Stepwise Development of Security Protocols: A Speech Act-Oriented Approach

Phan Minh Dung
Asian Institute of Technologies
Computer Science and Information Management Program
PO Box 4, Klong Luang, 12120 Pathumthani, Thailand
dung@cs.ait.ac.th

Phan Minh Thang
Asian Institute of Technologies
Acrors Program
PO Box 4, Klong Luang, 12120 Pathumthani, Thailand
thangphm@cs.ait.ac.th

ABSTRACT

We propose a novel multi-layers paradigm for the design of key exchange protocols. In the top layer, protocols are specified in a high-level, declarative, formal language using speech acts as the basic building blocks. The declarative semantics of speech acts are specified by their preconditions and effects like in Hoare logics. A protocol logic, called ProtoLog, is developed for reasoning about speech act oriented protocols. Using the language of speech acts, protocol designers could develop their protocols in an modular and compositional way that are correct from the outset.

High-level speech act-oriented protocols are automatically translated into lower-level message exchanging protocols by a "protocol compiler" that implements speech acts by sending and receiving appropriate encrypted messages.

To demonstrate the applicability of our idea, we apply it on the class of well-designed key exchange protocols where a protocol is well-designed if a speech act is executed only if its preconditions are satisfied. We develop a "protocol compiler" for the class of well-designed protocols and prove the soundness and a limited form of completeness of the protocol logic ProtoLog wrt the translation, implemented by the compiler, under the Dolev-Yao assumption of perfect cryptography. An immediate corollary from the soundness result is the guarantee of the secrecy of exchanged keys (an essential security requirement of key exchange protocols) in well-designed protocols.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Logics of programs, Pre- and post-conditions*

General Terms

Security, Design, Languages, Verification.

Keywords

Security protocols, cryptographic protocols

1. INTRODUCTION

Security protocols play a pivotal role in information security. But despite the commonly recognized fact that information security has become a serious concern in commercial deployment of application and middleware, the development and deployment of security protocols is still an error-prone and difficult process [8, 19, 29, 23, 22, 3], done in an ad-hoc manner, without a formal specification of their security requirements. We hence need a process for making a stepwise transition from high-level security protocol design through development to code.

In general, security protocols often have structures that determines the composition of the intention and semantics of the individual messages. Recognizing these structure could allow us to create a radically new modular approach to security protocol development in which high-level security designs are translated stepwise to code. The obtained protocols at all levels are guaranteed to be correct from the outset. To illustrate this idea, let us look at an example

Example 1: Consider a protocol for distributing a fresh session key K generated by a server S, to two principals A and B. At the end of the protocol, both A and B are expected to get K and also to know that the other has got K as well.

$(1) A \rightarrow S : req, newkey, A, Init, S, Server, N_a, B$
$(2) S \rightarrow A : \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}}$
$(3) S \rightarrow B : \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
$(4) B \rightarrow A : \{req, keyconfirm, B, Resp, A, Init, N_b, Hash(K), S\}_{K_A}$
$(5) A \rightarrow B : \{rep, keyconfirm, A, Init, B, Resp, N_b\}_{K_B}$
$(6) B \rightarrow A : \{inf, keyconfirm, B, Resp, A, Init, S\}_K$

Note that K_X (resp. K_{XY}) represents a public (resp. secret common) key of X (resp. between X and Y). $\{g\}_k$ denotes the encryption of g using key k. Following the prudent engineering practice of Abadi and Needham [2] each message is designed to contain explicitly the identity of the sender, receiver together with their roles in the protocol as well as the purpose of the message.

The first message is a request from A, as initiator, to S, as server, for a fresh session key with B. The keywords *req, newkey* indicate that the message represents a request for a new key. N_a is a fresh nonce.

S replies by sending a newly generated session key K to A in the second message. The keywords *rep, newkey* indicate that the message is a reply to an earlier request for a new key. After getting the reply message, A knows that K is a fresh session key between A and B generated by S.

S informs B about key K in the third message (B plays the role of a responder). The keywords *inform, newkey* indicate that the message is intended to inform B about a new key. After receiving the third message, B could only say that it has been informed about K without being sure whether K is fresh or not. This is because B has no knowledge about K before receiving the third message and hence, B could not verify whether this message is just sent recently or it has been replayed by a penetrator.

To verify the information it has just obtained, B requests A in the fourth message to confirm that A knows that K is indeed a fresh session key between A and B generated by S. The keywords *req, keyconfirm* indicate that the message represents a request for confirming the status of the included key. N_b is a fresh nonce.

A replies to B by sending the fifth message to confirm that K is indeed a fresh session key between A and B. The keywords *rep, keyconfirm* indicate that the message represents a reply to an earlier request to confirm the status of K. After getting this message, B knows that what it has been informed before is correct.

B sends A the sixth message to confirm that B now knows that K is a secret session key between A and B. The keywords *inform, keyconfirm* indicate that the intention of the message is for B, to confirm to A that it knows about K.

The above protocol could be viewed as an implementation of the following more abstract one using speech acts as the basic building blocks [1]:

S_1 : *A requests S to generate a new session key for communication with B*
S_2 : *S replies to A by sending A a newly generated key K*
S_3 : *S informs B that K is a new session key between A and B generated by S*
S_4 : *B requests A to confirm that K is a fresh session key between them generated by S*
S_5 : *A replies to B to confirm that K is indeed a fresh session key between them generated by S*
S_6 : *B informs A to confirm that B knows that K is a fresh session key between them generated by S*

where A,B,S play the roles of initiator, responder and key server respectively.

The new approach proposed in this paper allows protocol designers to develop their protocols in an abstract speech act-oriented language like. A "protocol compiler" would translate automatically the abstract protocols into lower-level ones. □

In this paper, we propose a multi-layered approach in the design of security protocols. In the top layer, protocols are specified using speech acts as the basic building blocks. Like in Hoare logics, the declarative semantics of speech acts are specified by their preconditions and effects. High-level protocols are translated automatically into the lower-level message-exchanging protocols by "protocol compilers" where speech acts are implemented by sending and receiving appropriate encrypted messages. Like in conventional programming, "protocol compilers" are developed by the designers of speech act languages, not by their users. Ensuring the correctness of "protocol compilers" is hence a responsibility of the speech act language designers, not of the protocol programmers. In the proposed new paradigm, designers could develop security protocols that are correct from the outset, in an intuitive, modular and compositional language of speech acts.

Key exchange protocols form an important and well-studied class of security protocols. Due to the limited space and to get the idea of the new approach easier across to the readers, we focus technically in this paper on key exchange protocols. We introduce a speech act language for the class of well-designed key exchange protocols together with a protocol logic called ProtoLog for reasoning about the mental states of principals participating in protocol runs where a protocol is well-designed if a speech act is executed only if its preconditions are satisfied. We develop a "protocol compiler" for the introduced language and prove the soundness and a limited form of completeness of the protocol logic ProtoLog wrt the translation, implemented by the compiler, under the Dolev-Yao assumption of perfect cryptography. An immediate corollary from the soundness result is the guarantee of the secrecy of exchanged keys (an essential security requirement of key exchange protocols) in well-designed protocols.

The paper is organized as follows. In chapter 2, we present a language of speech acts, and introduce well-designed protocols. In chapter 3, a protocol logic called ProtoLog for reasoning about the mental states of principals participating in protocol runs. A "protocol compiler" is introduced in chapter 4 and the soundness and completeness of the protocol logic ProtoLog wrt the translation implemented by the compiler are proved in chapter 5. In chapter 6, we conclude and discuss related works.

2. SPEECH ACTS AND WELL-DESIGNED PROTOCOLS

We first propose a simple logical language for specifying the mental states of protocol principals. In contrast to BAN-style logics [6, 27], we do not employ a belief operator. Instead we have two operators: "being informed" and "knowing". Intuitively, to say that an agent knows something is to say that the agent's information about this something is correct and the agent is also aware about the correctness of the information while to say that an agent is informed of something simply indicates that the principal has obtained a piece of information without giving any hint about the correctness of the information as well as the awareness of the agent about it. Jumping ahead, our actual semantic definition follows Fagin et all [12] in defining that an agent knows something if it is true in the current state and true in every other state in which the agent makes precisely the same observations.

We assume the existence of pairwise disjoint sets **NONCE, NVAR, KEY, KVAR** and **PI, PVAR** of nonces, nonce variables, keys, key variables and principals and principal variables respectively. There is a distinguished identifier $PE \in PI$ denoting the *penetrator*. A principal that is not the penetrator is called *regular*. In security protocols, prin-

[1]Speech acts have been proposed as the basic primitives for declarative agent communication language in the AI community [18]

cipals often play different roles like the roles of initiators, responders or key servers. The roles of the principals involved in a protocol form an important component in its semantics. We assume the existence of a finite set **RO** of predefined role identifiers. A *principal term* is either a principal from PI, or a principal variable from PVar. *Nonce term* and *key terms* are defined similarly.

Let A,B,C,X,Y,Z be principal terms, n be a nonce term and K be a key term and ρ be a role. A *basic formula* has one of the following forms:

1. $GK_{A,\rho}(K, B, C)$ stating that principal A acting in role ρ has generated a fresh key K as a session key between B and C
2. $GN_{A,\rho}(n)$ stating that principal A acting in role ρ has freshly generated nonce n
3. $HN_{A,\rho}(n)$ stating that principal A acting in role ρ has nonce n
4. $Informed_{A,\rho}(K, X, Y, Z)$ stating that principal A acting in role ρ has been informed that K is generated recently by Z as a session key between X,Y.
5. $Key(K, A, B, C)$ stating that K is a session key between principals A and B generated recently by C.
6. $Access(A, K)$ stating that A has access to key K

A *formula* is composed from basic formulas using the logical operators $\wedge, \vee, \neg, \rightarrow$ together with the knowledge operator $Know_{A,\rho}F$ stating that A acting in the role ρ, knows that F holds.

One may wonder of whether it is possible to replace the notion Key(K,A,B,C) by just Key(K,A,B) without mentioning explicitly the generator C of K. Example 1 is a case where an employment of Key(K,A,B) would lead the responder B to believe in the trustworthiness of K even if S is the penetrator and A mistakenly believes S is honest while B is aware about the true identity of S.

2.1 KPL: A Keys-Principals Association Logic

We introduce now KPL (Keys Principals Association Logic), a specialized version of the modal logic S5, for reasoning about keys and its association to principals. The logic is needed to define well-designed protocols. Let A,B,C,D be regular principal identifiers and X be a principal identifier. KPL extends the system S5 in modal logic [17, 12] with the following axioms:

A1 $Key(K, A, B, C) \rightarrow Key(K, B, A, C)$
A2 $Informed_{A,\rho}(K, B, C, D) \rightarrow Informed_{A,\rho}(K, C, B, D)$
A3 $GK_{A,\rho}(K, B, C) \rightarrow GK_{A,\rho}(K, C, B)$
A4 $GN_{A,\rho}(n) \rightarrow HN_{A,\rho}(n)$
A5 $GK_{A,\rho}(K, B, C) \rightarrow Key(K, B, C, A)$
A6 $F \rightarrow Know_{A,\rho}F$ *for all basic formula F of the form* $GK_{A,\rho}(K, B, C)$, $GN_{A,\rho}(n)$, $HN_{A,\rho}(n)$ *and* $Informed_{A,\rho}(K, B, C, D)$
A7 $Know_{A,\rho}Key(K, B, C, D) \rightarrow Informed_{A,\rho}(K, B, C, D)$
A8 $Informed_{A,\rho}(K, B, C, D) \rightarrow Access(A, K)$

Axioms A1-A5 and A8 follow directly from the intuitions of the involved basic formulas. The intuition of axiom A6 is that if a principal is doing something then it is also aware about it. Axiom A7 relates the knowing and being informed operators, stating the obvious that if A knows that K is a fresh session key between B,C generated by D then A is also informed about it. The axioms and proof rule in modal system S5 [17, 12] are adapted to KL as follows:

$$Know_{A,\rho}F \rightarrow F$$
$$Know_{A,\rho}F \wedge Know_{A,\rho}(F \rightarrow G) \rightarrow Know_{A,\rho}G$$
$$Know_{A,\rho}F \rightarrow Know_{A,\rho}Know_{A,\rho}F$$
$$\neg Know_{A,\rho}F \rightarrow Know_{A,\rho}\neg Know_{A,\rho}F$$
From $\vdash F$ infer $\vdash Know_{A,\rho}F$

Table 1: Speech-Acts Forms

Request	Reply	Inform
Sender: A	Sender: A	Sender: A
Role of Sender: ρ	Role of Sender: ρ	Role of Sender: ρ
Receiver: B	Receiver: B	Receiver: B
RoleOfReceiver: τ	RoleOfReceiver: τ	RoleOfReceiver: τ
Type: t	Type: t	Type: t
Content: Con	Content: Con	Content: Con
Reply-With: n	Reply-To: n	

2.2 Speech Acts and Well Designed Protocols

We introduce a set of speech acts consisting of Request, Reply and Inform acts that we believe form a core of primitives that capture the most essential structures in security protocols. This set is extensible to capture further structures with new acts.

There are two types of speech acts: *newkey, keyconfirm.* Intuitively, an act of type *newkey* is used for distributing a new key while an act of type *keyconfirm* is used for confirming the knowledge of a new key. For example, the first (resp. last) three acts in the abstract protocol in example 1 are of type *newkey* (resp. *keyconfirm*).

Speech acts are defined as speech act forms that contain no variables. The structure of speech act forms are given in the table 1 where A,B are principal terms, ρ, τ are roles , n is a nonce term, $t \in \{newkey, keyconfirm\}$, and Con represents the content of the acts. It is required that the roles of sender and receiver in an act must be different, i.e. $\rho \neq \tau$. The nonce n in a request act is generated randomly by A when A performs the act.

The content of a request of type *newkey* has the form *Key(?,A,C,B)* stating that A requests B to generate a new session key for A to communicate with C.

The content of a request of type *keyconfirm* has the form *Key(K,A,B,C)* stating that A asks B to confirm that B knows that K is a fresh session key between A and B generated by C.

In a reply or inform act of type *newkey*, the sender A sends the receiver B a freshly generated session key for communication with C. If the key has not been generated, then the sender has to generate it first and then sends it. In this case the content of the act has the form $\nu K.Key(K, B, C, A)$. Otherwise it has the form $Key(K, B, C, A)$ in a reply act or the form $Key(K, B, C, A)$ or $Key(K, B, A, C)$ (depending on whether K is generated by A or by C) in an inform act.

The content of a reply or inform act of type *keyconfirm* is simply of the form $Key(K, A, B, C)$ affirming that K is indeed a fresh session key between A and B generated by C.

The declarative semantics of speech acts is specified by their preconditions and effects. The preconditions describe the necessary information and knowledge a honest principal should have to be able to send the acts and for the receiver to accept and process it *without leaking any secret information or being fooled by the penetrator.* The effects of a speech act describe the information and knowledge a principal gains after sending it (for the sender) or receiving it (for the receiver). Regular principals are assumed to be honest.

Let S be a speech act form as defined above. The preconditions and effects of the event of sending (resp. receiving) S are denoted by Pre(+S) and Effect(+S) (resp. Pre(-S) and Effect(-S)) and formalized in the tables 2-5 below.

The definition of key exchange protocols is based on a notion of conversation form where a **conversation form** is a pair (Req, Rep) of request and reply act forms of the same type and with the same nonce such that 1) the sender (resp. receiver) and its role in Req coincide with the receiver (resp. sender) and its role in Rep respectively, and 2) if the acts in the conversation are of type *keyconfirm* then their contents coincide, and 3) if the acts in the conversation are of type *newkey* and the content of the request has the form Key(?,A,C,B) then the content of the reply has the form Key(K,A,C,B) or $\nu K.Key(K, A, C, B)$

Definition *A* **speech act oriented key exchange protocol** *(or simply key exchange protocol) is a sequence of speech act forms $S_1, \ldots, S_k$ such that*

1) For each request (resp. reply) S_i, there is exactly one reply (resp. request) S_j, $j > i$ (resp. $j < i$) such that S_i, S_j (resp. S_j, S_i) form a conversation form.

2) Each principal term has at most one role, i.e. for each principal term P in AP, if there exist S_i, S_j such that P appears as a sender or receiver in both S_i and S_j then P has the same role in both acts. [2]

3) Every role in AP is occupied by exactly one principal term, i.e. for all principal terms P, Q in AP, if there exist S_i, S_j and $\rho \in RO$ such that P has role ρ in S_i and Q has role ρ in S_j then $P = Q$.

4) Variables representing nonces of different conversation forms are different.

5) The penetrator identifier does not appear in any S_i.

6) There is exactly one key term appearing in the protocol and the first act in which it appears is a reply or inform act of type newkey that is also the only act in the protocol with a content of the form $\nu K.Key(K, A, B, C)$

Now we can introduce well-designed protocols. For each speech act form S, define $Pre(S) = Pre(+S) \wedge Pre(-S)$ and $Effect(S) = Effect(+S) \wedge Effect(-S)$.

Definition *A key exchange protocol $AP = S_1, \ldots, S_k$ is said to be* **well-designed** *if speech acts are scheduled for execution only when their preconditions are satisfied, that means following conditions are satisfied:*

1) $Pre(S_1) = True$

2) For each $2 \leq i \leq k$: $Effect(S_1) \wedge \ldots \wedge Effect(S_{i-1}) \vdash Pre(S_i)$

It is not difficult to see that the abstract protocol in example 1 is well-designed. The following example illustrates how a violation of the well-designed condition could lead to serious flaws.

Let $S_1, \ldots, S_6$ be the acts of the abstract protocol in example 1. Consider a new protocol $P' = S_1', S_2', S_3', S_4', S_5'$ defined by $S_i' = S_i$ for $i \in \{1, 2, 3\}$ and $S_4' = S_6$ and S_5' is an inform act of type *keyconfirm* from A to B to confirm the receipt of K. P' is in fact an abstraction of the Needham-Schroeder protocol with symmetric key (NSPS) [8]. P' is not well-designed as the preconditions of S_4' do not follow from the effects of the previous acts in it. Similar to NSPS, P' is subjected to a replay attack in which the penetrator replays an old message S_3' to B whose key K has become stale. The penetrator then intercepts the message S_4' that B sends to A and then sends S_5' to B as specified by the protocol. B is then tricked into falsely believing that it has a secret common key for communicating with A generated by S.

3. PROTOLOG: A LOGIC FOR WELL-DESIGNED PROTOCOLS

We introduce now ProtoLog, a protocol logic, for reasoning about the mental states of principals participating in well-designed protocols. Let $\rho \in RO$ be a role in a protocol $AP = S_1, \ldots, S_n$. The **ρ-track of** AP is the sequence $AP_\rho = \sigma_{i_1} S_{i_1}, \ldots, \sigma_{i_k} S_{i_k}$ where $\sigma_{i_j} \in \{+, -\}$, $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, such that for each $1 \leq j \leq n$, $+S_j \in AP_\rho$ (resp.$-S_j \in AP_\rho$) iff ρ is the role of the sender (resp. receiver) in S_j. For illustration, the initiator-track of the abstract protocol in example 1 is $+S_1, -S_2, -S_4, +S_5, -S_6$ while its responder-track is $-S_3, +S_4, -S_5, +S_6$. A **run** of principal A in the role ρ according to a protocol AP is a ground instance of a prefix of the ρ-track AP_ρ of AP in which A is the principal having the role ρ. A is also called the principal of such run. A **proper-regular** run is a run where the penetrator PE does not appear in any of its acts. Let $R = E_1 \ldots E_k$ be a run. Define $Effect(R) = Effect(E_1) \wedge \ldots \wedge Effect(E_k)$. Note that $Effect(nil) = True$.

To reason about the effects of runs of protocols , we introduce new formulas called **protocol formulas** of the form **AP**.**R** [**F**] stating that formula F holds after run R according to AP has been executed (by the principal of R).

Let AP be a protocol, E be an event of sending or receiving a speech act, R be a proper-regular run of A in a role ρ according to AP, and F,F' be formulas. Further let A,B,C,D be regular principal identifiers and X be a principal identifier. Protolog extends KPL with following axioms and proof rules:

A9 $Key(K, B, C, A) \rightarrow GK_{A,\rho_1}(K, B, C) \vee \ldots \vee GK_{A,\rho_k}(K, B, C)$
where $RO = \{\rho_1, \ldots, \rho_k\}$

A10 $GK_{A,\rho}(K, B, C) \wedge GK_{A',\rho'}(K, B', C') \rightarrow (A, \rho) = (A', \rho') \wedge (B, C) = (B', C')$

A11 $A \neq B \wedge A \neq C \wedge A \neq D \rightarrow \neg Informed_{A,\rho}(K, B, C, D)$

A12 $Key(K, B, C, A) \wedge X \neq A \wedge X \neq B \wedge X \neq C \rightarrow \neg Access(X, K)$

Effect Rule

$$\frac{}{\vdash AP.R\,[Effect(R)]}$$

Consequence Rule

$$\frac{\vdash AP.R\,[F], \;\vdash F \rightarrow F'}{\vdash AP.R\,[F']}$$

[2]This condition does not forbid a principal to play more than one role in an actual run of a protocol (when two different principal variables are instantiated by the same principal identifier), but it ensures that normally different principals have different roles in a run of the protocol.

Table 2: Request Acts

Act S	Request	
	newkey	keyconfirm
Pre(+S)	True	$Informed_{A,\rho}(K,A,B,C)$
Effect(+S)	$GN_{A,\rho}(n)$	$GN_{A,\rho}(n)$
Pre(-S)	True	$Know_{B,\tau}Key(K,A,B,C)$
Effect(-S)	$HN_{B,\tau}(n)$	$Know_{B,\tau}Informed_{A,\rho}(K,A,B,C) \wedge HN_{B,\tau}(n)$

Table 3: Reply Acts

Act S	Reply		
	newkey		keyconfirm
	$Con \equiv \nu K.Key(K,B,C,A)$	$Con \equiv Key(K,B,C,A)$	
Pre(+S)	$HN_{A,\rho}(n)$	$HN_{A,\rho}(n) \wedge$ $GK_{A,\rho}(K,B,C)$	$Know_{A,\rho}Key(K,A,B,C) \wedge$ $HN_{A,\rho}(n)$
Effect(+S)	$GK_{A,\rho}(K,B,C)$	True	True
Pre(-S)	$GN_{B,\tau}(n)$		$GN_{B,\tau}(n) \wedge$ $Informed_{B,\tau}(K,A,B,C)$
Effect(-S)	$Know_{B,\tau}GK_{A,\rho}(K,B,C) \wedge$ $Know_{B,\tau}HN_{A,\rho}(n)$		$Know_{B,\tau}HN_{A,\rho}(n) \wedge$ $Know_{B\tau}Know_{A\rho}Key(K,A,B,C)$

Table 4: Inform Acts

Act S	Inform		
	newkey		keyconfirm
	Con $\equiv \nu K.Key(K,B,C,A)$	$Con \equiv Key(K,B,X,Y)$ where (X,Y) = (A,C) or (C,A)	
Pre(+S)	True	$Know_{A,\rho}Key(K,B,X,Y)$	$Know_{A,\rho}Key(K,A,B,C)$
Effect(+S)	$GK_{A,\rho}(K,B,C)$	True	True
Pre(-S)	True	True	$Know_{B,\tau}Key(K,A,B,C)$
Effect(-S)	$Informed_{B,\tau}(K,B,C,A)$	$Informed_{B,\tau}(K,B,X,Y)$	$Know_{B,\tau}Know_{A,\rho}$ $Key(K,A,B,C)$

Axiom A9 states that if a key K is generated by A then K is generated by A acting in some role. Axioms A10,A11,A12 are based on the honesty assumption of regular principals. The intuition of axiom A10 is that generated keys are random and hence it is impossible for an agent to generate the same key twice. Axiom A11 follows from the structure of the speech acts that requires that when a key is sent in a speech act, information about the association of the key to the receiver of the act must be included. Hence a honest principal would never receive a key that is not associated with it. Axiom A12 states that freshly generated session keys are accessible only by their generator and the principals for whom they are generated. This axiom implies that well-designed protocols protect the secrecy of the exchanged keys.

Let AP be the abstract protocol in example 1. Let R_0, R_1 be complete runs of regular principals A and B in the initiator (denoted by I) and responder (denoted by R) roles respectively. S is a regular principal in the role of a server. Applying the effect rule and then the consequence rule, we obtain:
$\vdash AP.R_0[Know_{A,I}Know_{B,R}Key(K,A,B,S)]$ and
$\vdash AP.R_1[Know_{B,R}Know_{A,I}Key(K,A,B,S)]$
Hence
$\vdash AP.R_0[Know_{A,I}Key(K,A,B,S)]$ and
$\vdash AP.R_1[Know_{B,R}Key(K,A,B,S)]$
Applying the inference rule and axioms of S5 and the axiom A12, we obtain:
$\vdash AP.R_0[Know_{A,I}\neg Access(PE,K)]$ and
$\vdash AP.R_1[Know_{B,R}\neg Access(PE,K)]$

4. IMPLEMENTING SPEECH ACTS

The events of sending or receiving speech acts are implemented by the actions of sending or receiving some message. A *sending action* has the form $+\mathbf{m}$ or $+\nu\mathbf{b}.\mathbf{m}$ while a *receiving action* has the form $-\mathbf{m}$ where m is a message and b is a key or nonce. The notation νb indicates that b is freshly (or randomly) generated at this action (i.e. the action actually consists of two tasks: generating b and sending out the message). m is also called the *message of the respective action.*

Note that for ease of reference, we have made a distinction between the higher-level notion of events of sending or receiving speech acts denoted by +S or -S where S is a speech act, and the lower-level notion of actions of sending or receiving messages. As such, events are implemented by actions.

Let S be a speech act. The messages appearing in the actions implementing the events of sending or receiving S are called the *representations of S.* A speech act could have more than one representation.

Taking a hint from prudent engineering [2], a representation of a speech act should contain vital information about its type, its content, the identity and role of its sender and receiver and other information like reply-to, and reply-with-nonces. We simply assume that the principals somehow recognize the basic message components like nonce, keys and principal identifiers when they see them. See [16] on how it could be done.

Let S be a speech act from a sender A in role ρ to a receiver B in role τ. The notation $\mathbf{m_S}$ refers to a representation of S. We also often say that m_S *represents* S.

If S is an inform act of type *newkey* whose content is of the form Key(K,B,X,Y) (where (X,Y) = (A,C) or (C,A)) then
$m_S = \{inf, newkey, A, \rho, B, \tau, K, B, X, Y\}_{K_{AB}}$, or
$m_S = \{inf, newkey, A, \rho, B, \tau, K, B, X, Y\}_{K_B}$,

If S is an inform act of type *keyconfirm* with a content of the form Key(K,A,B,C) then
$m_S = \{inf, keyconfirm, A, \rho, B, \tau, C\}_K$, or
$m_S = \{inf, keyconfirm, A, \rho, B, \tau, Hash(K), C\}_{K_B}$

The representation of a request (resp. reply) act depends on the representation of the reply (resp. request) act in the same conversation. Let (S,S') be a conversation form and n be the nonce in S,S'.

If (S,S') is a conversation of type *newkey* and the content of S' is $\nu K.Key(K,A,C,B)$, then there are at least two different ways to represent (S,S'):

- $m_S = \{req, newkey, A, \rho, B, \tau, n, C\}_{K_B}$
  $m_{S'} = \{rep, newkey, B, \tau, A, \rho, n, K, C\}_{K_A}$
- $m_S = req, newkey, A, \rho, B, \tau, n, C$
  $m_{S'} = \{rep, newkey, B, \tau, A, \rho, n, K, C\}_{K_{AB}}$,

If (S,S') is a conversation of type *keyconfirm* and the content of S is Key(K,A,B,C), then there are at least two different ways to represent (S,S'):

- $m_S = \{req, keyconfirm, A, \rho, B, \tau, n, Hash(K), C\}_{K_B}$
  $m_{S'} = \{rep, keyconfirm, B, \tau, A, \rho, n\}_{K_A}$
- $m_S = \{req, keyconfirm, A, \rho, B, \tau, n, Hash(K), C\}_{K_{AB}}$
  $m_{S'} = \{rep, keyconfirm, B, \tau, A, \rho, n, Hash(K), C\}_{K_{AB}}$,

The event +S of sending a speech act S is implemented by the action $+\mathbf{m_S}$ with two exceptions: **1**) if S is a request act with nonce n then the event of sending S is implemented by the action $+\nu\mathbf{n}.\mathbf{m_S}$, and **2**) if S is a reply or inform act of type *newkey* with a content of the form $\nu K.Key(K,X,Y,Z)$, then the event of sending S is implemented by the action $+\nu\mathbf{K}.\mathbf{m_S}$.

The event -S of receiving a speech act S is always implemented by the action $-\mathbf{m_S}$

As there are many different ways to implement speech acts, a speech act-oriented protocol could have many different implementations. For example, there are at least 16 different but correct ways to implement the abstract protocol in example 1.

Though a speech act could have many representations, it is easy to see that each representation represents exactly one speech act

Lemma 1: Let S and R be speech acts and m_S, m_R be representations of S,R respectively. If S and R are different acts then $m_S \neq m_R$

4.1 Role Topology and Message Forwarding

Many security protocols do not allow principals acting in certain roles to communicate directly. An example is the well-known Otway-Rees protocol [8] (see **appendix B**) that does not allow initiators to communicate directly with servers. All messages between principals in these roles are routed through the responders.

In general, each security protocol P assumes the existence of a directed graph $G = (RO, V)$, $V = RO \times RO$ describing the connection topology of the roles in P. A direct link from role ρ to role τ in G represents a direct communication channel from principals acting in role ρ to principals acting in role τ. A question that immediately arises is how to implement a speech act if the role topology forbids a direct communication between its sender and receiver.

Consider the abstract speech act oriented protocol in example 1. Suppose that the connection topology allows direct communication only between initiators and servers as well as between initiators and responders. In this case, the speech act (3) in which S informs B about the new key K is implemented by letting S (server) send message to B (responder) through A (initiator). See appendix C for more details.

We give now the general translation algorithm. Let P be a well-designed speech act oriented protocol and $G = (RO, V)$ be a given connection topology for the roles in P

Let S be a speech act in P and ρ, τ be the roles of the sender A and receiver B in S respectively. Further let $\rho, \rho_1, \ldots, \rho_m, \tau$ be a shortest path from ρ to τ with $A_1, \ldots, A_m$ being the principal terms occupying the roles $\rho_1, \ldots, \rho_m$ in P. S is implemented by forwarding the message m_S from A through $A_1, \ldots, A_m$ to B as:

$A \xrightarrow{m_S} A_1 \xrightarrow{m_S} \ldots \xrightarrow{m_S} A_m \xrightarrow{m_S} B$

P is translated into a message exchanging protocol by successively translating each of its acts as above.

5. SOUNDNESS AND COMPLETENESS OF PROTOLOG

Strand spaces have been introduced by Fabrega, Herzog and Guttman [11] to give an operational semantics for message-exchanging protocols under the Dolev-Yao assumption of perfect cryptography. We adapt the strand space model to our framework to give an operational semantics to the lower-level message-exchanging protocols obtained by implementing the speech acts as described in the previous chapter. The protocol logic Protolog is then interpreted over the adapted strand models.

Let AP be an arbitrary but fixed speech act oriented key exchange protocol. For simplicity and due to the limited space, we assume that AP has a role topology in which there is a direct link between every pair of different roles (all the results in this chapter are proved for the general case in the full paper). A strand is a sequence of nodes labelled by actions of sending or receiving messages. The ith node of a strand s is denoted by (s,i). Act(s,i) denotes the ith action in s. The message of the action labelling (s,i) is denoted by $term(s, i)$. Often, if there is no danger of confusion, a strand is identified with the sequence of actions labelling its nodes. A binary relation $\Rightarrow$ over the set of strand nodes is defined by $\Rightarrow = \{((s, i), (s, i + 1)) \mid s \text{ is a strand }\}$

Let S be a set of strands and (s,i) be a node in S. A key or nonce b is said to *originate at (s,i)* if (s,i) is a sending action and b occurs in term(s,i) and for all $j < i$, b does not occur in term(s,j). b is said to *uniquely originate at (s,i) in S* if (s,i) is the only node in S at which b originates.

There are two kinds of strands, *regular strand* and *penetrator strands.* A node lying on a regular (resp. penetrator) strand is called a *regular (resp. penetrator) node.* We often say that *a node N implements a speech act event E* if the action labelling N implements E. For a regular principal A and a role $\rho \in RO$, an *AP-regular strand* of A in role ρ is a regular strand $s = N_1, \ldots, N_m$ such that each N_i implements the event E_i in a run $R = E_1, \ldots, E_m$ of A in role ρ according to AP. We also say that *strand s implements run R.* There are eight kinds of penetrator strands:

Key-strand: $\langle +K \rangle$ where $K \in KEY \setminus \mathcal{K}$
RO-strand: $\langle \rho \rangle$ where $\rho \in RO$
PI-strand: $\langle +A \rangle$ where $A \in PI$
Nonce-strand: $\langle +n \rangle$ where $n \in NONCE$
S-strand: $\langle -gh \quad +g \quad +h \rangle$
E-strand: $\langle -K \quad -h \quad +\{h\}_K \rangle$
C-strand: $\langle -g \quad -h \quad +gh \rangle$
D-strand: $\langle -K^{-1} \quad -\{h\}_K \quad +h \rangle$

where $\mathcal{K}$ consists of all secret shared keys K_{AB}, and private keys K_A^{-1} of regular principals.

With the exception of the Key, and RO-strands, the other definitions of penetrator strands coincide with the definitions given in [11, 14]. In [11, 14], the penetrator is assumed to possess initially a set of keys and the key the penetrator could pick up in a key strand should belong to this set. We do not impose this restriction on the penetrator as we assume that the penetrator has access to any algorithm that could be used for key or nonce generation.

The assumption that the keys or nonces generated by regular principals are random and hence could not be generated by the penetrator is captured by their unique origination requirement in the defintion of historical bundles (introduced shortly below)

A **bundle of AP** is is defined as a pair $(S, \rightarrow)$ whereas S is a finite set of strands, $\rightarrow$ is a binary relation over the set of nodes in S such that the following two conditions are satisfied:

1) For each node (s,i) in S, if Act(s,i) is a receiving action then there is exactly one node (s',i') such that $Act(s', i')$ is a sending action and $(s', i') \rightarrow (s, i)$ and $term(s, i) = term(s', i')$ hold

2) The transitive closure of the relation $\rightarrow \cup \Rightarrow$ is acyclic

3) Every regular strand in S is AP-regular.

The notation $n \mapsto n'$ means that either $n \Rightarrow^+ n'$ with n a receiving node and n' a sending node, or $n \rightarrow n'$. A *path* is a finite sequence of nodes and edges $n_1 \mapsto n_2 \mapsto \ldots \mapsto n_k$. A strand s is said to **precede** (resp. **follow**) a strand r in a bundle $(S, \rightarrow)$ if there is path starting in s (resp. r) and ends in r (resp. s).

To deal with replay attacks, we introduce the idea of **historical bundle** as a triple $(S_0, S_1, \rightarrow)$ whereas $(S_0 \cup S_1, \rightarrow)$ is a bundle and S_0 (resp. S_1) is the set of **past** (resp. **recent**) strands. Recent strands represent runs that have just happened recently while past strands represent strands that had happened in the past. Therefore it is possible that the penetrator has access to keys and nonces generated in past strands. It is required that strands that precede a past strand are also past strands while strands that follow a recent strand are recent strands. A node is said to be **recent** (resp. **past**) if it lies on a recent (resp. past) strand. Historical bundles are required to satisfy the following two conditions:

1) For each recent node (s,i) in S, if a nonce or a key b is generated at (s,i), i.e. $Act(s, i) = +\nu b.m$, then b uniquely originates at (s,i) in S.

2) If a key or nonce b originates at a regular node (s,i) then b is also generated at (s,i).

Note that the assumption that the keys or nonces generated by principals during a recent run of the protocol are random is captured by their unique origination requirement.

¿From lemma 1 it follows immediately that for each regular node N, term(N) represents uniquely a speech act. We say that a *key K is associated with principal identifiers A,B,C at a node N* if K is associated with principal identifiers A,B,C in the speech act represented by term(N). Note again that if K is associated with A,B,C in a speech act then the sender and receiver of this act are among A,B,C.

Definition(**Model Semantics**) Let $\mathcal{B}$ be a historical bundle of a well-designed protocol, A,B,C be regular principals, D be an arbitrary principal and ρ be a role. Further let n be a nonce and K be a key.

1. $\mathcal{B} \models HN_{A,\rho}(n)$ iff there exists a recent regular strand $s \in \mathcal{B}$ of A in role ρ such that n is contained in an action in s.
2. $\mathcal{B} \models GN_{A,\rho}(n)$ iff there exists a recent regular strand $s \in \mathcal{B}$ of A in role ρ such that nonce n is generated in some sending action in s.
3. $\mathcal{B} \models Informed_{A,\rho}(K, B, C, D)$ iff there is a recent regular strand $s \in \mathcal{B}$ of A in role ρ and a node N in s such that N implements the receiving event of an inform or reply act and K is associated with B,C,D at N.
4. $\mathcal{B} \models GK_{A,\rho}(K, B, C)$ iff there exists a recent regular strand $s \in \mathcal{B}$ of A in role ρ such that key K is generated at some sending node N in s and K is associated with B,C,A at N
5. $\mathcal{B} \models Access(D, K)$ iff

   (a) D is a regular principal and there exists a recent regular strand $s \in \mathcal{B}$ of D such that K is contained in an action in s, or

   (b) D is the penetrator and there exists a penetrator node N such that K = term(N)
6. $\mathcal{B} \models Key(K, A, B, C)$ iff there exists a recent regular strand $s \in \mathcal{B}$ of C such that key K is generated at some sending node N in s and K is associated with A,B,C at N

Let $\mathcal{B}$ and $\mathcal{B}'$ be historical bundles and A be a regular principal identifier and ρ be a role. Further let Ω (resp. Ω') be the set of recent nodes of A in role ρ in $\mathcal{B}$ (resp. $\mathcal{B}'$). We say that $\mathcal{B}, \mathcal{B}'$ are *recent* (A, ρ)*-indistinguishable*, denoted by $\mathcal{B} \equiv_{A,\rho} \mathcal{B}'$ if there is a bijection ϕ between Ω and Ω' such that for all nodes $N, N' \in \Omega$ following conditions hold:
1) The actions labelling N and $\phi(N)$ coincide.
2) $N \Rightarrow N'$ iff $\phi(N) \Rightarrow \phi(N')$

What a principal in a role ρ knows at a certain state of the world depends on what it considers to be a possible world at this state [12]. Syverson [25] defined a world state as a bundle. Similarly, we define a world state as a historical bundle where a possible world for a principal A acting in a role ρ in a state $\mathcal{B}$ is a historical bundle $\mathcal{B}'$ such that $\mathcal{B}, \mathcal{B}'$ are recent (A, ρ)-indistinguishable.

Definition(**Model Semantics**, continued)
$\mathcal{B} \models Know_{A,\rho}F$ iff for all $\mathcal{B}'$ s.t. $\mathcal{B}, \mathcal{B}'$ are recent (A, ρ)-indistinguishable: $\mathcal{B}' \models F$

Let R be a run of a principal A according to a well-designed AP. Define $\models$ **AP**.**R**[**F**] iff for each historical bundle $\mathcal{B}$ of AP, if $\mathcal{B}$ contains a recent strand implementing R then $\mathcal{B} \models F$.

Let AP be a well-designed protocol, $\mathcal{B}$ be a historical bundle of AP, F be a formula and R be a proper-regular run of A in role ρ according to AP. The soundness as well as a limited completeness of the protocol logic ProtoLog are established in the following theorems.

THEOREM 1. *(Soundness Theorem)*
1) If $\vdash F$ *then* $\models F$
2) If $\vdash AP.R[F]$ *then* $\models AP.R[F]$
Proof: *Appendix D*

THEOREM 2. *(Limited Completeness Theorem)*
If $\models AP.R[Know_{A,\rho}Key(K, B, C, D)]$
then $\vdash AP.R[Know_{A,\rho}Key(K, B, C, D)]$
Proof: *Appendix E*

6. CONCLUSIONS AND RELATED WORKS

We have proposed a new approach to development of security protocols that are correct from the outset, by stepwise translation from high-level speech act oriented abstraction to lower-level message-exchanging protocols. We have demonstrated the power of this idea by technically applying it on the development of key exchange protocols. Our approach allows a protocol designer to work exclusively in the abstract high level language of speech acts. Protocol designers only need to ensure that their protocols are correct with respect to the high-level protocol logic ProtoLog. A correct "protocol compiler" that translates speech act events into message sending and receiving actions, ensures the correctness of the obtained lower-level message exchanging protocols. Ensuring the correctness of the "protocol compiler" is a responsibility of the speech act language designers, not of the protocol programmer. We have given a "protocol compiler" for the class of key exchange protocols and proved its soundness under the assumption of perfect cryptography.

As the logic ProtoLog is designed for reasoning about speech-act-oriented protocols, it could not be applied to reasoning about unstructured protocols. But this is not a weakness of our logic, very much like a logic designed for reasoning about structured Java programs is not expected to be used to reason about unstructured Fortran programs.

Recently much attention has turned to verification of protocols like SSL (Secure Socket Layer), TLS (Transport Layer Security) and SET (Secure Electronic Transactions) that are more complex than the class of key exchange protocols. The prevalent method for verification is based on human interaction with a powerful semi-automatic prover [29, 23, 22, 3]. But as pointed out in [4], the complexity of protocols like SET has probably set a limit on the applicability of this approach. To scale up, new advances are needed. We believe that the approach proposed in this paper could be extended naturally to deal with new classes of protocols including TLS or SET. We are working on it in an ongoing work.

There is a good body of work on designing security protocols. Guttman [13] has argued that the framework of authentication tests could be used in the design of security protocols. But it is not clear what kind of security goals are satisfied by authentication tests. The notion of conversation we introduced could be viewed as a high-level declarative embodiment of the idea of authentication tests. Gong and

Syverson [15] has developed an informal method for developing fail-stop security protocols. Meadows [21] has suggested that the design of cryptographic protocols should be layered in which a abstract model is used at the top layer and each successive layer is an implementation of the layer above it. A requirement language for security protocols has been given by Syverson and Meadows in [26]. Buttyan, Staamann and Wilhelm [7] has proposed an abstract BAN-like logic to be used in the protocol design. But it is not clear how to translate a protocol specified in the abstract logic into a lower level protocol and how to verify the correctness of such translation. Boyd and Mao [5] have discussed informally set-theoretic guidelines for the design of key exchange protocols. Abadi and Needham [2] have proposed a set of informal guidelines for authentication protocol design. Perrig and Song [28, 24] has developed tools based on the idea of strand space for analyzing protocols and later applied it in the design of protocols. A more recent work by Datta, Derek, Mitchell and Pavlovic [9] is especially relevant to our work. Though they do not propose a high-level language for protocol programming comparable to our language of speech acts, the techniques they study could turn out to be especially useful in the development and optimization of "protocol compiler". Abadi work on secrecy by typing [1] seems to be closely related to our result on the secrecy of exchanged keys of well-designed protocols.

7. REFERENCES

[1] M. Abadi Secrecy by typing in security protocols, JACM, 46, 5, 1999, 749-786

[2] M. Abadi, R. Needham. Prudent engineering practices for cryptographic protocols, IEEE Transactions on SE, 22(1): 6-15, 1996

[3] G. Bella, F. Massacci, L.C. Paulson Verifying the SET registration protocol, IEEE, Journal on selected areas in communication 21, 1, 2003

[4] G. Bella, F. Massacci, L.C. Paulson An overview of the verification of SET, International J. of Information security, in Press

[5] C. Boyd, W. Mao. Designing secure key exchange protocol, Esorics'94, pp 93-105

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

[7] L. Buttyan, S. Staamann, U. Wilhelm A simple logic for authentication protocol design, Proceedings of the 11th IEEE Computer Security Foundation Workshop, 153-162, 1998

[8] J. Clark, J. Jakob. A survey of authentication protocol literature, version 1, Department of Computer Science, University of York, Nov 1997

[9] A. Datta, A. Derek, J. Mitchell, D. Pavlovic A derivation system for security protocols and its logical formalization , IEEE Computer Security Foundation Workshop, 2003

[10] T. Dierks, C. Allen The TLS protocol version 1.0, RFC 2246, January 1999

[11] F. J.T. Fabrega, J.C. Herzog, J.D. Guttman. Strand spaces: Why is a security protocol correct ? Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp 160-171, 1998, IEEE Computer Scociety Press

[12] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi. Reasoning about knowledge. MIT Press, 1995

[13] J.D. Guttman. Security protocol design via authentication test, Proceedings of the 15th IEEE Computer Security Foundation Workshop, 2002

[14] J.D. Guttman, F. J.T. Fabrega Authentication tests and the structure of bundles, Theoretical computer science, 2001

[15] L. Gong, P. Syverson. Fail-stop protocols: An approavch to designing secure protocols, Proceedings of teh 5th International Conference on Dependable Computing for Critical Applications, 1995, pp 44-55

[16] J. Heather, G. Lowe, S. Schneider How to prevent type flaw attacks on security protocols, 13th CSFW, 2000

[17] G.E. Hughes, M.J. Cresswell. An introduction to modal logic, Methuen, London and NewYork, 1985

[18] Y. Labrou, T. Finin, Y. Peng. Agent communication languages: The current landscape, IEEE Intelligent Agents, March/April 1999, 45-52

[19] G. Lowe An attack on the Needham-Schroeder public key authentication protocol, Information Processing Letters 56, 1995

[20] L. Loeb Secure Electronic Transactions: Introduction and technical Reference, Artech House Pub., 1998

[21] C. Meadows. Formal verification of cryptographic protocols: A Survey, pp 135-150, Asiacrypt 1994,

[22] J. Mitchell, V. Schmatikov, U. Stern Finite State Analysis of SSL 3.0, 7th Usenix Security Symposium, 1998

[23] L.C. Paulson Inductive analysis of the internet protocol TLS, ACM Transaction on Computer and System Security, 1999

[24] A. Perrig, D. Song Looking for a diamond in the desert: Extending automatic protocol generation to three-party auhtentication and keyagreement protocols, Proceedings of the 13th IEEE Computer Security Foundation Workshop, 2000

[25] P. Syverson. Towards a strand semantics for authentication logic, Electronic Notes in Theoretical Computer Science, 20,2000

[26] P. Syverson, C. Meadows A formal language for cryptographic protocol requirements. Design, Codes and Cryptography, 7(1 and 2): 27-59, 1996

[27] P. Syverson, P.C. van Oorshot. On unifying some cryptographic protocols, Proceedings of the 1994 IEEE Symposium on Security and Privacy, 14-28

[28] D.X. Song. Athena: a new efficient automated checker for security protocol analysis, Proceedings of the 12th IEEE Computer Security Foundation Workshop, 1999

[29] D. Wagner, B. Schneier Analysisof the SSL 3.0 protocol, In 2nd Usenix workshop on electronic commerce, 1996

APPENDIX

A. APPENDIX A

A.1 Request Acts

Let S be a request act form as defined in the main text.

- Let the type of S be *newkey*. Let Key(?,A,C,B) be the content of S. As A does not need any knowledge to send out a request for a new key, the preconditions of sending such a request is True. That means

$$Pre(+S) \equiv True$$

When sending send out a request, A generates a new fresh nonce for the reply-with field. Hence the effects of sending out S is that A has generated a new nonce.

$$Effect(+S) \equiv GN_{A,\rho}(n)$$

As B does not need any knowledge to receive a request for a new key, the preconditions of receiving such a request is also True. That means

$$Pre(-S) \equiv True$$

For B, the effects of getting a request is that B now has the nonce n.

$$Effect(-S) \equiv HN_{B,\tau}(n)$$

- Let the type of S be *keyconfirm*. Let Key(K,A,B,C) be the content of S. For A to be able to send out such a request, A should be informed about K earlier. Hence

  $$Pre(+S) \equiv Informed_{A,\rho}(K, A, B, C)$$

  The effect of sending a request of type *keyconfirm* is the same as the effect of sending out a requests of type *newkey*:

  $$Effect(+S) \equiv GN_{A,\rho}(n)$$

  Before receiving a request act of type *keyconfirm*, receiver B should know that K is indeed a secret common key between A and B generated by C. Hence

  $$Pre(-S) \equiv Know_{B,\tau}Key(K, A, B, C)$$

  Because B knows that K is a secret common key between A and B, when B gets a request of type *keyconfirm*, B knows that A has been informed about K as a fresh session key between A and B. Hence

  $$Effect(-S) \equiv Know_{B,\tau}Informed_{A,\rho}(K, A, B, C) \wedge HN_{B,\tau}(n)$$

A.2 Reply Acts

Let S be a reply act form as defined in the main text.

- Let the type of S be *newkey* and the content of S be *Key(K,B,C,A)* or $\nu K.Key(K, B, C, A)$.

  For A to send B a key in a reply act, A must have been requested by B to do so before. If K has not been generated, i.e. the content of S has the form $\nu K.Key(K, B, C, A)$ then the precondition simply requires that A has obtained the Reply-To nonce n before:

  $$Pre(+S) \equiv HN_{A,\rho}(n)$$

  The effects of sending out such act is that A has freshly generated key K as a session key between B and C. Hence

  $$Effect(+S) \equiv GK_{A,\rho}(K, B, C)$$

  In contrast, if K has been generated, i.e. the content of S has the form $Key(K, B, C, A)$ then the preconditions states also that A must have generated the key K before sending the act:

  $$Pre(+S) \equiv GK_{A,\rho}(K, B, C) \wedge HN_{A,\rho}(n)$$

  In this case, A does not gain any new information or knowledge after sending out such act:

  $$Effect(+S) \equiv True$$

  A receiver of a reply act would receive it only if it has requested for such a act before. Hence the preconditions for B to act as a receiver in a reply act S of type *newkey* is

  $$Pre(-S) \equiv GN_{B,\tau}(n)$$

  The effects for B after receiving S is that B knows K is a fresh key between B and C generated by A, and B also knows that A has received nonce n. Hence

  $$Effect(-S) \equiv Know_{B,\tau}GK_{A,\rho}(K, B, C) \wedge Know_{B,\tau}HN_{A,\rho}(n)$$

- Let the type of S be *keyconfirm* and the content of S be Key(K,A,B,C)

  The preconditions for A to send out S is that A knows that K is a fresh session key between A and B generated by C and A has received nonce n before:

  $$Pre(+S) \equiv Know_{A,\rho}Key(K, A, B, C) \wedge HN_{A,\rho}(n)$$

  A, as the sender in S, does not gain any new information after sending S. Hence:

  $$Effect(+S) \equiv True$$

  The preconditions for B to receive S is that B must have requested for it before, i.e. B has generated the nonce n and B must be informed about K. Hence

  $$Pre(-S) \equiv GN_{B,\tau}(n) \wedge Informed_{B,\tau}(K, A, B, C)$$

  After receiving S, B knows that K is indeed a fresh session key between A and B generated by C and B also knows that A also knows it.

  $$Effect(-S) \equiv Know_{B,\tau}Know_{A,\rho}Key(K, A, B, C) \wedge Know_{B,\tau}HN_{A,\rho}(n)$$

A.3 Inform Acts

Let S be a inform act form as defined in the main text.

- Let the type of S be *newkey* and the content of S be $\nu K.Key(K, B, C, A)$ or Key(K,B,X,Y) where (X,Y) = (A,C) or (C,A). If K has not been generated, i.e. the content of S has the form $\nu K.Key(K, B, C, A)$ then A does not need any specific information to be able to send the act:

  $$Pre(+S) \equiv True$$

  The effect for A as the sender in this case is:

  $$Effect(+S) \equiv GK_{A,\rho}(K, B, C)$$

  In contrast, if the key has been generated before the act, i.e. the content of the act is of the form *Key(K,B,X,Y)* where (X,Y) = (A,C) or (C,A) , then A must have known about K before sending out the act as we assume that regular principals honest and hence would inform others only about what they know:

  $$Pre(+S) \equiv Know_{A,\rho}Key(K, B, X, Y)$$

  In this case, A does not gain any new information after sending B the inform act. Hence

  $$Effect(+S) \equiv True$$

There is no preconditions for B to receive S. Hence

$$Pre(-S) \equiv True$$

After receiving S, B is informed about key K. Hence

$$Effect(-S) \equiv Informed_{B,\tau}(K,B,X,Y)$$

- Let the type of S be *keyconfirm* and its content be Key(K,A,B,C). For A to be able to confirm to B that K is generated by C as a fresh session key between A and B, A should know that K is indeed generated by C as a fresh session key between A and B. Hence

  $$Pre(+S) \equiv Know_{A,\rho}Key(K,A,B,C)$$

  The sender of an inform act of any type obviously does not gain any information after sending the act.

  $$Effect(+S) \equiv True$$

  The preconditions for B to receive an inform act of type *keyconfirm* is that B knows that K is indeed generated by C as a fresh session key between A and B. Hence after receiving the act, B knows also that A knows that K is generated by C as a fresh session key between A and B. Hence

  $$Pre(-S) \equiv Know_{B,\tau}Key(K,A,B,C)$$
  $$Effect(-S) \equiv Know_{B,\tau}Know_{A,\rho}Key(K,A,B,C)$$

B. OTWAY-REES PROTOCOL

(1)$A \rightarrow B:\ M, A, B, \{N_a, M, A, B\}_{K_{AS}}$
(2)$B \rightarrow S:\ M, A, B, \{N_a, M, A, B\}_{K_{AS}}, \{N_b, M, A, B\}_{K_{BS}}$
(3)$S \rightarrow B:\ M, \{N_a, K_{AB}\}_{K_{AS}}, \{N_b, K_{AB}\}_{K_{BS}}$
(4)$B \rightarrow A:\ M, \{N_a, K_{AB}\}_{K_{AS}}$

C. APPENDIX C

Given that the connection topology allows direct communication only between initiators and servers as well as between initiators and responders, the abstract speech act oriented protocol in example **??** is implemented by the following message exchanging protocol:
(1) $A \rightarrow S: req, newkey, A, Init, S, Server, N_a, B$
(2) $S \rightarrow A: \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}}$
(3.1) $S \rightarrow A: \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(3.2) $A \rightarrow B: \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(4) $B \rightarrow A: \{req, keyconfirm, B, Resp, A, Init, N_b, Hash(K), S\}_{K_A}$
(5) $A \rightarrow B: \{rep, keyconfirm, A, Init, B, Resp, N_b\}_{K_B}$
(6) $B \rightarrow A: \{inf, keyconfirm, B, Resp, A, Init, S\}_K$
Steps (2) and (3.1) could be combined by letting S sending both m_{S_2}, m_{S_3} to A in one step resulting in:
(1) $A \rightarrow S:\ req, newkey, A, Init, S, Server, N_a, B$
(2') $S \rightarrow A: \{rep, newkey, S, Server, A, Init, N_a, K, B\}_{K_{AS}},$
$\{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(3') $A \rightarrow B: \{inf, newkey, S, Server, B, Resp, K, B, A, S\}_{K_{BS}}$
(4) $B \rightarrow A: \{req, keyconfirm, B, Resp, A, Init, N_b, Hash(K), S\}_{K_A}$
(5) $A \rightarrow B: \{rep, keyconfirm, A, Init, B, Resp, N_b\}_{K_B}$
(6) $B \rightarrow A: \{inf, keyconfirm, B, Resp, A, Init, S\}_K$

D. APPENDIX D

For each bundle $\mathcal{B}$, define $\prec_{\mathcal{B}}$ to be the transitive closure of $\rightarrow \cup \Rightarrow$.

THEOREM 3. *(Soundness Theorem)*

1. If $\vdash F$ *then* $\models F$
2. If $\vdash AP.R[F]$ *then* $\models AP.R[F]$

Proof*(Sketch) The full proof is more than 15 pages long. We hence will give only a sketch of it here.*

1. Let $\vdash F$. By induction on the structure of a derivation of F wrt the axioms and proof rules of ProtoLog.
   (a) Base case: F is an axiom of ProtoLog. It is obvious that $\models F$ holds for $F \in \{A1 - A6, A8 - A10\}$ or F is an axiom of the S5 system. It remains to show that $\models F$ for $F \in \{A7, A11, A12\}$.
      i. Let $F = A11$. Assume the contrary. Therefore there exists a historical bundle $\mathcal{B}$ and regular principal identifiers A,B,C,D such that $\mathcal{B} \models Informed_{A,\rho}(K,B,C,D)$ and $A \notin \{B,C,D\}$. Hence there is a recent regular strand $s \in \mathcal{B}$ of A in role ρ and a node N in s such that N implements a sending or receiving event of an inform or reply act and K is associated with B,C,D at N. Let S be the speech act implemented by N. K is associated with B,C,D in S. From the structure of the speech acts, it follows that $A \in \{B,C,D\}$. Contradiction.
      ii. Let F = A12. Let $\mathcal{B}$ be a historical bundle such that $\mathcal{B} \models Key(K,A,B,C)$. It is enough to prove the following two assertions:
      1) For each recent regular strand $s \in \mathcal{B}$, for each node N of s, if K occurs in the speech act S represented by term(N) then K is associated with A,B,C in S .
      Proof: Let Ω be the set of all nodes N' in $\mathcal{B}$ such that $Effect(N') \vdash Key(K,A,B,C)$. It is clear that Ω is not empty. Let N be a minimal node in Ω wrt $\preceq_{\mathcal{B}}$. By case analysis, we can prove that N implements a sending event of an reply or inform act of type *newkey* and K is generated at N. Let Σ be the set of regular nodes N' in $\mathcal{B}$ in which K occurs in the speech act represented by N'. We prove by induction wrt to the restriction of $\preceq_{\mathcal{B}}$ on Σ that K is a key associated with A,B,C at all nodes in Σ. The assertion hence follows.
      2) $\mathcal{B} \not\models Access(PE,K)$.
      Proof: Assume the contrary. Applying the above assertion and analyzing the structure of the representations of speech acts, we will get a contradiction.
      iii. Let F = A7. Let $\mathcal{B}$ be a historical bundle such that $\mathcal{B} \models Know_{A,\rho}Key(K,B,C,D)$ for regular principal identifiers A,B,C,D. It is not difficult to see that $A \in \{B,C,D\}$ for otherwise we could show that for the bundle $\mathcal{B}_0$ consisting of those strands in $\mathcal{B}$ that are either a strand of A or precede a strand of A, $\mathcal{B}_0 \models Know_{A,\rho}Key(K,B,C,D)$ and K does not appear in $\mathcal{B}_0$. Contradiction to the property that $\mathcal{B}_0 \models Key(K,B,C,D)$.
      Suppose now that $\mathcal{B} \not\models Informed_{A,\rho}(K,B,C,D)$. It follows from the definition of the relation $\models$, that for each strand s of A in $\mathcal{B}$, if K occurs in

s then K occurs in a sending or receiving event of a request act. Hence K occurs in the sending or receiving event of a request act of type *key-confirm*. Let N be a node implementing such a event and N be the first node on its strand containing K. Because N represents a sending or receiving event of a request act of type *keyconfirm*, and the protocol is well-designed, there must be a node M preceding N on the same strand also containing K. Contradiction.

(b) Inductive case: We only need to show that if $\models F$, then $\models Know_{A,\rho}F$ for all regular principal identifier A and ρ. This assertion follows immediately from the definition of model semantics.

2. After having proved assertion (1), we only need to show the validity of the propagation, consequence and initiation rules wrt the model semantics. The validity of the consequence and initiation rules wrt the model semantics follows immediately from the definition of the model semantics. The validity of the propagation rule is proved by analyzing the structure of Effect(E). The proof is long (more than 10 pages) and tedious. We refer the readers to the full paper (not listed in the reference to keep the paper anonymous).

E. APPENDIX E

Let T be a prefix of the ρ-track AP_ρ of AP such that R is a ground instance of T. Let θ be the most general unifier between T and R. Let θ' be a ground substitution over the variables appearing in AP such that θ, θ' coincide for the variables appearing in T and θ' assigns distinct constants to distinct variables not in T. Let $\mathcal{B}$ denote a historical bundle of AP satisfying following properties:

1) There is no penetrator node in $\mathcal{B}$

2) There is no past bundle in $\mathcal{B}$

3) For each role ρ that appears in AP, there is exactly a strand in $\mathcal{B}$ implementing $AP_\rho\theta'$

4) For each strand s in $\mathcal{B}$ there is exactly a role ρ that appears in AP such that s implements $AP_\rho\theta'$

The existence of $\mathcal{B}$ is obvious. Let s be the strand in $\mathcal{B}$ that implements R. Let $\mathcal{B}'$ denote the bundle obtained from $\mathcal{B}$ by removing all nodes that do not precede a node in s. $\mathcal{B}'$ is called the *ideal scenario* of R, denoted by *IS(R)*. It is easy to see that IS(R) is a historical bundle of AP.

Theorem(Limited Completeness Theorem)
If $\models AP.R[Know_{A,\rho}Key(K,B,C,D)]$
then $\vdash AP.R[Know_{A,\rho}Key(K,B,C,D)]$

Proof:
Assume the contrary that $\models AP.R[Know_{A,\rho}Key(K,B,C,D)]$ holds but $\vdash AP.R[Know_{A,\rho}Key(K,B,C,D)]$ does not hold. There are three possible cases: 1) K does not appear in R, or 2) K appears in R but is not associated with B,C,D, or 3) K appears in R and is associated with B,C,D. Suppose that the first two cases happen. Then it is clear that $IS(R) \not\models Key(K,B,C,D)$, contradiction. Therefore it holds that K appears in R and is associated with B,C,D. From $\not\vdash AP.R[Know_{A,\rho}Key(K,B,C,D)]$, it is not difficult to see that 1) all receiving events in R are receiving events of inform or request acts of type *newkey*, and 2) all sending events in R are sending events of request acts. Let s be the strand in $IS(R)$ that implements R. Let r be a new strand that is exactly like s with the exception that their nonces if exist are different. Construct a historical bundle $\mathcal{B}$ of AP from $IS(R)$ by 1) adding r to IS(R) and 2) connect the receiving nodes in r with the corresponding sending nodes in IS(R) and 3) assign the past status to all strands in IS(R), and the recent status to r. It is obvious that $\mathcal{B} \not\models Key(K,B,C,D)]$. Contradiction. The theorem is proved.