A (and on objects built from them such as strands), while substitutions σ act on extended messages in $\mathsf{A}[V]$, and on objects built from them such as formulas and multisets of formulas. A ground message $t \cdot \sigma$ in A is still not a "constant value" in the sense that homomorphisms α can still act on it, yielding differing results $\alpha(t \cdot \sigma)$.

abort or recover. Since each transaction is determined by a label $\mathcal{L}_m(hm, hk) = A\,\hat{}\,B\,\hat{}\,T\,\hat{}\,hm\,\hat{}\,hk$, where $T = T_0$, it maintains a fact for each such value. This fact indicates either (1) that the no message has as yet been received in connection with this session; or (2) that the session has been recovered, in which case the evidence of receipt is also kept in the record; or (3) that the session has been aborted, in which case the signed abort request is also kept in the record. Thus, the state record for the session with label L' is a fact of one of the three forms:

$$\mathsf{unseen}(T, L') \qquad \mathsf{recovered}(T, L', \mathsf{EOR}'') \qquad \mathsf{aborted}(T, L', \mathsf{AT}')$$

Naturally, a programmer will maintain a sparse representation of this state, in which only the last two forms are actually stored. A query for ℓ that retrieves nothing indicates that the session ℓ is as yet unseen. This programming strategy requires that an unseen fact and a recovered or aborted fact never need to be stored for the same session, and this invariant is ensured by Lemma 13, Clause 1.

We name predicates appearing in facts in the state by full words, formed using past participles, namely unseen, recovered, aborted. We name predicates used as events, i.e. in the formula labeling transitions, using contractions formed from present tense verbs, namely rcvr, abrt and their forced correlates frcvr, fabrt.

Four types of events synchronize with T's state. First, events of the form $\mathsf{rcvr}(T, \ell, e)$ deposit $\mathsf{recovered}(T, \ell, e)$ facts into the state. They require the state to contain an $\mathsf{unseen}(T, \ell)$ fact or a preexisting $\mathsf{recovered}(T, \ell, e)$ fact with the same e, which are consumed.

$$\mathsf{unseen}(T, \ell) \xrightarrow{\mathsf{rcvr}(T,\ell,e)} \mathsf{recovered}(T, \ell, e) \qquad (r1)$$

$$\mathsf{recovered}(T, \ell, e) \xrightarrow{\mathsf{rcvr}(T,\ell,e')} \mathsf{recovered}(T, \ell, e) \qquad (r2)$$

The second of these forms ensures that repeated rcvr events succeed, with no further state change. Indeed, the primed variable e' in rule (r2) allows the event to occur even if the new value for e' is distinct from the old one. This could in fact happen if the initiator A starts two sessions for the same M and K, but using different random values R, R'. Rule (r2) retains the old signed value e. In an execution where A originates M or K uniquely, then the rule (r2) will never be executed with a value of e' distinct from the value of e stored in the state.

In rule (r1), the value of e is passed from a strand engaging in the state synchronization event $\mathsf{rcvr}(T, \ell, e)$ into the state. In (r2), no value is passed in either direction, but the presence of a fact $\mathsf{recovered}(T, \ell, e)$ allows the strand executing $\mathsf{rcvr}(T, \ell, e')$ to proceed.

The event $\mathsf{abrt}(T, \ell, a)$ deposits a $\mathsf{aborted}(T, \ell, a)$ fact into the state, and requires the state to contain either an $\mathsf{unseen}(T, \ell)$ fact or a preexisting $\mathsf{aborted}(T, \ell, a)$ fact, which are consumed.

$$\mathsf{unseen}(T, \ell) \xrightarrow{\mathsf{abrt}(T,\ell,a)} \mathsf{aborted}(T, \ell, a) \qquad (a1)$$

$$\mathsf{aborted}(T, \ell, a) \xrightarrow{\mathsf{abrt}(T,\ell,a')} \mathsf{aborted}(T, \ell, a) \qquad (a2)$$

The variable a' in (a2) plays a rule similar to that of e' in (r2); it allows execution to proceed with no state change even when the two abort requests were prepared with different random factors.

Finally, there is an event for a forced recover $\mathsf{frcvr}(T, \ell, e)$ and one for a forced abort $\mathsf{fabrt}(T, \ell, a)$. These may occur when the $\mathsf{recovered}$ fact [or respectively, the $\mathsf{aborted}$ fact] is already present, so that attempt to abort [or respectively, to recover] must yield the opposite result, as in Fig. 3.

$$\mathsf{recovered}(T, \ell, e) \xrightarrow{\mathsf{frcvr}(T,\ell,e)} \mathsf{recovered}(T, \ell, e) \qquad (f1)$$

$$\mathsf{aborted}(T, \ell, a) \xrightarrow{\mathsf{fabrt}(T,\ell,a)} \mathsf{aborted}(T, \ell, a) \qquad (f2)$$

In rule (f1), a TTP strand requesting an abort is forced to issue a recovery token instead to the initiator A, because the state indicates that B has already requested recovery. Thus, the state blocks continued execution of a TTP TtpAb1 strand for ℓ, and permits only a TtpAb2 strand for ℓ to proceed. In this rule, the value e is also passed out of the state and back to the requesting strand, thereby determining what signed EOR is returned to A. In (f2), a is passed back to the ongoing TtpRc2 strand, determining what AR to return within the TTP's signed AT.

Definition 12 *A* GW initial state *is a multiset* Σ *such that:*

1. *No fact* $\mathsf{recovered}(T, \ell, e)$ *or* $\mathsf{aborted}(T, \ell, a)$ *is present in* Σ*;*

2. *The multiplicity* $|\,\mathsf{unseen}(T, \ell)|_\Sigma$ *of any* $\mathsf{unseen}(T, \ell)$ *in* Σ *is at most 1.*

$\mathcal{C}$ *is a* GW computation *if it is a computation using the set* R_W *of the six rules above, starting from a* GW *initial state* Σ_0.

There are several consequences of the definitions. The first says that the multiplicity of facts for a single session ℓ does not increase, and initially starts at 0 or 1, concentrated in $\mathsf{unseen}(T, \ell)$. The next two say that a $\mathsf{recovered}(T, \ell, e)$ fact arises only after a $\mathsf{rcvr}(T, \ell, e)$ event, and a $\mathsf{aborted}(T, \ell, a)$ fact after an $\mathsf{abrt}(T, \ell, e)$ event. The fourth points out that a $\mathsf{rcvr}(T, \ell, e)$ event and an $\mathsf{abrt}(T, \ell, a)$ event never occur in the same computation. Finally, a $\mathsf{rcvr}(T, \ell, e)$ event must precede a $\mathsf{frcvr}(T, \ell, e)$ event, and likewise for aborts and forced aborts.

Lemma 13 *Let* $\mathcal{C} = \Sigma_0 \overset{\rho_0,\sigma_0}{\longrightarrow} \Sigma_1 \overset{\rho_1,\sigma_1}{\longrightarrow} \dots \overset{\rho_j,\sigma_j}{\longrightarrow} \Sigma_{j+1}$ *be a* GW *computation.*

1. *For any* ℓ *and* $i \le j+1$*, the sum over all* e, a *of the multiplicities of all facts* $\mathsf{unseen}(T, \ell)$*,* $\mathsf{recovered}(T, \ell, e)$*,* $\mathsf{aborted}(T, \ell, a)$ *is unchanged:*

$$\begin{aligned} 1 \ge |\,\mathsf{unseen}(T, \ell)|_{\Sigma_0} \;&=\; \sum_{a,e} \big(\,|\,\mathsf{unseen}(T, \ell)|_{\Sigma_i} + |\,\mathsf{recovered}(T, \ell, e)|_{\Sigma_i} \\ &+\; |\,\mathsf{aborted}(T, \ell, a)|_{\Sigma_i}\,\big). \end{aligned}$$

2. $|\,\mathsf{unseen}(T,\ell)|_{\Sigma_k}$ *is a non-increasing function of* k.

3. $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_k}$ *and* $|\,\mathsf{aborted}(T,\ell,a)|_{\Sigma_k}$ *are non-decreasing functions of* k.

4. *If* $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_i}=1$*, then there is a* j *with* $0<j<i$ *such that:*

   (a) $\mathsf{lab}(\rho_j)\cdot\sigma_j=\mathsf{rcvr}(T,\ell,e)$*;*

   (b) *For all* k *s.t.* $0\le k\le j$, $|\,\mathsf{unseen}(T,\ell)|_{\Sigma_k}=1$*;*

   (c) *For all* $k>j$, $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_k}=1$.

5. *If* $|\,\mathsf{aborted}(T,\ell,e)|_{\Sigma_i}=1$*, then there is a* j *with* $0<j<i$ *such that:*

   (a) $\mathsf{lab}(\rho_j)\cdot\sigma_j=\mathsf{abrt}(T,\ell,e)$*;*

   (b) *For all* k *s.t.* $0\le k\le j$, $|\,\mathsf{unseen}(T,\ell)|_{\Sigma_k}=1$*;*

   (c) *For all* $k>j$, $|\,\mathsf{aborted}(T,\ell,e)|_{\Sigma_k}=1$.

6. $\forall i,k,a,e$*, not both*

$$\begin{array}{rl} & \mathsf{lab}(\rho_i)\cdot\sigma_i=\mathsf{rcvr}(T,\ell,e)\ \textit{or}\ \mathsf{frcvr}(T,\ell,e)\\ \textit{and} & \mathsf{lab}(\rho_k)\cdot\sigma_k=\mathsf{abrt}(T,\ell,a)\ \textit{or}\ \mathsf{fabrt}(T,\ell,a)\end{array}$$

Proof. 1. The first inequality $1\ge|\,\mathsf{unseen}(T,\ell)|_{\Sigma_0}$ holds by the definition of GW initial. The equality with the sum holds inductively, because each of the rules consumes one fact of one of these forms, and regenerates one fact of one of these forms.

2. No rule produces a formula $\mathsf{unseen}(T,\ell)$ in its rhs.

3. Each rule that consumes a formula $\mathsf{recovered}(T,\ell,e)$ or $\mathsf{aborted}(T,\ell,a)$ in its lhs regenerates that formula in its rhs.

4. By the definition of GW initial, $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_0}=0$, so let j_0 be the least number such that in $\Sigma_{(j_0+1)}$ we have $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_{(j_0+1)}}=1$. By taking cases on the rules, we see that ρ_{j_0} is an instance of rule $r1$. Hence its label is $\mathsf{rcvr}(T,\ell,e)$. Moreover, $|\,\mathsf{unseen}(T,\ell)|_{\Sigma_{(j_0)}}=1$. By clauses 2–3, $\mathsf{unseen}(T,\ell)$ was always previously present, and $\mathsf{recovered}(T,\ell,e)$ is always subsequently present.

5. By symmetry, interchanging aborts and recovers.

6. Suppose i and k are such. Choosing $j>\max(i,k)$, by Clause 2 $|\,\mathsf{recovered}(T,\ell,e)|_{\Sigma_j}=1$. By Clause 3 $|\,\mathsf{aborted}(T,\ell,e)|_{\Sigma_j}=1$. Hence their sum is 3, contradicting Clause 1. □

Initiator and Responder State. The initiator and responder have rules with empty precondition, that simply deposit record values into their state. These records are of the forms $\mathsf{eor}(A,L,\mathsf{EOR},M,K,R)$, $\mathsf{eoo}(B,L,\mathsf{EOO},M,K,R)$, and $\mathsf{aborted}(P,\ell,[\![\,[\![\,\mathsf{ab\_rq}\ \hat{}\ \mathsf{h}(\ell)\,]\!]_A\,]\!]_T)$. Of these, the last is used both by the initiator and the responder, with parameter $\ell = L$ for the initiator and $\ell = L^* = A\,\hat{}\,B\,\hat{}\,T\,\hat{}\,\mathsf{h}(e_1)\,\hat{}\,x$ for the responder (see p. 15). The rules are:

$$\cdot \quad \xrightarrow{\mathsf{depEOR}(A,\ell,e,M,K,R)} \mathsf{eor}(A,\ell,e,M,K,R)$$

$$\cdot \quad \xrightarrow{\mathsf{depEOO}(B,\ell,e,M,K,R)} \mathsf{eoo}(B,\ell,e,M,K,R)$$

$$\cdot \quad \xrightarrow{\mathsf{depAT}(P,\ell,a)} \mathsf{aborted}(P,\ell,a)$$

6 Progress Assumptions

We introduce two kinds of progress properties for protocols. One of them formalizes the idea that certain messages, if sent, must be delivered to a regular participant, i.e. that these messages traverse resilient channels. The second is the idea that principals, at particular nodes in a strand, must progress. These are either nodes where they could transmit a message with guaranteed delivery, or else nodes where they could engage in an enabled state synchronization event.

6.1 Guaranteed Delivery

Definition 14 *1. G is a set of* guaranteed delivery assumptions *for* Π *iff* G *is a set of transmission nodes $r \downarrow i$, where each r is a role of* Π.

2. A transmission node $n = s' \downarrow i$ is a guaranteed delivery node *for* Π, G *iff for some α and some $r \in \Pi$, $s' = r \cdot \alpha$ and $r \downarrow i \in G$.*

3. A bundle $\mathcal{B}$ satisfies guaranteed delivery *for* Π, G *iff, for every guaranteed delivery node $n \in \mathcal{B}$,*

if *there exists a regular reception node m_1 of* Π *such that* $\mathsf{msg}(m_1) = \mathsf{msg}(n)$ *and either:*

(a) $m_1 = s \downarrow 1$; or

(b) for some regular node $m_0' \in \mathcal{B}$, there is an $m_0 \sim m_0'$ such that $m_0 \Rightarrow m_1$,

then *there is exactly one regular node $m \in \mathcal{B}$ such that $n \rightarrow_{\mathcal{B}} m$.*

Thus, guaranteed delivery says that a transmission is received if doing so requires only (i) adding the first node of a new strand, or else (ii) switching a strand with a node in $m_0' \in \mathcal{B}$ with a *similar* strand containing m_0 and extending it one more step to m_1. The definition of "similar" in Defn. 2 is that two nodes are similar, $n \sim m$, if their strands have sent and received the same messages, and engaged in the same state synchronizations, up to this point.

In defining guaranteed delivery, there is no question about whether the recipient node m is the right or wrong node to receive the message transmitted by n. Since m is a regular node, its role, and the parameters determined on any earlier nodes $m_0 \Rightarrow^+ m$, constrain what messages it can receive. The guaranteed delivery condition just ensures that some one regular node, which must therefore satisfy these requirements, will receive the message. Adversary nodes are also free to receive it.

Guaranteed Delivery for Wang's Protocol The guaranteed delivery assumptions for Wang's protocol are not surprising. They are the messages transmitted on resilient channels between the principals and the Trusted Third Party. These are A's transmission of AR and B's transmission of RR in Fig. 2, and T's six transmissions in Fig. 3.

6.2 Transmission Progress

The second flavor of progress concerns principals who are ready to make a transmission for a message with guaranteed delivery. A bundle satisfies transmission progress if they have always continued either to perform this transmission or else some other action that their roles permit.

Definition 15 $\mathcal{B}$ satisfies transmission progress *iff, for every $m_0 \in \mathcal{B}$, and for every regular node n_0:*

if *(i) $m_0 \sim n_0$, (ii) $n_0 \Rightarrow n_1$, and (iii) n_1 is a guaranteed delivery transm. node,*

then *there exists an $m_1 \in \mathcal{B}$ such that $m_0 \Rightarrow m_1$.*

The transmission progress property says that if we look at any terminal node of $\mathcal{B}$—i.e. any node $m_0 \in \mathcal{B}$ where there is no $m_1 \in \mathcal{B}$ such that $m_0 \Rightarrow m_1$—then m_0 cannot continue to make a guaranteed delivery transmission. "Can continue" here means that any *similar* node n_0 is immediately followed $n_0 \Rightarrow n_1$ by a guaranteed transmission node n_1.

As an example of transmission progress, suppose that a bundle $\mathcal{B}$ contains the first node n_0 of an initiator A strand. Since A may send a message with guaranteed delivery to the Ttp immediately after n_0, transmission requires that A has not stopped at n_0. Regardless of whether n_0 is the first node of an Init1 strand or the similar first node of an Init2 strand, the fact that an Init2 strand continues with a guaranteed delivery message means that A must continue to some node n_1 after n_0. If A has not received the expected message from B, and has not yet transmitted an abort request to the Ttp, then $\mathcal{B}$ does not satisfy the transmission progress property.

However, a bundle $\mathcal{B}$ containing a first responder strand node n_0 of B need not progress. Every responder strand continues with the transmission of EOR to A, which does not have guaranteed delivery. Thus, the definition permits B to stop at this point. However, it does not permit B to stop after that transmission, since a continuation after that point is the guaranteed delivery transmission to the Ttp.

In stating this lemma, we use the notion of the $\mathcal{B}$-height of a strand s. The $\mathcal{B}$-height of s is the number of nodes of s that are in $\mathcal{B}$. Equivalently, it is the

(1-based) index of the last node of s in $\mathcal{B}$. Strand s has *full height* in $\mathcal{B}$ iff its $\mathcal{B}$-height is its length. It is *stopped at* a node in $\mathcal{B}$ iff that node is $s \downarrow i$, and s's $\mathcal{B}$-height is i, but s has length $> i$. It is *stopped before* a node iff that node is $s \downarrow i+1$, and s's $\mathcal{B}$-height is i.

Lemma 16 *If $\mathcal{B}$ is a bundle for Wang's protocol, and $\mathcal{B}$ satisfies transmission progress, then the $\mathcal{B}$-height of a strand s*

is not 1 *if s an* Init*i strand;*

is not 2 *if s a* Resp*i strand, or if s is an* Init1*,* Init4*,* Init5 *strand, or if s is a* Ttp *strand.*

Proof. In all of these cases, the last node in $\mathcal{B}$ would be equal or similar to a node with a successor having guaranteed delivery. □

6.3 State Progress

Just as we do not want a strand to stop when it could perform a guaranteed delivery transmission, we also do not want it to stop when it could perform a state synchronization event. However, this is not simply a property of a bundle; whether a state synchronization event can occur also depends on the state. Thus, it is a property of an execution $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$.

Definition 17 (State Progress) *Let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be an execution. $\mathcal{E}$* satisfies state progress *for Π constrained by R iff, for every $m_0 \in \mathcal{B}$, and for every regular node n_0:*

if *(i) $m_0 \sim n_0$; (ii) $n_0 \Rightarrow n_1$; (iii) node n_1 is a state synchronization event ϕ; and (iv) for some $\rho \in R$ and σ, $\mathsf{lab}(\rho) \cdot \sigma = \phi$ and $\mathsf{lhs}(\rho) \cdot \sigma \subseteq \mathsf{last}(\mathcal{C})$;*

then *there exists an $m_1 \in \mathcal{B}$ such that $m_0 \Rightarrow m_1$.*

In the case of Wang's protocol, an execution that satisfies state progress certainly does not have an initiator or responder stopped immediately before its deposit event depEOO, depEOR, or depAT. These events have empty precondition, and therefore are enabled in every state. Thus, each strand that has reached the previous node in $\mathcal{B}$, can progress compatible with any $\mathcal{C}$, and therefore (if state progress is satisfied) has progressed and performed the deposit.

If, moreover, $\mathsf{unseen}(T, \ell) \in \mathsf{first}(\mathcal{C})$, then by Lemma 13, Clause 1, there is exactly one fact $\mathsf{unseen}(T, \ell), \mathsf{recovered}(T, \ell, \cdot), \mathsf{recovered}(T, \ell, \cdot) \in \mathsf{last}(\mathcal{C})$. Hence, if any Ttp strand is of $\mathcal{B}$-height 1, then either it can progress or its similar node can progress on the other branch shown in Fig. 3. Thus, summarizing these considerations:

Lemma 18 *Let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be an execution that satisfies state progress. If $\mathsf{unseen}(T, \ell) \in \mathsf{first}(\mathcal{C})$ and s is a strand whose parameters match the session ℓ, then the $\mathcal{B}$-height of s*

is not 1 *if s is a* Ttp *strand;*

is not 3 *if s is an* Init$1, 2, 3$ *strand, or a* Resp$i$ *strand;*

is not 4 *if s is an* Init$4, 5$ *strand.* $\square$

6.4 Stability in Wang's Protocol

Stability combines guaranteed delivery, transmission progress, and state progress:

Definition 19 (Stable) *$\mathcal{E}$ is a* stable execution *if (1) $\mathcal{B}$ satisfies guaranteed delivery for G; (2) $\mathcal{B}$ satisfies transmission progress; and (3) $\mathcal{E}$ satisfies state progress.*

In a stable execution, each strand has reached a "natural stopping point," where messages with guaranteed delivery have been delivered; no strand has stopped when a transmission with guaranteed delivery is waiting to happen; and no strand has stopped when an enabled state synchronization event is waiting to happen. We may now summarize the consequences of Lemmas 16, 18 for stable executions:

Lemma 20 *Let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be a stable execution for Wang's protocol, and let ℓ be a session label with Ttp T and $\mathsf{unseen}(T, \ell) \in \mathsf{first}(\mathcal{C})$.*

1. *If s is an initiator strand with $\mathcal{B}$-height ≥ 1, then either s has full height or s is awaiting an* AT *or* EOR *from T. In particular, if s is an* Init1 *strand with $\mathcal{B}$-height ≥ 1, then s has full height.*

2. *If s is a responder strand with $\mathcal{B}$-height ≥ 2, then either s has full height or s is awaiting a $K \hat{} R$ or* AT *from T. In particular, if s is a* Resp1 *strand with $\mathcal{B}$-height ≥ 2, then s has full height.*

3. *If s is a T Ttp strand with session label ℓ and $\mathcal{B}$-height ≥ 1, then s has full height.*

Lemma 20 is entirely local in the sense that it involves only the structure of individual strands and the state computation $\mathcal{C}$. It does not depend on any global or cross-strand properties of $\mathcal{B}$. We now use Lemma 20 and the definition of guaranteed delivery, to infer global properties of stable executions. In particular, initiator and responder strands are not stopped waiting for the Ttp. However, in this lemma, we make no use of the security goals established in Section 4, and therefore do not need any assumptions about non-origination or unique origination in $\mathcal{B}$.

Lemma 21 *Let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be a stable execution for Wang's protocol, and let ℓ be a session label with Ttp T and $\mathsf{unseen}(T, \ell) \in \mathsf{first}(\mathcal{C})$. Let*

$$S_0 = \{n \in \mathcal{B} \colon \text{ regular } n \text{ transmits a Ttp request for } \ell\}$$

be the set S of nodes making requests with session label ℓ.

1. *Letting $S_1 = \{m \in \mathcal{B}\colon m$ regular and $n \rightarrow m$ for some $n \in S_0\}$, $\rightarrow$ is a bijection between S_0 and S_1.*
2. *Letting $S_2 = \{m' \in \mathcal{B}\colon m \Rightarrow \cdot \Rightarrow m'$ for some $m \in S_1\}$ be the second strand successors of nodes in S_1, $\Rightarrow \circ \Rightarrow$ is a bijection between S_1 and S_2.*
3. *Letting $S_3 = \{n' \in \mathcal{B}\colon n'$ regular and $m' \rightarrow n'$ for some $m' \in S_2\}$, $\rightarrow$ is a bijection between S_2 and S_3.*
4. *No strand is stopped at any $n \in S_0$.*

Proof. 1. Since $\mathsf{msg}(n)$ is received on the first node of a TTP strand, the unique-regular-delivery property of guaranteed delivery entails that $\rightarrow$ is a function onto S_2. The bundle property that reception nodes have $\rightarrow$ in-degree 1 implies this function is one-to-one.

2. By Lemma 20, Clause 3, and the linearity of strands.

3. Since we know from Clauses 1, 2 that $\Rightarrow \circ \Rightarrow \circ \rightarrow$ is a bijection from S_0 to S_2, we know that each m' results from a request that is still waiting. Thus, we may apply the unique-regular-delivery property again.

4. By composition. □

Observe from this that if $\mathcal{E}$ is a stable execution and L is a session label with $\mathsf{unseen}(T, L) \in \mathsf{last}(\mathcal{C})$, then we can extend $\mathcal{E}$ to a stable execution containing also an initiator session with label L by adding (e.g.) an Init2 strand and a TtpAb1 strand of full height. If $\mathsf{unseen}(T, L) \in \mathsf{last}(\mathcal{C})$, then we can extend $\mathcal{E}$ to a stable execution containing also an initiator session with label L by adding (e.g.) a Resp2 strand and a TtpRc1 strand of full height. Thus, stable executions may be (stably) extended to contain new sessions for either initiator or responder. The extended executions do not require any additional atomic values to originate, except the parameters A, B, T, M, K, R.

This construction does not essentially depend on adding the new session at the end of $\mathcal{E}$, although we will not pause now to formalize a method to incorporate the new session in the midst of $\mathcal{E}$.

7 Correctness of Wang's protocol

No protocol can protect principals that do not follow it.

Thus, correctness concerns stable executions in which at least one of A, B comply with the protocol. We also assume that the trusted third party T merits our trust, and obeys the protocol. However, we should certainly not assume that the other party B, A complies with the protocol. The protocol must ensure a satisfactory outcome even if it cheats. This trust model follows [17].

We formalize this compliance relative to an existing execution $\mathcal{E}$ by saying that a new local initiator or responder session is *compatible* with $\mathcal{E}$ if it relates properly with the existing behavior in $\mathcal{E}$:

Definition 22 *Let n be a regular node, and let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be an execution. Node n is* compatible *with $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ iff $n = s \downarrow 1$ where either:*

- *s is an initiator strand with parameters A, B, T, M, K, R, and*
  1. $\mathsf{signk}(A), \mathsf{signk}(T), \mathsf{pubk}(T)$ *are non-originating in $\mathcal{B}$;*
  2. *M, K are non-originating in $\mathcal{B}$; and*
  3. *for $L = A \mathbin{\hat{}} B \mathbin{\hat{}} T \mathbin{\hat{}} \mathsf{h}(\{\!|M|\!\}_K) \mathbin{\hat{}} \mathsf{h}(K)$, $\mathsf{unseen}(T, L) \in \mathsf{first}(\mathcal{C})$; or else*
- *s is a* Resp1 *strand with parameters A, B, T, M, K, R, and*
  1. $\mathsf{signk}(B), \mathsf{signk}(T), \mathsf{pubk}(T)$ *are non-originating in $\mathcal{B}$; and*
  2. *for $L = A \mathbin{\hat{}} B \mathbin{\hat{}} T \mathbin{\hat{}} \mathsf{h}(\{\!|M|\!\}_K) \mathbin{\hat{}} \mathsf{h}(K)$, $\mathsf{unseen}(T, L) \in \mathsf{first}(\mathcal{C})$.*

In both cases, we assume that the active principal's signature key has been used only in accordance with the protocol in $\mathcal{B}$, and we assume that a T has been chosen whose signature key and private decryption key have been used only in accordance with the protocol. We also assume in both cases that the session label is initially unseen in $\mathcal{C}$. In the first case, we also assume that M and K have never yet been used, and in the second case we assume that the responder's parameters could lead to a successful outcome. We need not consider a Resp3 strand in which the responder receives an ill-formed EM or EK, since the initiator will then never be able to convince a judge that a message was successfully delivered. Although B certainly may not find out how EM and EK are composed, we need worry about the outcome only when they are in fact well formed. (See our comment on p. 6 and also [21, Fig. 3].)

This correctness theorem combines the elements we have introduced:

Theorem 23 (Wang's Fair Exchange) *Let $\mathcal{E} = (\mathcal{B}, \mathcal{C}, \Phi)$ be a stable* GW-*execution with $\mathsf{unseen}(T, L) \in \Sigma_0$, and let n be compatible with $\mathcal{E}$. Let $\mathcal{E}' = (\mathcal{B}', \mathcal{C}', \Phi')$ be a stable extension of $\mathcal{E}$ containing n, and in which no additional atomic values originate, apart from A, B, T, M, K, R.*

1. **If** *$n = s \downarrow 1$, where s is an initiator node with parameters A, B, T, M, K, R, and either:*
   - *(a)* $\mathsf{eoo}(B, L, \mathsf{EOO}, M, K, R) \in \mathsf{last}(\mathcal{C}')$*; or else*
   - *(b) there is $m_0 \in \mathcal{B}'$ such that* $\mathsf{msg}(m_0) = M$,

   then $\mathsf{eor}(A, L, \mathsf{EOR}, M, K, R) \in \mathsf{last}(\mathcal{C}')$*, and*
   there is no $m_1 \in \mathcal{B}'$ such that $\mathsf{msg}(m_0) = \mathsf{AT}$.

2. **If** *$n = s \downarrow 2$, where s is a responder strand, and either:*
   - *(a)* $\mathsf{eor}(A, L, \mathsf{EOR}, M, K, R) \in \mathsf{last}(\mathcal{C}')$*; or else*
   - *(b) there exist $m_0, m_1, m_2, m_3 \in \mathcal{B}'$ such that*

   $$\mathsf{msg}(m_0) = \mathsf{EOR}, \quad \mathsf{msg}(m_1) = M, \quad \mathsf{msg}(m_2) = K, \quad \textit{and} \quad \mathsf{msg}(m_3) = R,$$

   then *either*

   $$\mathsf{eoo}(B, L, \mathsf{EOO}, M, K, R) \in \mathsf{last}(\mathcal{C}'),$$

   or else

   $$\mathsf{aborted}(B, L, \mathsf{AT}) \in \mathsf{last}(\mathcal{C}').$$

Proof. We break the proof up into four cases, depending on which assumptions are in force.

1a. Since $\mathsf{eoo}(B, L, \mathsf{EOO}, M, K, R) \in \mathsf{last}(\mathcal{C}')$, a transition labeled

$$\mathsf{depEOO}(B, L, e, M, K, R)$$

has occurred in $\mathcal{C}'$. Hence, there is either a Resp1 or a Resp2 strand s' with parameters A, B, T, M, K, R of full height in $\mathcal{B}'$. In particular, there is a node $s' \downarrow 3$ receiving $K \,\hat{}\, R$.

Since n is compatible with $\mathcal{E}$, we may apply Lemma 4, to infer that there is an n_3 with either $\mathrm{Init1}_3(n_3, A, B, T, M, K, R)$ or $\mathrm{TtpRc1}_3(n_3, A, B, T, y, K, R)$. In the former case, Lemma 18 completes the proof. In the latter case, we may infer (Lemma 13) that no $\mathsf{abrt}(T, L, a)$ event occurs in $\mathcal{C}'$. Thus, s is not an Init2 strand. By Lemmas 20 and 21, s has full height as an Init1,3 strand, implying that depEOR has occurred.

If in addition there were an m_1 with $\mathsf{msg}(m_1) = \mathsf{AT}$, then by Lemma 6, Clause 1, then $\mathrm{TtpAb1}_3(m_1, A, B, T, y, x)$, $\mathrm{TtpRc2}_3(m_1, A, B, T, y, K, R)$, or $\mathrm{TtpCf2}_3(m_1, A, B, T, y, K, R)$. By Clause 2, there is then an m_2 with $\mathrm{Init2}_3(m_2, A, B, T, M, K, R)$ or $\mathrm{Init4}_4(m_2, A, B, T, M, K, R)$. However, since we are assuming M uniquely originating, this contradicts our earlier conclusion that $\mathrm{Init1}_3(n_3, A, B, T, M, K, R)$.

1b. This assumption also allows us to apply Lemma 4, with the same consequence.

2a. The assumption ensures that we can apply Lemma 5, Clause 1, so that s's $\mathcal{B}$-height is at least 2. Thus, Lemmas 20–21 imply that s has full height, and has deposited either an eoo or an AT.

2b. Similar to Part 2a. □

7.1 Conclusion

This formalism has also been found to be convenient to model the interface to a cryptographic device, the Trusted Platform Module, which combines cryptographic operations with a repository of state [20]. Thus, it appears to be a widely applicable approach to the problem of combining reasoning about cryptographic protocols with reasoning about state and histories.

Acknowledgments. Thanks to Wan Fokkink for the opportunity to present a much earlier version of this work at a workshop at the Vrije Universiteit, Amsterdam, in February 2008, and to Jan Cederquist, Mohammad Torabi Dashti, and Sjouke Mauw for their comments. Thanks also to Iliano Cervesato, whose invitation to speak at ASIAN 2007 stimulated the work described in Section 5.

The very helpful and perceptive comments of the anonymous referees led to substantial improvements.

References

[1] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE J. Sel. Areas in Comms.*, 18(4):593–610, 2000.

[2] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 138–146, New York, NY, USA, 1999. ACM.

[3] Jan Cederquist, Mohammad Torabi Dashti, and Sjouke Mauw. A certified email protocol using key chains. In *Advanced Information Networking and Applications Workshops/Symposia (AINA'07), Symposium on Security in Networks and Distributed Systems (SSNDS07)*, volume 1, pages 525–530. IEEE CS Press, 2007.

[4] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A metanotation for protocol analysis. In *Proceedings, 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[5] Rohit Chadha, John C. Mitchell, Andre Scedrov, and Vitaly Shmatikov. Contract signing, optimism, and advantage. In *Concur — Concurrency Theory*, LNCS, pages 366–382. Springer, 2003.

[6] Mohammad Torabi Dashti. *Keeping Fairness Alive.* PhD thesis, Vrije Universiteit, Amsterdam, 2007.

[7] Pierpaolo Degano, Joshua D. Guttman, and Fabio Martinelli, editors. *Formal Aspects in Security and Trust, 5th International Workshop, FAST 2008, Malaga, Spain, October 9-10, 2008, Revised Selected Papers*, volume 5491 of *Lecture Notes in Computer Science*. Springer, 2009.

[8] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 523–538. Springer, March 2007. Extended version at URL:`http://eprint.iacr.org/2006/435`.

[9] Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004. Initial version appeared in *Workshop on Formal Methods and Security Protocols*, 1999.

[10] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical Report 175, Computer Science Department, Technion, 1980.

[11] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.

[12] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 449–466, London, UK, 1999. Springer-Verlag.

[13] Joshua D. Guttman. Cryptographic protocol composition via the authentication tests. In Luca de Alfaro, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, number 5504 in LNCS, pages 303–317. Springer, March 2009.

[14] Joshua D. Guttman. Security theorems via model theory. *EPTCS: Expressiveness in Concurrency*, 8:51, 2009. doi:10.4204/EPTCS.8.5.

[15] Joshua D. Guttman, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Programming cryptographic protocols. In Rocco De Nicola and Davide Sangiorgi, editors, *Trust in Global Computing*, number 3705 in LNCS, pages 116–145. Springer, 2005.

[16] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, June 2002. Conference version appeared in *IEEE Symposium on Security and Privacy*, May 2000.

[17] Francis Klay and Laurent Vigneron. Automatic methods for analyzing non-repudiation protocols with an active intruder. In Degano et al. [7], pages 192–209.

[18] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Optimal efficiency of optimistic contract signing. In *Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 113–122, New York, May 1998. ACM.

[19] Michael Rabin. How to exchange secrets by oblivious transfer. Technical report, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981. Available at `http://eprint.iacr.org/2005/187`.

[20] Ariel Segall and Joshua Guttman. A strand space/multiset rewriting model of TPM commands. MTR 080181, The MITRE Corporation, Bedford, MA, July 2008.

[21] Guilin Wang. Generic non-repudiation protocols supporting transparent off-line TTP. *Journal of Computer Security*, 14(5):441–467, 2006.

A CPSA Definition for Wang's Protocol

We give here verbatim the CPSA input file used to check confidentiality and authentication for Wang's protocol. The corresponding output files may be found at `http://web.cs.wpi.edu/~guttman/spiss`. For instance,

`http://web.cs.wpi.edu/~guttman/spiss/wang_shapes.xml`

is a browser-readable file that summarizes all of the queries and the shapes that CPSA returns for each query.

```
;; CPSA input file to represent
;; Wang's Fair Exchange Protocol

;; This macro implements the CPSA
;; convention of regarding a hash as
;; an asymmetric encryption.  The
;; decryption key inverse to (pubk
;; hash) is assumed non-originating
;; in roles using hashing.

(defmacro (hash x)
  (enc x (pubk hash_id)))

;; Wang's encrypted format for the
;; message payload.  Observe that m,
;; k get their values from the
;; like-named messages in the context
;; where the macro is expanded.

(defmacro (em)
  (enc m k))

;; The session label L as a fn of
;; the values that *should be* the
;; hashes of the encrypted message EM
;; and the session key K.

(defmacro (l-prime y x)
  (cat a b t y x))

;; The recipient can check that a
;; session label has an l-prime where
;; y is the hash of another parameter.
;; Hence recipient roles use l-star.

(defmacro (l-star e1 x)
  (l-prime (hash e1) x))

;; The Evidence of Origin is checked
;; to be this fn of other parameters
;; available to the recipient.

(defmacro (eoo-star x e1 e2)
  (enc (cat "eootag"
    (hash (l-star e1 x)) e2)
       (privk "sign" a)))

;; The Evidence of Receipt is computed
;; as this fn of other parameters
;; available to the recipient.

(defmacro (eor-star x e1 e2)
  (enc (cat "eortag"
    (hash (l-star e1 x)) e2)
       (privk "sign" b)))

;; The Recovery Request as computed by
;; the recipient.

(defmacro (rr-star x e1 e2)
  (cat (l-star e1 x) e2
       (eoo-star x e1 e2)
       (eor-star x e1 e2)
       (enc (cat "rcrq" (l-star e1 x) e2)
            (privk "sign" b))))

;; The Abort Token is checked
;; to be this fn of other parameters
;; available to the recipient.
```

```
(defmacro (at-star e1 x)
  (enc (enc "abrq" (l-star e1 x)
            (privk "sign" a))
       (privk "sign" t)))

;; How the initiator computes the
;; encrypted key package:

(defmacro (ek l)
  (enc (cat "keytag" (hash l) k r)
       (pubk "encr" t)))

;; The Initiator's computed
;; version of L.

(defmacro (l)
  (l-prime (hash (enc m k)) (hash k)))

;; The TTP can check that the session
;; label L is this fn of other
;; parameters available to the TTP.

(defmacro (l-prime-prime y)
  (cat a b t y (hash k)))

;; The TTP can check that the EOO
;; is this fn of other parameters
;; available to the TTP.

(defmacro (eoo-prime-prime y)
  (enc (cat "eootag"
            (hash (l-prime-prime y))
    (ek (l-prime-prime y)))
       (privk "sign" a)))

;; The TTP can check that the EOR
;; is this fn of other parameters
;; available to the TTP.

(defmacro (eor-prime-prime y)
  (enc (cat "eortag"
    (hash (l-prime-prime y))
    (ek (l-prime-prime y)))
       (privk "sign" b)))

;; The TTP can check that the RR
;; is this fn of other parameters
;; available to the TTP.

(defmacro (rr-prime-prime y)
  (cat (l-prime-prime y)
       (ek (l-prime-prime y))
       (eoo-prime-prime y)
       (eor-prime-prime y)
       (enc "rcrq" (l-prime-prime y)
    (ek (l-prime-prime y))
            (privk "sign" b))))

;; The TTP can check that the Confirmation
;; Request is this fn of other parameters
;; available to the TTP.

(defmacro (cf-prime-prime y)
  (cat (l-prime-prime y) (ek (l-prime-prime y))
       (eoo-prime-prime y) (eor-prime-prime y)
       (enc "cfrq" (l-prime-prime y)
    (ek (l-prime-prime y))
            (privk "sign" b))))

;; Defn of Wang's Fair Exchange Protocol.
;; Closely follows the presentation in the
;; figures in Section 3.  In the TTP roles,
;; we have put a dummy send or receive in place
;; of the state synchronization event to keep
;; the indices matching the spec.

(defprotocol wang basic

  ;; Successful initiator run with no TTP involvement

  (defrole init1
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (send (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (recv (eor-star (hash k) (enc m k) (ek (l))))
     (send (cat k r)))
    (non-orig (privk hash_id)))

  ;; Aborted initiator run

  (defrole init2
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (send (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (send (enc "abrq" (l) (privk "sign" a)))
     (recv (enc "abcf"
                (enc "abrq" (l) (privk "sign" a))
                (privk "sign" t))))
    (non-orig (privk hash_id)))

  ;; Initiator run with abort request and forced recovery

  (defrole init3
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (send (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (send (enc "abrq" (l) (privk "sign" a)))
     (recv (eor-star (hash k) (enc m k) (ek (l)))))
    (non-orig (privk hash_id)))

  ;; Aborted initiator run, but with EOR received

  (defrole init4
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (send (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (recv (eor-star (hash k) (enc m k) (ek (l))))
     (send (enc "abrq" (l) (privk "sign" a)))
     (recv (enc "abcf"
                (enc "abrq" (l) (privk "sign" a))
                (privk "sign" t))))
    (non-orig (privk hash_id)))

  ;; Initiator run with abort requested after EOR received,
  ;; but recovery forced

  (defrole init5
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (send (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (recv (eor-star (hash k) (enc m k) (ek (l))))
     (send (enc "abrq" (l) (privk "sign" a)))
     (recv (eor-star (hash k) (enc m k) (ek (l)))))
    (non-orig (privk hash_id)))

  ;; Successful responder run with no TTP involvement

  (defrole resp1
    (vars (a b t hash_id name) (m data) (r text) (k skey))
    (trace
     (recv (cat (l) (enc m k) (ek (l))
                (eoo-star (hash k) (enc m k) (ek (l)))))
     (send (eor-star (hash k) (enc m k) (ek (l))))
     (recv (cat k r)))
    (non-orig (privk hash_id)))
```

```
;;Responder run with recovery via TTP

(defrole resp2
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (trace
   (recv (cat (l) (enc m k) (ek (l))
              (eoo-star (hash k) (enc m k) (ek (l)))))
   (send (cat (l) (ek (l)) (eoo-star (hash k) (enc m k) (ek (l)))
              (eor-star (hash k) (enc m k) (ek (l)))))
   (send (cat (l) (ek (l)) (eoo-star (hash k) (enc m k) (ek (l)))
              (eor-star (hash k) (enc m k) (ek (l)))
              (enc "rcrq" (l) (ek (l)) (privk "sign" b))))
   (recv (cat k r)))
  (non-orig (privk hash_id)))

;; Responder run with recovery request and forced abort

(defrole resp3
  (vars (a b t hash_id name) (e1 e2 x mesg))
  (trace
   (recv (cat (l-star e1 x) e1 e2 (eoo-star x e1 e2)))
   (send (cat (l-star e1 x) e2 (eoo-star x e1 e2) (eor-star x e1 e2)))
   (send (cat (l-star e1 x) e2 (eoo-star x e1 e2) (eor-star x e1 e2)
              (enc "rcrq" (l-star e1 x) e2 (privk "sign" b))))
   (recv (enc "abcf"
              (enc "abrq" (l-star e1 x) (privk "sign" a))
              (privk "sign" t))))
  (non-orig (privk hash_id)))

;; TTP handles an abort

(defrole ttp-ab1
  (vars (a b t hash_id name) (y x mesg))
  (trace
   (recv (enc "abrq" (l-prime y x) (privk "sign" a)))
   (send (cat "sync-abrq"
              (enc "abrq" (l-prime y x) (privk "sign" a))))
   (send (enc "abcf"
              (enc "abrq" (l-prime y x) (privk "sign" a))
              (privk "sign" t))))
  (non-orig (privk hash_id)))

;; TTP forces recovery in response to an abort request

(defrole ttp-ab2
  (vars (a b t hash_id name) (y x e mesg))
  (trace
   (recv (enc "abrq" (l-prime y x) (privk "sign" a)))
   (recv (cat "sync-abrq"
             (enc "eortag" (hash (l-prime y x)) e
                  (privk "sign" b))))
     (send (enc "eortag" (hash (l-prime y x)) e
               (privk "sign" b))))
    (non-orig (privk hash_id)))

  ;; TTP handles a recovery

  (defrole ttp-rc1
    (vars (a b t hash_id name) (r text) (k skey) (y mesg))
    (trace
     (recv (rr-prime-prime y))
     (send (cat "sync-rc-req"
               (rr-prime-prime y)))
     (send (cat k r)))
    (non-orig (privk hash_id)))

  ;; TTP forces abort in response to a recovery request

  (defrole ttp-rc2
    (vars (a b t hash_id name) (r text) (k skey) (y mesg))
    (trace
     (recv (rr-prime-prime y))
     (recv (cat "sync-rc-req"
(enc "abrq" (l-prime-prime y) (privk "sign" a))))
     (send (enc "abcf"
               (enc "abrq" (l-prime-prime y) (privk "sign" a))
               (privk "sign" t))))
    (non-orig (privk hash_id)))

  ;; TTP handles a confirm request

  (defrole ttp-cf1
    (vars (a b t hash_id name) (r text) (k skey) (y mesg))
    (trace
     (recv (cf-prime-prime y))
     (send (cat "sync-cf-req"
               (cf-prime-prime y)))
     (send (enc (cf-prime-prime y) (privk "sign" t))))
    (non-orig (privk hash_id)))

  ;; TTP replies with abort token given confirm request

  (defrole ttp-cf2
    (vars (a b t hash_id name) (r text) (k skey) (y mesg))
    (trace
     (recv (cf-prime-prime y))
     (recv (cat "sync-cf-req"
               (enc "abrq"(l-prime-prime y) (privk "sign" a))))
     (send (enc "abcf"
                (enc "abrq"(l-prime-prime y) (privk "sign" a))
                (privk "sign" t))))
    (non-orig (privk hash_id))))

;; End of Wang's protocol defn.

;; Two experiments to prove Lemma 4.1

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand init1 1 (b b) (t t) (m m) (r r) (k k))
  (deflistener m)
  (non-orig
   (privk "encr" t))
  (uniq-orig m k)
  (comment "Experiment 1 to prove Lemma 4.1."))

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand init1 1 (b b) (t t) (m m) (r r) (k k))
  (deflistener k)
  (non-orig
   (privk "encr" t))
  (uniq-orig m k)
  (comment "Experiment 2 to prove Lemma 4.1."))

;; We now have three experiments
;; to prove Lemma 4.2, clause 1.

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand init1 3 (b b) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" b))
  (comment "First of three experiments to prove Lemma 4.2, clause 1."))

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand init3 3 (b b) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" b))
  (comment "Second of three experiments to prove Lemma 4.2, clause 1."))

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand init5 4 (b b) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" b))
  (comment "Third of three experiments to prove Lemma 4.2, clause 1."))
```

```
;; Two experiments to prove 4.2, Clause 2.

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand resp1 3 (a a) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" a))
  (comment
   "First of two experiments to prove Lemma 4.2, clause 2."))

(defskeleton wang
  (vars (a b t hash_id name) (m data) (r text) (k skey))
  (defstrand resp2 4 (a a) (t t) (m m) (r r) (k k))
  (non-orig
   (privk "sign" a))
  (comment
   "Second of two experiments to prove Lemma 4.2, clause 2."))

;; Experiment to prove Lemma 4.2, clause 3.

(defskeleton wang
  (vars (a b t hash_id name) (e1 e2 x mesg))
  (defstrand resp3 4 (a a) (t t))
  (non-orig
   (privk "sign" a))
  (comment
   "Experiments to prove Lemma 4.2, clause 3."))

;; One experiment to prove Lemma 4.3, Clause 1.

(defskeleton wang
  (vars (a b t hash_id name) (y x mesg))
  (deflistener
    (enc "abcf" (enc "abrq" (l-prime y x) (privk "sign" a))
 (privk "sign" t)))
  (non-orig
   (privk "sign" t))
  (comment
   "Experiments to prove Lemma 4.3, clause 1."))

;; Three experiments to prove Lemma 4.3, Clause 2.

(defskeleton wang
  (vars (y x mesg) (a b t name))
  (defstrand ttp-ab1 3 (y y) (x x) (a a) (b b) (t t))
  (non-orig (privk "sign" a)  (privk "encr" t))
  (comment
   "Experiment 1 to prove Lemma 4.3, clause 2."))

(defskeleton wang
  (vars (a b t hash_id name) (r text) (k skey) (y mesg))
  (defstrand ttp-rc2 3 (a a) (b b) (t t))
  (non-orig (privk "sign" a)  (privk "encr" t))
  (comment
   "Experiment 2 to prove Lemma 4.3, clause 2."))

(defskeleton wang
  (vars (a b t hash_id name) (r text) (k skey) (y mesg))
  (defstrand ttp-cf2 3 (a a) (b b) (t t))
  (non-orig (privk "sign" a)  (privk "encr" t))
  (comment
   "Experiment 3 to prove Lemma 4.3, clause 2."))
```

Contents

List of Figures