Trust–Based Navigation in Distributed Systems

Raphael Yahalom* The Hebrew University, Jerusalem

Birgit Klein** and Thomas Beth University of Karlsruhe

ABSTRACT: In distributed environments entities may be associated with arbitrary trust assumptions regarding other entities. A method is presented for analyzing distributed protocols that are designed to reflect such particular trust relations. A new protocol for obtaining public keys in a multidomain environment without necessarily assuming that all trust relations correspond to a global hierarchical structure is presented and analyzed using the new method. The potential advantages of this approach over previous ones are discussed.

* Raphael Yahalom was supported in part by grants from the Recanati and Schonbrunn Foundations at the Hebrew University, Jerusalem.

** Birgit Klein was supported by the German Ministry of Research and Technology in the REMO (REference MOdel for secure systems) project under contract number 01 IS 202 C. REMO is a joint venture of the European Institute for System Security at the University of Karlsruhe, the German National Research Center for Computer Science (GMD), the Industrieanlagen Betriebsgesellschaft (IABG), Siemens AG, and Telematic Services mbH (TELES).

1. Introduction

The preservation of confidentiality and integrity of messages exchanged between entities in widely distributed multidomain environments is the goal of many research and implementation efforts. For example, systems like *Privacy Enhanced Mail* (PEM) (Kent 1993, Linn 1993) have recently received considerable attention.

In such environments, possibly consisting of millions of entities, issues such as name management, key management, and autonomy of local domains need to be considered if realistic, scalable solutions are to be implemented. Approaches for dealing with such issues are often based on a global hierarchy of entities. For example, the PEM design is based on an X.500 naming hierarchy (CCITT 1988a) and on an X.509 authentication/certification hierarchy (CCITT 1988b). Such hierarchical structures capture, in addition to naming information, trust and key-sharing relations between entities.

Such an underlying global hierarchy is associated with various advantages. It is conceptually simple and elegant, it scales well, and it constitutes a natural framework for establishing management policies. However, a hierarchy may in some cases be over-constraining (Birrell et al. 1986, Lampson et al. 1991, Gligor et al. 1992, Yahalom et al. 1993). In particular, there are real-world situations in which an entity does not necessarily trust all other entities at higher hierarchical levels but may still wish, and be able to, securely interact with their subordinates. In (Birrell et al. 1986, Lampson et al. 1991, Gligor et al. 1992) particular extensions to a hierarchical structure were proposed and examined. However, for each such extension, it can be shown that there are particular circumstances in which relationships between entities cannot be supported and thus that even the extended structure is potentially over-restrictive. Consequently, in (Yahalom et al. 1993) an approach was introduced to incorporate trust relations that do not necessarily correspond to any underlying global structure.

In this article we establish how trust may be acquired by entities during protocol executions. Such an approach for acquiring trust significantly extends the model in (Yahalom et al. 1993) in which the trust can be considered only as *potential*. The method we present for analyzing acquired trust extends the logic-based class of protocol analysis methods [e.g., (Burrows et al. 1989; Gong et al. 1990; Abadi & Tuttle 1991)].

We present a general protocol for obtaining public keys in a multidomain environment that relies on such acquired trust. Our approach results in a protocol that may rely on the properties of a global hierarchical structure, but may also allow any deviations from that structure, increasing the flexibility as required. Whereas the optimal point in the trade-off between flexibility and scalability is, of course, environment dependent, it seems likely that trust relations that are not consistent with the conventional ones are the exception rather than the rule. Consequently, the protocol presented here would normally perform well, but, as required, may also be sufficiently flexible. Our approach for establishing acquired trust enables us to analyze such protocols.

The rest of this article is organized as follows. In the next section we introduce our model and present the trust acquisition rules. In section 3, we describe the protocol execution environment assumed. In section 4 a general *trust acquisition based* protocol for this environment is presented, and its properties analyzed using the model of section 2. In section 5 the execution of the protocol in particular circumstances is demonstrated as an example. General comparisons to related work are presented in section 6, and in section 7 we conclude.

2. *Incorporating Trust in Distributed Executions*

A distributed system consists of multiple entities interconnected by communication links. Entities may communicate by sending messages over these links. Each entity is associated with a unique identifier. Adversaries may read, modify, generate, or replay messages. Consequently cryptographic schemes need to be used for securing message exchanges. We consider in this article only public key schemes (e.g., (Rivers et al. 1978; ElGamal 1985)) and assume that the usual strong properties are approximated.

A protocol defines a sequence of messages to be exchanged as well as the corresponding local actions to be taken by the participating entities. Each entity is associated with a state that is initialized at the beginning of a protocol execution and may be updated during such an execution.

Similarly to the Gong, Needham, and Yahalom (GNY) model (Gong et al. 1990), we assume that an entity's state consists of two parts. One part contains the items that the entity *possesses* at each execution point. Another part contains the *beliefs* of that entity about items and about other entities. The notion of a *belief* is similar to that which was introduced by Burrows, Abadi, and

Needham (BAN) (Burrows et al. 1989) and that was also used by GNY and by Abadi and Tuttle (AT) (Abadi & Tuttle 1991).

Like the GNY work, we use $\ni$ to denote *possesses* and $\models$ to denote *believes*. For example, the expression $P \ni K_{Q+}$ represents the fact that K_{Q+} is possessed by entity P (i.e., at a particular point in the execution the item K_{Q+} is in the *possessions* part of P's state). The expression $P \models \overrightarrow{K_{Q+}}$ represents the fact that P *believes* that K_{Q+} is the public key of Q (i.e., similarly, that belief is part of P's *beliefs* state at a particular execution point). $\lhd$ denotes a message *seen* by P at some point in an execution. $\{msg\}_K$ denotes a message *msg* encrypted with a key K.

Rules such as the following one establish (in the spirit of (Burrows et al. 1989, Gong et al. 1990, Abadi & Tuttle 1991)) how entities' state may change during an execution:

$$\frac{P \ni K_{Q+}, P \lhd \{msg\}_{K_{Q-}}}{P \ni msg}$$

If during an execution, P possesses K_{Q+}, and P sees $\{msg\}_{K_{Q-}}$ (K_{Q-} denotes Q's private key) then P may also possess msg.

A particular type of belief is *trust*. In (Yahalom et al. 1993) a few primitive trust classes were introduced. The analysis in this article focuses on three of these.

Trust with respect to identification, denoted $P \cdot$ *trusts*$_{id-\tau}$ (*Q*) [1]

MEANING. By participating in some appropriate *binding* protocol (or being properly initialized), the entity Q is assumed by P to have in a dedicated part of its state bindings between unique identifiers of entities in the set τ, their correct public keys, and possibly also the corresponding key–expiration–time. Modifications of this *binding* state are only possible as a result of executing the appropriate *binding* protocol.

Trust with respect to maintaining relatively synchronized clocks, denoted $P \cdot$ *trusts*$_{sc}$ (*Q*)

MEANING. By participating in some appropriate *clock synchronization* protocol (perhaps with some time server) and relying on its own clock rate, Q's clock is assumed by P to be always closely synchronized with P's clock. That is, there is always a difference of at most Δ ticks (for some Δ) between these two clock readings.

1. The notion of trust here represents *actual trust*, which is stored in an entity's state (and so is also referred to as *stored trust*). That is different from the derived trust notion presented in (Yahalom et al. 1993), which represents *potential* trust between entities. We use "$\cdot$*trusts*" in contrast to "*trusts*" in (Yahalom et al. 1993) to represent such stored trust.

Trust with respect to executing the algorithmic protocol steps correctly, denoted $P \cdot trusts_{ps}\ (Q)$

MEANING. For a given protocol, the entity Q is assumed by P to always perform the algorithmic steps of the protocol correctly and consistently with the protocol specifications. In particular, Q is assumed to associate the correct interpretation with its input and output messages, take the appropriate required local actions, and truthfully represent its execution state in any statement it makes.

Note that trust in one aspect does not necessarily imply trust in another. For example, a certification authority that is trusted with respect to the provision of reliable bindings between identifiers in its domain and public keys with expiration times may not necessarily be able at all times to keep its clock from drifting (or maliciously being set) significantly backwards or forwards.

In this article we consider only cases in which an entity may trust another in (at least) all the three aspects above. Let $\cdot trusts_{y_\tau}$ represent the three statements $\cdot trusts_{id-\tau}$, $\cdot trusts_{ps}$, and $\cdot trusts_{sc}$. As is described below, such a trust may either be *direct* or be acquired by obtaining *recommendations* from other entities. Let seq denote a (possibly empty) sequence of entities. Then seq in the expression $P \cdot trusts^{seq}_{y_\tau}(Q)$ represents the (possibly empty) sequence of entities whose recommendations were used (in the sense defined in rules 2–5 below) to acquire that trust.

The following rule enables an entity to gain a belief regarding another's public key.

RULE 1.

$$\frac{P \cdot trusts^{seq}_{y_\tau}\ (Q),\ \ P \ni K_{Q+},\ \ P \models \overrightarrow{K_{Q+}}, P \triangleleft \{ts, R, \overrightarrow{K_{R+}}\}_{K_{Q-}}, R \in \tau}{P \models \overrightarrow{K_{R+}}}$$

ANALYSIS. P possesses an item that it believes to be Q's public-key (condition 2 and 3). With that key P can believe that:

- Q said that K_{R+} is R's public-key: because of conditions 3, 4, and the assumptions regarding the encryption scheme.[2]

2. The fact that this message format is associated with such a *meaning* can be supported if necessary, for example, by incorporating a message-type field in the message.

- Q said it *recently* (e.g., within the current protocol execution): Q included a timestamp in the message, P trusts Q with respect to relatively synchronized clocks, and P can check that the ts value is sufficiently close to the current time.

- Q's above statement can be considered by P to be reliable: P trusts Q with respect to maintaining correct id-keys binding for entities in τ, and R is in τ (condition 5). Also, P trusts Q with respect to performing algorithmic steps correctly and so believes that Q's statements truthfully represent its state.

- Statements such as Q's above are assumed, throughout this article, to contain expiration-time information along with the actual key.

Consequently, P may believe that K_{R+} is R's public key. [3] ○

An entity may be willing to trust a second one (in some respect) because it established that a third one trusts the second (in the same respect). This is referred to as an entity trusting the *recommendation* of another about the trustworthiness of yet another entity. In particular, if P trusts the recommendations of Q, a dedicated part of Q's state is assumed by P to contain reliable bindings between entities and trustworthiness aspects.

Similarly, an entity P may trust another with respect to recommendations about another entity which can recommend yet another entity which ... which can recommend the trustworthiness of some entity. However, for such an expression to be useful, certain constraints may be imposed by P on such a transitive closure. For example, P may wish to consider recommendations only *from* entities within a certain organization, restrict the number of (transitive) recommendations, or consider recommendations in some respect only *about* entities that are not located in certain countries.

Such constraints are classified to *path constraints* ($\mathcal{S}p$)—representing a set constraining the possible *recommending* entities and *target constraints* ($\mathcal{S}t$)—representing a set constraining the possible *recommended* entities. As was demonstrated in (Yahalom et al. 1993), the specification of these sets can be based on such parameters as *the recommendation path traversed thus far*, etc.

The expression

$$P_i \cdot \textit{trusts.rec}^{seq}_{y_\tau}\ (P_j)\ \textit{when.path}[\mathcal{S}p_{i,j}]\textit{when.target}[\mathcal{S}t_{i,j}]$$

represents the fact that P_i trusts the recommendations of P_j about the identity of entities that may be trusted to recommend entities which ... which may be trusted

3. Note that P may obviously also *possess* K_{R+} after decrypting the message with Q's public key.

to recommend entities which may be trusted with respect to the above three aspects, as long as each recommending entity after P_j is a member in the set $\mathcal{S}p_{i,j}$ and every entity recommended is a member in the set $\mathcal{S}t_{i,j}$. Note that each entity P_i may store multiple such expressions representing different associations between potential recommending and recommended entities. *seq* represents a (possibly empty) sequence of entities via which this trust expression was obtained by P_i (see rule 3 below). The trust recommendation expression thus allows entities to acquire, in particular circumstances, new trusts.

For the sake of simplicity we assume for the rest of this article that if $P_i \cdot \mathit{trusts.rec}_{y_\tau}(P_j)$ then also $P_i \cdot \mathit{trusts}_{y_\tau}(P_j)$.

The next two rules establish how new stored trust expressions may be acquired during an execution (we use both "," and "**and**" to denote logical *and*):

RULE 2.

$$\frac{\begin{array}{c} P \cdot trusts.rec^{seq}_{y_\tau}(Q)\ when.path[\mathcal{S}p]\ when.target[\mathcal{S}t] \\ \textbf{and}\ P \ni K_{Q+},\ P \models \overrightarrow{K_{Q+}},\ P \lhd \{ts, Q \cdot trusts^{seq'}_{y_{\tau'}}(R)\}_{K_{Q-}} \\ \textbf{and}\ R \in \mathcal{S}t, (\forall P_i \in seq' : P_i \in \mathcal{S}p) \end{array}}{P \cdot trusts^{seq \bullet Q \bullet seq'}_{y_{\tau \cap \tau'}}(R)}$$

ANALYSIS: P possesses an item that it believes to be Q's public key (conditions 2 and 3).

For reasons analogous to the ones in the analysis of rule 1, when P sees the message (fourth condition) it can believe that Q made the corresponding statement recently. That statement is Q's recommendation about R.

P trusts, with certain constraints, Q's recommendations regarding trustworthy entities (condition 1).

P's *target constraints set*—specifying its constraints regarding entities that can be recommended—includes R (fifth condition) and so R is an acceptable recommended entity for P.

According to the definitions of the trust expressions, seq and seq' represent the sequence of entities whose recommendations were used to derive the corresponding trust expressions (the expressions in the first and fourth conditions, respectively).[4]

4. *seq* is empty if the expression is a direct trust—it was not acquired as a result of recommendations by other entities.

The entities used in order for Q to acquire its trust expression in R (seq') are all included in the path constraints set associated with P's trust expression (sixth condition).

Consequently, P may accept Q's recommendation and acquire a new trust in R. The sequence of entities associated with that new expression is the concatenation of the two sequences associated with the previous expressions. The set of the entities which may be identified is the intersection of the corresponding sets (τ and τ') in the previous expressions. ○

RULE 3.

$$\frac{\begin{array}{c} P \cdot trusts.rec_{y_\tau}^{seq}\ (Q)\ when.path[\mathcal{S}p]\ when.target[\mathcal{S}t] \\ \textbf{and } P \lhd \{ts, Q \cdot trusts.rec_{y_{\tau'}}^{seq'}\ (R)\ when.path[\mathcal{S}p']\ when.target[\mathcal{S}t']\}_{K_{Q-}} \\ \textbf{and } P \ni K_{Q+},\ P \models \overrightarrow{K_{Q+}},\ R \in \mathcal{S}p, (\forall P_i \in seq' : P_i \in \mathcal{S}p) \end{array}}{P \cdot trusts.rec_{y_{\tau \cap \tau'}}^{seq \bullet Q \bullet seq'}\ (R)\ when.path[\mathcal{S}p \cap \mathcal{S}p']\ when.target[\mathcal{S}t \cap \mathcal{S}t']}$$

ANALYSIS. Analogous to the analysis of rule 2 except that here we are deriving an expression about P's trust in the recommendation of R. Consequently, R has to be a member in P's path constraints set $\mathcal{S}p$, that is, it is an acceptable recommending entity for P (condition 5).

The constraints associated with the new acquired expression need to reflect the corresponding constraints of the two expressions from which it was derived. Hence, representations of the new constraint sets are the representation of the intersection of the corresponding constraint sets.[5] ○

Finally, the following two rules enable an entity to acquire a new trust expression based on consistency checks that are performed by an entity it trusts. Note that such checks are considered as algorithmic steps and so an entity that trusts another to follow such steps correctly may trust it to perform these consistency checks appropriately.

5. Note that if one assumes, as we did above, that $P \cdot trusts.rec_{y_\tau}^{seq}\ (Q)$ implies $P \cdot trusts_{y_\tau}^{seq}\ (Q)$, then an additional conclusion of the form $P \cdot trusts_{y_{\tau \cap \tau'}}^{seq \bullet Q \bullet seq'}\ (R)$ may be added to rule 3. Such a conclusion requires another condition for that rule, namely $R \in \mathcal{S}t$.

RULE 4.

$$\frac{\begin{array}{c} P \cdot trusts.rec_{y_\tau}^{seq}\ (Q)\ when.path[Sp]\ when.target[St] \\ \textbf{and } P \models Q \ni (Sp, St, \tau),\ P \ni K_{Q+},\ P \models \overrightarrow{K_{Q+}} \\ \textbf{and } P \triangleleft \{ts, \text{P is allowed to } \cdot trusts_{y_{\tau'}}\ \text{R}, seq'\}_{K_{Q-}}, \end{array}}{P \cdot trusts_{y_{\tau'}}^{seq \bullet Q \bullet seq'}\ (R)}$$

ANALYSIS. P believes that Q possesses the constraints associated with P's trust in Q's recommendations (condition 2). P trusts Q to correctly follow the required algorithmic steps (assumption about any such trust recommendation expression). Consequently P may accept Q's statements of the form of (condition 5) as valid, and interpret them as represented in the new acquired expression. ○

RULE 5.

$$\frac{\begin{array}{c} P \cdot trusts.rec_{y_\tau}^{seq}\ (Q)\ when.path[Sp]\ when.target[St] \\ \textbf{and } P \models Q \ni (Sp, St, \tau),\ P \ni K_{Q+},\ P \models \overrightarrow{K_{Q+}} \\ \textbf{and } P \triangleleft \{ts, \text{P is allowed to } \cdot trusts.rec_{y_{\tau'}}\ \text{R}, seq', Sp', St'\}_{K_{Q-}}, \end{array}}{P \cdot trusts.rec_{y_{\tau'}}^{seq \bullet Q \bullet seq'}\ (R)\ when.path[Sp']\ when.target[St']}$$

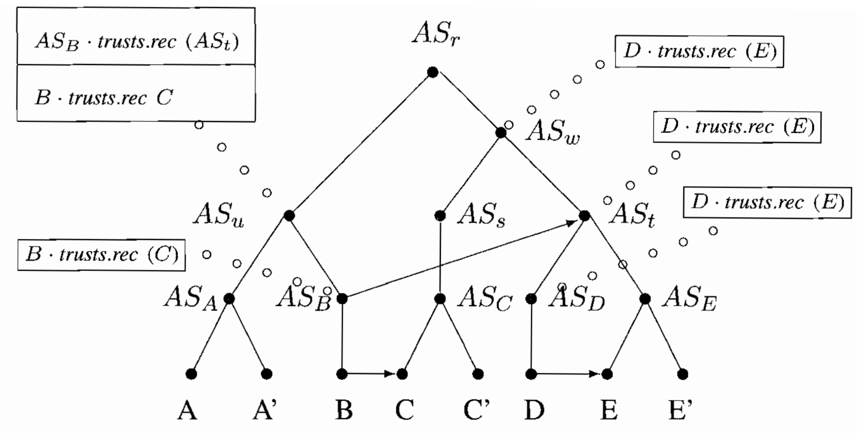
ANALYSIS. Analogous to the analysis of rule 4.

3. *The Execution Environment*

In the next section we introduce a protocol for obtaining public keys in a multidomain environment. In this section we present a model of the environment in which that protocol is assumed to execute.

We assume a *hierarchical name space* (e.g., such as that provided by the X.500 directory service (CCITT 1988a)) Each entity may have some *initial* possessions and beliefs and some *initial* trust expressions stored in its state before the beginning of a protocol execution. For the sake of simplicity we assume that $P_i \cdot trusts_{y_\tau}\ (P_j)$ or $P_i \cdot trusts.rec_{y_\tau}\ (P_j)$ implies $P_i \ni K_{P_j+}$ and $P_i \models \overrightarrow{K_{P_j+}}$ for each such initial trust expression P_i has.[6]

6. In general, a similar approach can be extended for cases in which an entity initially trusts another but does not possess its public key.



$AS_B \cdot trusts.rec\ (AS_t)$
$B \cdot trusts.rec\ C$
AS_r
$D \cdot trusts.rec\ (E)$
AS_w
$D \cdot trusts.rec\ (E)$
AS_u
AS_s
AS_t
$D \cdot trusts.rec\ (E)$
$B \cdot trusts.rec\ (C)$
AS_A
AS_B
AS_C
AS_D
AS_E
A
A'
B
C
C'
D
E
E'

Figure 1. A Hierarchy with Cross Links.

It seems reasonable that an entity's stored trust relations would normally correspond to the conventional ones implied by the hierarchy—namely each entity trusts the recommendations of and about entities on a path up and then down in the hierarchy. As was demonstrated in (Yahalom et al. 1993), the trust recommendation constraints may be used to express that. We refer to expressions representing extensions or restrictions with respect to the conventional up and down paths as *exceptional trust expressions.* Trust of one entity in the recommendations of another which is not its parent or child in the hierarchy is referred to as a *cross link.*

Finally, each entity contains a *trust-based routing table.* Such a routing table at an entity contains the exceptional expressions of all its descendants in the hierarchy. As we shall see, the information in these routing tables is used to optimize the navigation in a network. The correctness and freshness of the information they contain may only affect the execution performance (and not its correctness). Therefore the contents of such tables can be regarded as cached hints.

Assume a hierarchy of entities A, A', B, C, C', D, E, E', AS_A, AS_B, AS_C, AS_D, AS_E, AS_s, AS_t, AS_u, AS_w, AS_r as shown in Figure 1 and additional trust relations expressions of the following form representing cross-links:

$$AS_B \cdot trusts.rec_{y_{\tau_1}}\ (AS_t)\ when.path[Sp_1]\ when.target[St_1]$$

$$B \cdot trusts.rec_{y_{\tau_2}}\ (C)\ when.path[Sp_2]\ when.target[St_2]$$

$$D \cdot trusts.rec_{y_{\tau_3}}\ (E)\ when.path[Sp_3]\ when.target[St_3]$$

There are four different potential trust paths between A and E:

1. $A, AS_A, AS_u, AS_B, AS_t, AS_E, E$
2. $A, AS_A, AS_u, AS_r, AS_w, AS_t, AS_E, E$
3. $A, AS_A, AS_u, AS_B, B, C, AS_C, AS_s, AS_w, AS_t, AS_E, E$
4. $A, AS_A, AS_u, AS_B, B, C, AS_C, AS_s, AS_w, AS_t, AS_D, D, E$

Only the second path corresponds to a normal up and down traversal of a hierarchy. If, for example, A does not trust a node such as AS_E, it may still be able to set up a secure session with E.

4. *Obtaining an Entity's Public Key*

4.1. *A Protocol*

Assume that in the environment we described an entity A wishes to send an entity B a secret message that only B will be able to read. A thus needs to obtain B's public key from a source it considers trustworthy in the relevant aspects. In this subsection we present a protocol for obtaining such a key. Each *navigational* next step decision in the protocol's execution is based on an algorithm presented in the next subsection. The properties of the protocol are then analyzed using the model we have presented.

Informally, there are three types of messages in the protocol. A forward *public-key (pk) request* message includes information such as the path traversed thus far and the corresponding constraints sets. A backward *dead-end* message represents the fact that some path could not lead to a trusted source of the desirable public key. A backward *public-key response* message includes a binding between an entity and its public key along with the path that was used to obtain that public key.[7]

Below, A denotes the initiating entity and B the target entity whose public key A requires. All other entities are denoted as AS_i, AS_j, etc. S_n is a unique session identifier generated by A for each one of its requests. The global uniqueness of S_n can be guaranteed by a prefix which contains A's unique name. $AS_i \rightarrow AS_j : msg$, denotes a message msg sent from AS_i to AS_j. ts denotes a timestamp.

7. Note that, as established in the protocol analysis in section 4.3 and appendix A, the trust recommendation path need not necessarily be included in the *pk-response* messages. However, such information may in some cases be useful and so is included in our current description of the protocol.

A :

Initiates the protocol by sending a *pk-request* message to some entity AS_A for which there is an expression of the form $A \cdot \textit{trusts}^{seq}_{y_\tau}(AS_A)$ or $A \cdot \textit{trusts.rec}^{seq}_{y_\tau}(AS_A)$ in A's state [8]:

$A \rightarrow AS_A : \{\{S_n, ts, [A, AS_A], K_{B+}?, \mathcal{S}p', \mathcal{S}t', \tau\}_{K_{A-}}\}_{K_{AS_A+}}$

where $K_{B+}?$ represents a request for B's public key, and $\mathcal{S}p'$, $\mathcal{S}t'$, τ represent the constraints in the corresponding expression $A \cdot \textit{trusts.rec}^{seq}_{y_\tau}(AS_A)$ (in the case of an $A \cdot \textit{trusts}^{seq}_{y_\tau}(AS_A)$ expression the sets $\mathcal{S}p'$ and $\mathcal{S}t'$ are empty).

Each entity AS_j:

When receiving a *pk-request* message from AS_i:

```
$AS_i \rightarrow AS_j$ :
$\{\{S_n, ts, [A, AS_A, ..., AS_i, AS_j], K_{B+}?, \mathcal{S}p', \mathcal{S}t', \tau\}_{K_{AS_i-}}\}_{K_{AS_j+}}$
If $AS_j$ possesses an item which it believes is B's public key $K_{B+}$
then: Generate a pk-response message [9] and send it to $AS_i$:
    $AS_j \rightarrow AS_i$ :
    $\{\{S_n, ts, [A, AS_A, ..., AS_i, AS_j, AS_i], B, K_{B+}\}_{K_{AS_j-}}\}_{K_{AS_i+}}$
else: Store an $S_n$ record:
    $S_n, ts, [A, AS_A, ..., AS_i, AS_j], K_{B+}?, \mathcal{S}p', \mathcal{S}t', \tau, DE$
    (where the dead-end set $DE$ is empty)
    Perform Next-Step($S_n$) algorithm
    If Next-Step($S_n$) is null (no possible next step)
    then: Delete $S_n$ record
        Send to $AS_i$ a dead-end message:
        $S_n, ts, AS_j \rightarrow AS_i : \{\{AS_i, AS_j, deadend\}_{K_{AS_j-}}\}_{K_{AS_i+}}$
    else (Next-Step($S_n$) is $AS_k$):
        $\mathcal{S}p'' := \mathcal{S}p' \cap \mathcal{S}p_{AS_j,AS_k}$
        $\mathcal{S}t'' := \mathcal{S}t' \cap \mathcal{S}t_{AS_j,AS_k}$
        $\tau'' := \tau \cap \tau_{j,k}$
```

8. The choice to which such AS_A the initial message will be sent can be determined by considerations similar to these specified below for the general case *next-step* selection.

where $\mathcal{S}p_{AS_j,AS_k}$ and $\mathcal{S}t_{AS_j,AS_k}$ are, respectively, the path and target constraints in an expression $AS_j \cdot \textit{trusts.rec}^{seq'}_{y_{\tau_{j,k}}}(AS_k)$ statement in AS_j's state. If there is only a $AS_j \cdot \textit{trusts}^{seq'}_{y_{\tau_{j,k}}}(AS_k)$ expression then the sets $\mathcal{S}p_{AS_j,AS_k}$ and $\mathcal{S}t_{AS_j,AS_k}$ (and so the new $\mathcal{S}p'$ and $\mathcal{S}t'$) are empty.

Send a *pk-request* message to AS_k:

$AS_j \rightarrow AS_k$:

$\{\{S_n, ts, [A, AS_A, ..., AS_i, AS_j, seq', AS_k],$
$K_{B+}?, \mathcal{S}p'', \mathcal{S}t'', \tau''\}_{K_{AS_j-}}\}_{K_{AS_k+}}$

When receiving a *dead-end* message from AS_k:

$AS_k \rightarrow AS_j : \{\{S_n, ts, AS_k, AS_j, deadend\}_{K_{AS_i-}}\}_{K_{AS_j+}}$

Add AS_k to the dead-end set DE of the S_n record

Perform Next-Step(S_n) algorithm

If Next-Step(S_n) is null

then: Delete S_n record

Send to AS_i (where AS_i is the last entity in the sequence stored in the S_n record) a dead-end message:

$AS_j \rightarrow AS_i : \{\{S_n, ts, AS_i, AS_j, deadend\}_{K_{AS_j-}}\}_{K_{AS_i+}}$

else (Next-Step(S_n) is AS_l):

$\mathcal{S}p'' := \mathcal{S}p' \cap \mathcal{S}p_{AS_j,AS_l}$

$\mathcal{S}t'' := \mathcal{S}t' \cap \mathcal{S}t_{AS_j,AS_l}$

$\tau'' := \tau \cap \tau_{j,l}$

Send a *pk-request* message to AS_l:

$AS_j \rightarrow AS_l$:

$\{\{S_n, ts, [A, AS_A, ..., AS_i, AS_j, AS_l], K_{B+}?,$
$\mathcal{S}p'', \mathcal{S}t'', \tau''\}_{K_{AS_j-}}\}_{K_{AS_l+}}$

(where, as above, the various fields are derived from the S_n record.)

9. That message is encrypted with the recipient's public key to prevent unnecessary disclosure of the potentially sensitive trust relation information. Such encryption may in some cases not be necessary. Also note that throughout we assume that a recipient always checks the value of a ts field and ignores messages it considers too old.

When receiving a *pk-response* message from AS_k:

$AS_k \rightarrow AS_j$:

$\{\{S_n, ts, [A, AS_A, .., AS_i, AS_j, AS_k, \ldots, AS_l, \ldots, AS_k, AS_j],$
$B, K_{B+}\}_{K_{AS_k-}}\}_{K_{AS_j+}}$

Delete the S_n record

Send a *pk-respond* message to AS_i:

$AS_j \rightarrow AS_i$:

$\{\{S_n, ts, [A, AS_A, .., AS_i, AS_j, AS_k, \ldots, AS_l, \ldots, AS_k, AS_j, AS_i],$
$B, K_{B+}\}_{K_{AS_j-}}\}_{K_{AS_i+}}$

A:

After it receives the *pk-response* message from AS_A,[10] A may send secret message to B:

$A \rightarrow B : \{A, B, ts, message\}_{K_{B+}}$

4.2. The Next Step Algorithm

The algorithm presented below is performed by each entity participating in a protocol execution to determine the entity to which it should forward a *public key request* message.

Such a choice has two dimensions:

Correctness—The next entity should be consistent with constraints implied by the execution path up to the current point.

Optimization—The next entity should, if possible, eventually lead us to the appropriate target entity. The path should not contain more steps than necessary.

The Next-Step algorithm at some entity AS_j accepts a S_n record of the form

$$S_n, ts, SEQ, K_{B+}?, Sp', St', \tau, DE$$

as an input, and returns the next entity to which a *public key request* message should be sent.

An *eligible* such next entity AS_e is one for which all the following conditions hold:

10. If A receives a deadend message from every such AS_A that it trusts, then A can conclude that it is impossible for it to obtain B's public key from a trusted source (see section 4.3). As discussed in section 4.3, *time-out* mechanisms can be incorporated to abort particular path executions in the face of long delays (e.g., due to system failures).

- There are $AS_j \cdot$ *trusts*$^{seq}_{y_{\tau''}}$ (AS_e) or AS_j $\cdot$ *trusts.rec*$^{seq}_{y_{\tau''}}$ (AS_e) expressions in AS_j's state.
- $B \in (\tau \cap \tau'')$ (B may be identified along such path).
- $AS_e \in \mathcal{S}p'$ or $AS_e \in \mathcal{S}t'$ (consistency with all the path and target constraints).
- $AS_e \notin SEQ$ (AS_e has not been visited in the current path).
- $AS_e \notin DE$ (AS_e has not already signaled a dead-end to AS_j since the last *pk-request* has arrived to AS_j).

Next-Step ($\mathbf{S_n}$) at an entity $\mathbf{AS_j}$

If B is a descendant of AS_j and AS_j's child towards B is eligible, then return that child.

If AS_j trusts the parent of B, and that parent is eligible, then return that parent.

If AS_j has a cross-link to an ancestor of B, and that ancestor is eligible, then return that ancestor.

If according to AS_j's trust routing table one of AS_j descendants has a cross-link to an ancestor of B or *trusts*$_y$ the parent of B, and AS_j's child towards that descendant is eligible, then return that child.[11]

If AS_j's parent is eligible, then return that parent.

If AS_j has a cross-link to an eligible entity, then return that entity.

If according to AS_j's trust routing table one of AS_j descendants has a cross-link to a potentially eligible entity, and AS_j's child towards that descendant is eligible, then return that child.

Return **null**

4.3. Protocol Analysis

Assume a directed graph in which each node is an entity and there is a directed edge from node P_i to node P_j if there is some trust recommendation expression of the form $P_i \cdot$ *trusts.rec*$^{seq}_{y_\tau}$ (P_j) in P_i's state.

Traversing a directed path in such a graph is defined to correspond to examining, for each entity along that path, its stored trust expressions.

11. Attempts to predict whether the target of the cross-link is likely to be eligible may be incorporated to determine whether such a next step is indeed the best current one.

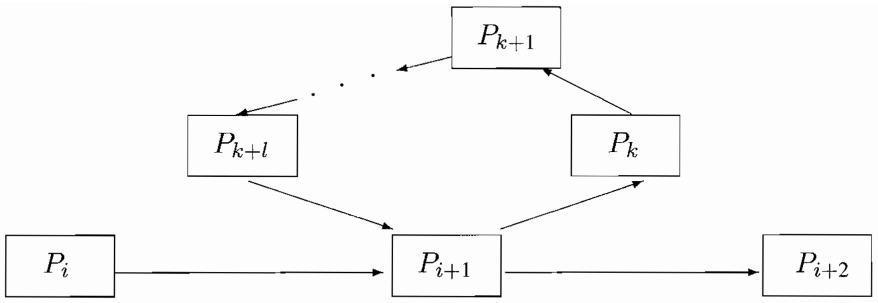


Figure 2. A Cyclic Path.

PROPOSITION 1: In order for P_i to establish all the entities in which it **can** trust, there are cases in which all the directed paths starting from a node P_i need to be traversed.

PROOF: Assume that one such directed path was not traversed, and so that the trust expressions of one node P_e on it (and not on any other path) have not been examined.

The state of P_e may include an expression of the form $P_e \cdot \textit{trusts}^{seq}_{y_\tau}\ P_f$. Given the appropriate trust recommendation expressions starting with P_i's, the above expression could lead (by rule 2) to $P_i \cdot \textit{trusts}^{seq'}_{y_{\tau'}}\ P_f$. Depending on other trust relations in the system, such an expression may otherwise not be obtainable. □

PROPOSITION 2: In order for P_i to establish all the entities in which it **can** trust, it is sufficient to traverse only acyclic paths.

PROOF: Consider an environment with a cyclic path as described in Figure 2.

The directed edges of the graph correspond to expressions of the form:

$P_i \cdot \textit{trusts.rec}_{y_{\tau_{i,i+1}}}\ (P_{i+1})\ when.path[Sp_{i,i+1}]\ when.target[St_{i,i+1}]$,

$P_{i+1} \cdot \textit{trusts.rec}_{y_{\tau_{i+1,i+2}}}\ (P_{i+2})\ when.path[Sp_{i+1,i+2}]\ when.target[St_{i+1,i+2}]$,

$P_{i+1} \cdot \textit{trusts.rec}_{y_{\tau_{i+1,k}}}\ (P_k)\ when.path[Sp_{i+1,k}]\ when.target[St_{i+1,k}]$,

$P_k \cdot \textit{trusts.rec}_{y_{\tau_{k,k+1}}}\ (P_{k+1})\ when.path[Sp_{k,k+1}]\ when.target[St_{k,k+1}]$,

$P_{k+1} \cdot \textit{trusts.rec}_{y_{\tau_{k+1,k+2}}}\ (P_{k+2})\ when.path[Sp_{k+1,k+2}]\ when.target[St_{k+1,k+2}]$,

...

$P_{k+l} \cdot \textit{trusts.rec}_{y_{\tau_{k+1,i+1}}}\ (P_{i+1})\ when.path[Sp'_{k+l,i+1}]\ when.target[St'_{k+l,i+1}]$.

Consider the following paths:

path 1:	$[P_i,\ P_{i+1}]$
path 2:	$[P_i,\ P_{i+1},\ P_k]$
path 3:	$[P_i,\ P_{i+1},\ P_k,\ P_{k+1}]$
$\vdots$	
path $l+2$:	$[P_i,\ P_{i+1},\ P_k,\ P_{k+1}, ...,\ P_{k+l}]$
path $l+3$:	$[P_i,\ P_{i+1},\ P_{i+2}]$
path $l+4$:	$[P_i,\ P_{i+1},\ P_k,\ P_{k+1}, ...,\ P_{k+l},\ P_{i+1},\ P_{i+2}]$

Iterative application of rule 3 would result in new trust recommendation expressions that may be obtained by P_i. In each application of rule 3, the new constraints sets are intersected with the current ones. Thus, every trust recommendation expression obtained via a cyclic path will never be less constrained than one that is obtained via some acyclic path.

In particular, after traversing path $l+3$ P_i may obtain (using rule 3) an expression of the form:

$$P_i \cdot \textit{trusts.rec}_{y_{\tau_{i,i+1} \cap \tau_{i+1,i+2}}}(P_{i+2}) \quad when.path[\mathcal{S}p_{i,i+1} \cap \mathcal{S}p_{i+1,i+2}]$$
$$when.target[\mathcal{S}t_{i,i+1} \cap \mathcal{S}t_{i+1,i+2}]$$

while after traversing path $l+4$ it may obtain only

$$P_i \cdot \textit{trusts.rec}_{y_{\tau_{i,i+1} \cap \tau_{i+1,k} \cap \tau_{k,k+1} \cap ... \cap \tau_{k+l,i+1} \cap \tau_{i+1,i+2}}}(P_{i+2})$$
$$when.path[\mathcal{S}p_{i,i+1} \cap \mathcal{S}p_{i+1,k} \cap \mathcal{S}p_{k,k+1} \cap ... \cap \mathcal{S}p_{k+l,i+1} \cap \mathcal{S}p_{i+1,i+2}]$$
$$when.target[\mathcal{S}t_{i,i+1} \cap \mathcal{S}t_{i+1,k} \cap \mathcal{S}t_{k,k+1} \cap ... \cap \mathcal{S}t_{k+l,i+1} \cap \mathcal{S}t_{i+1,i+2}]$$

Thus, path $l+4$ is redundant. □

PROPOSITION 3: The protocol may traverse all such acyclic paths starting at node A.

PROOF:

(a) The protocol traverses only acyclic paths.

Path traversal corresponds in the protocol to the sending of *pk-request* messages. Each *pk-request* message contains the sequence of entities along a trust path which were visited thus far. At each node, the identity of the next entity to which a *pk-request* message will be forwarded is determined by the Next-Step algorithm. That algorithm only chooses nodes which are *eligible*. One of the conditions defined for *eligibility* is that the next node is not already a member of the current path.

(b) The protocol may traverse *all* such paths.

At each node for each *pk-request* message, the Next-Step algorithm may choose any of the nodes in which that node trusts, as intended recipients of the next *pk-request* message. It will eventually choose each such possible entity, unless it receives a *pk-response* message from one of the next nodes to which it sent a *pk-request* message. □

PROPOSITION 4: The protocol execution terminates and is exponential in the number of nodes.

PROOF: There is a finite number of nodes. As established in proposition 3, each node may forward each *pk-request* message it receives to each of the entities it trusts. Therefore the complexity of the algorithm is exponential in the number of nodes.

The protocol ends when A receives either a *pk-response* message or dead-end messages from all the nodes which, according to its initial trust expressions, it trusts. Only acyclic paths are traversed (proposition 3) and each, of course, is of finite length. An acyclic path is never traversed twice—at each node the Next-Step algorithm will never select a node that has already returned a dead-end message for the current path (as determined by the S_n record). Therefore the protocol will terminate.[12] □

Note that traversing all acyclic paths is sufficient (proposition 2), that the protocol traverses all such acyclic paths (proposition 3) and that at each node a dead-end message is sent back only if the node cannot at all find an appropriate new node consistently with rules 2–5. Consequently if A receives dead-end messages from all the nodes it initially trusts then it can conclude that there is no trusted path via which it is possible for it to obtain B's public key.

Finally, the last proposition establishes that if the protocol terminates successfully—A receives a *pk-response* message, then A possesses a value that it can believe is B's public key.

PROPOSITION 5: If A receives a *pk-response* message then $A \ni K_{B+}$ and $A \models \overrightarrow{K_{B+}}$.

PROOF: Presented in Appendix A.

12. The termination analysis is based on the assumption that every message will eventually be received by its intended recipient. In practice, due to site failure or communication problems, *time-outs* may be triggered and used to abort particular path executions, as required.

4.4. Discussion

Although the general complexity of the protocol presented is exponential, optimizations that rely on assumptions about the fact that *most* trust relations correspond to the hierarchical structure may make the execution performance relatively efficient.

The advantage of the protocol presented is that it can execute in an environment in which there are arbitrary trust relations between entities. That advantage is, of course, associated with a cost. There is a state that needs to be maintained at some of the intermediate entities: trust routing tables and, during a protocol's execution, S_n records. The messages exchanged between entities may be longer than in the case of protocols that do not consider trust-based information. Similarly, there may be more such messages: some of the execution paths traversed may not necessarily be the shortest ones.

However, although a closer investigation of the performance implications is an open challenge, it seems that such costs are proportional to the level of deviations from the default trust-relationships. In other words, in environments in which most trust-relations are consistent with some global default (e.g., classical hierarchical relations) then the cost of dealing with the exceptional relations may not be excessive.

The protocol may be fine-tuned for specific circumstances. For example, if there is a high probability that the path traversed in order to obtain B's public key for A may also be acceptable for B with respect to A's public key, then that public key, and its related recommendations for B, can also be piggybacked on the exchange we described. Also, execution delays can be traded-off against numbers of messages, by adopting a multicast approach to a group of trusted entities, rather than the depth-first type approach we have presented.

5. An Example

In this section we describe an execution of the protocol in a particular setting.

Assume A, B and C' to be scientists in institutions in Jerusalem, Karlsruhe, and Bagdad, respectively. Other entities (see Figure 3) are servers administered by various authorities such as university departmental authorities (e.g., AS_A at the Hebrew University in Jerusalem), inter-governmental authorities (e.g., AS_u administered by a European Community authority), or global authorities (e.g., AS_r administered by some United Nations authority).

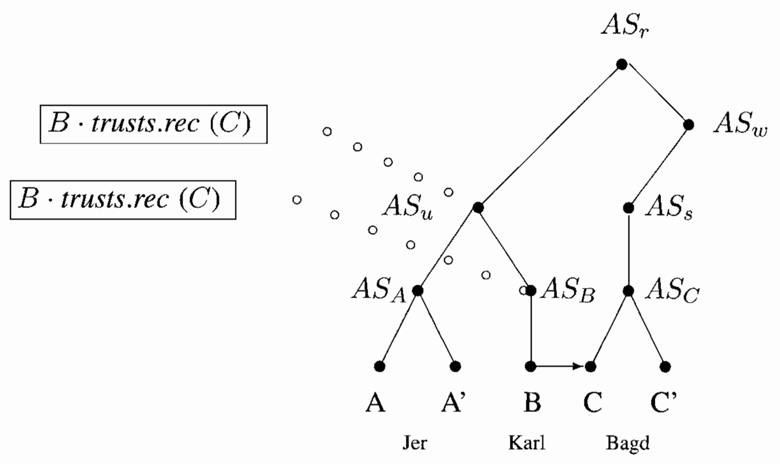


Figure 3. An Example Environment.

In general, entities trust their ancestors and may be willing to accept their recommendations regarding the trustworthiness of other entities, but with certain constraints. For example, A trusts his ancestors' recommendations about other entities (who are not necessarily related ancestorially) and even accept these recommended entities' recommendations. However, A regards further such cross recommendations from recommended entities as too remote. In addition, regardless of any recommendations, A is not willing to trust any servers that are administered directly by regimes with which Israel does not have diplomatic relations.

A's ancestors also trust their own ancestors' recommendations, but only consider a single cross recommendation as acceptable—reflecting a tighter trust policy.

Finally, B in Karlsruhe was a student with C who is now in Bagdad, and trusts it and its recommendations.

A wishes to collaborate with B and C' (and thus needs to communicate securely with them) but, given the current political climate, is not willing to trust servers administered directly by the Iraqi regime (such as AS_w). In contrast, A may be willing to trust, in some respects, certain other entities in Iraq. It may trust a server (e.g., AS_C) administered by a university department in Bagdad with respect to keeping clocks synchronized, following protocol specifications correctly, and identifying entities in its domain τ_2 (by an entity's domain we mean here the entity's children or the children of its parent).

The above scenario corresponds to three expressions of the following form:

$$AS_A \cdot \textit{trusts.rec}_{y_{\tau_1}} \; (AS_u) \; \textit{when.path}[\mathcal{S}p_1] \; \textit{when.target}[\mathcal{S}t_1]$$

where $\mathcal{S}p_1$ and $\mathcal{S}t_1$ represent the path and target constraints, respectively, which allow following the hierarchy and at most one cross-link,[13] and τ_1 represents the set of entities which may be identified—in this case: all entities in the recommended entity's domain.

$$B \cdot \textit{trusts.rec}_{y_{\tau_2}} \; (C) \; \textit{when.path}[\mathcal{S}p_2] \quad \textit{when.target}[\mathcal{S}t_2]$$

where $\mathcal{S}p_2$ and $\mathcal{S}t_2$ represent the path and target constraints, respectively, that allow only following the hierarchy, and τ_2 represents the set of entities in AS_C's domain.

$$A \cdot \textit{trusts.rec}_{y_{\tau_3}} \; (AS_A) \; \textit{when.path}[\mathcal{S}p_3] \; \textit{when.target}[\mathcal{S}t_3]$$

where $\mathcal{S}p_3$ and $\mathcal{S}t_3$ represent the path and target constraints, respectively, which allow following the hierarchy and at most two cross-links, but without entities such as AS_w, and τ_3 represents the set of entities which may be identified—in this case: all entities in the recommended entity's domain.

The following protocol execution may be initiated by A:

$A \rightarrow AS_A$: $\{\{S_n, ts, [A, AS_A], K_{C'+}?, \mathcal{S}p_3, \mathcal{S}t_3, \tau_3\}_{K_{A-}}\}_{K_{AS_A+}}$

AS_A Stores $S_n, ts, [A, AS_A], K_{C'+}?, \mathcal{S}p_3, \mathcal{S}t_3, \tau_3$

Performs Next-Step(S_n)

Next-Step(S_n) is AS_u (the parent of AS_A)

$\mathcal{S}p_4 = \mathcal{S}p_3 \cap \mathcal{S}p_1$

$\mathcal{S}t_4 = \mathcal{S}t_3 \cap \mathcal{S}t_1$

$\tau_4 = \tau_3 \cap \tau_1$

where $\mathcal{S}p_4$ and $\mathcal{S}t_4$ represent the path and target constraints, respectively, which allow to follow the hierarchy and at most one cross-link, but not to consider AS_w, and τ_4 represents the set of entities which may be identified.

13. The specification of such constraints is demonstrated in (Yahalom et al. 1993). We omit here the required formalism for the sake of brevity. Essentially, an expression can refer to path information and use predicates such as *parent-of* to restrict, for example, the number of up-to-down hierarchy transitions or the number of cross-links.

$AS_A \rightarrow AS_u$: $\{\{S_n, ts, [A, AS_A, AS_u], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4\}_{K_{AS_A-}}\}_{K_{AS_u+}}$

AS_u Stores $S_n, ts, [A, AS_A, AS_u], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4$

Performs Next-Step(S_n)

Next-Step(S_n) is AS_r (because there is no cross-link to an ancestor of C')

$AS_u \rightarrow AS_r$: $\{\{S_n, ts, [A, AS_A, AS_u, AS_r], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4\}_{K_{AS_A-}}\}_{K_{AS_u+}}$

AS_r Stores $S_n, ts, [A, AS_A, AS_u, AS_r], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4$

Performs Next-Step(S_n)

Next-Step(S_n) is null

Deletes S_n

$AS_r \rightarrow AS_u$: $\{\{S_n, ts, AS_r, AS_u, deadend\}_{K_{AS_r-}}\}_{K_{AS_u+}}$

AS_u $DE := DE \cup \{AS_r\}$

Performs Next-Step(S_n)

Next-Step(S_n) is AS_B (because of the cross-link at B)

$AS_u \rightarrow AS_B$:

$\{\{S_n, ts, [A, AS_A, AS_u, AS_B], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4\}_{K_{AS_u-}}\}_{K_{AS_B+}}$

$AS_B \rightarrow B$:

$\{\{S_n, ts, [A, AS_A, AS_u, AS_B, B], K_{C'+}?, \mathcal{S}p_4, \mathcal{S}t_4, \tau_4\}_{K_{AS_B-}}\}_{K_{B+}}$

$\vdots$

$\mathcal{S}p_5 = \mathcal{S}p_4 \cap \mathcal{S}p_2$

$\vdots$

$B \rightarrow C$:

$\{\{S_n, ts, [A, AS_A, AS_u, AS_B, B, C], K_{C'+}?, \mathcal{S}p_5, \mathcal{S}t_5, \tau_5\}_{K_{B-}}\}_{K_{C+}}$

$C \rightarrow AS_C$:

$\{\{S_n, ts, [A, AS_A, AS_u, AS_B, B, C, AS_C], K_{C'+}?, \mathcal{S}p_5, \mathcal{S}t_5, \tau_5\}_{K_{C-}}\}_{K_{AS_C+}}$

$AS_C \rightarrow C$:

$\{\{S_n, ts, [A, AS_A, AS_u, AS_B, B, C, AS_C, C], C', K_{C'+}\}_{K_{AS_C-}}\}_{K_{C+}}$

$C \rightarrow B$:

$$\{\{S_n, ts, [A, AS_A, AS_u, AS_B, B, C, AS_C, C, B], C', K_{C'+}\}_{K_{C-}}\}_{K_{B+}}$$

$$\vdots$$

$AS_A \rightarrow A$:

$$\{\{S_n, ts, [A, AS_A, \ldots, B, C, AS_C, C, \ldots, AS_A, A], C', K_{C'+}\}_{K_{AS_A-}}\}_{K_{A+}}$$

$$A \rightarrow C' : \{A, C', ts, message\}_{K_{C'+}}$$

6. *Related Work*

We discuss some previous approaches to the problem of initiating a secure session between entities in a multidomain environment and outline their relation to the approach presented in this article.

A recent project, titled Privacy Enhanced Mail (PEM), aims to enhance electronic mail systems to achieve security and privacy (Kent 1993, Linn 1993). The PEM work assumes that the entities are organized in a global name space hierarchy. When an entity wishes to obtain the public key of another, the hierarchy is traversed from the root of the hierarchy downwards towards the other entity. Such an approach implicitly assumes that each such entity trusts (e.g., with respect to identification) the root, as well as ancestors of the target entity. Such an approach is a special case of the one we presented here, in which there are no *exceptional trust relations*.

The work of Birrell et al. (Birrell et al. 1986) and more recently of Lampson et al. (Lampson et al. 1991) relaxes some of the trust-related requirements of PEM-like approaches. In particular, in their work entities may set up a secure session between themselves if they only trust the sub-hierarchy to which they both belong. Entities are not necessarily required to trust other parts of the hierarchy, such as the root. To obtain the public key of another entity, the subhierarchy is traversed upwards from one entity to the least common ancestor of it and the target entity and then downwards towards the target entity. In addition, entities may make use of a single cross-link within their trusted sub-hierarchy, as a shortcut between the upward and downward traversal of the hierarchy. In (Lampson et al. 1991) it is established that different hierarchical trust relations may be required by each of a session's participants, even with respect to the same set of intermediate entities. Again, these models correspond to a special case of our approach in which there are some particular types of *exceptional* trust relations, which are globally assumed. Namely, some entities may be considered *not trusted.*

As our example demonstrated, there are circumstances in which more flexible exceptional trust relations must be supported.

Gligor et al. further extend the type of trust structures that may be supported (Gligor et al. 1992). In particular, in their approach a single cross-link may be incorporated, thus linking two different sub-hierarchies. Such a scenario corresponds in our approach to a form of a single-step exceptional trust recommendation. It falls short of providing support for more elaborate recommendation types, and thus is not sufficient to deal with situations such as the one we presented in our example in section 5.

The problem of deriving new trust relations between entities is related to that of determining access rights in access control models (e.g., (Harrison et al. 1976, Rabin & Tygar 1987)).

In (Harrison et al. 1976), Harrison, Ruzzo, and Ullman examine the problem of determining whether, for given circumstances, a subject (using their terminology) may acquire a particular type of right to an object. One case of their analysis corresponds to our problem of deriving new trust relations (Yahalom et al. 1993). For that case they establish that there exists an algorithm of exponential complexity.

Rabin and Tygar present in (Rabin & Tygar 1987) a formalism that allows the specification of sets as privilege and protection components. Their algorithm, which determines whether a privilege component satisfies a protection component, is related to the path constraints consistency checks (rules 2–5) performed in our protocol.

Finally, consider the SPX protocol developed at Digital Equipment Corp. (Tardo & Alagappan 1991). The environment assumed by the SPX designers consists of a global hierarchical trust structure with particular *exceptional* relations. SPX assumes that each public key repository stores the public keys of a large number of entities. Thus, if A wishes to obtain the public key of B, the path that it is required to traverse may be short. Furthermore, pre-evaluations of cross-links and the 'up'-part of paths may be stored to speed up executions. The SPX approach may also be viewed as a special case of our approach. However the structure of the consequent message from A to B can, in our opinion, be considered as a flaw in the SPX design. That message structure can be represented as:

$$A \rightarrow B : \{A, ts, msg\}_{K_{B+}}$$

Where A is the name of entity A, ts is a timestamp, and msg may, for example, contain a session key generated by A to be shared with B.

In particular, B's identity is not specified explicitly within the message. That implies, that if B is to be convinced that the sender of the message indeed intended it to be targeted at B, B needs to trust (with respect to identification of B)

the path A used in order to obtain K_{B+} and its binding to B. Otherwise, as far as B is concerned, A may have believed that K_{B+} is some C's public key. However, if there are multiple such paths, or if in some domain the public key repository is replicated, B may not necessarily be able to infer which path was used by A. A solution to this problem is simply to require A to insert B's name into the message, as we do in the presented protocol.

7. Conclusions

Secure protocols need to reflect particular views of the participating entities about different types of trustworthiness of other entities. In this article we introduced an approach for establishing how trust expressions may be acquired during protocol executions. Such an approach enables the design of protocols for environments in which not all entities are assumed to conform to some pre-determined trust structure. The potential additional cost associated with such protocols may be traded off against the amount of additional trust-related flexibility required.

There are various directions in which this research can be further pursued. The performance/flexibility trade-off needs to be evaluated in particular settings and within the context of implementation prototypes. Other protocols need to be designed and some of the assumptions we made in this article may need to be relaxed. For example, an entity may initially trust the recommendations of another but not be able to directly interact with it because it does not possess the required cryptographic keys. Issues associated with the evolution of trust relations over time need also be considered. An entity's trust in another (in some respect) may change due to various events that may take place.

Acknowledgments
We thank Malte Borcherding, Marcus Otten, Peer Wichmann, and the three anonymous referees for helpful comments. The first author would like to thank Li Gong, Bill Harbison, Mark Lomas, Roger Needham, David Wheeler, and other members of the security group at Cambridge for numerous discussions over the last few years that inspired him to examine trust issues in distributed systems.

References

1. M. Abadi, M. R. Tuttle, A Semantics for a Logic of Authentication, *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing 1991*, pages 201–216.

2. A. Birrell, B. Lampson, R. Needham, M. Schroeder, A Global Authentication Service Without Global Trust, *Proceedings of the IEEE Conference on Security and Privacy 1986*, pages 223–230.
3. M. Burrows, M. Abadi, R. Needham, A Logic of Authentication, *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, Litchfield Park, Arizona, December 1989. Published as *ACM Operating Systems Review*, Vol. 23, No. 5 (1989). A fuller version was published as DEC System Research Center Report No. 39, Palo Alto, California, February 1989.
4. International Telegraph and Telephone Consultative Committee (CCITT), *The Directory: Overview of Concepts, Models, and Services, Recommendation X.500*, 1988.
5. International Telegraph and Telephone Consultative Committee (CCITT), X.509, *The Directory—Authentication Framework, IX^{TH}*. Plenary Assembly, Melbourne, 14–25 November 1988.
6. T. ElGamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Trans. on Information Theory* 31, 1985, pages 469–472.
7. V. D. Gligor, S.-W. Luan, J. N. Pato, On Inter-realm Authentication in Large Distributed Systems, *Proceedings of the IEEE Conference on Security and Privacy 1992*, pages 2–17.
8. L. Gong, R. Needham, R. Yahalom, Reasoning about Belief in Cryptographic Protocols, *Proceedings of the IEEE Conference on Security and Privacy 1990*, pages 234–248.
9. M. Harrison, W. Ruzzo, J. Ullman, Protection in Operating Systems, *Communications of the ACM*, Vol. 19, No. 8, August 1976, pages 461–471.
10. S. Kent, Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management, Proposed Standard Protocol RFC 1422, Digital Equipment Corporation, February 1993.
11. B. Lampson, M. Abadi, M. Burrows, E. Wobber, Authentication in Distributed Systems: Theory and Practice, *The 13th ACM Symposium on Operating Systems Principles*, October 1991.
12. J. Linn, Privacy Enhancement for Internet Electronic Mail, Part I: Message Encryption and Authentication Procedures, Proposed Standard Protocol RFC 1421, Digital Equipment Corporation, February 1993.
13. M. O. Rabin, J. D. Tygar, *An Integrated Toolkit for Operating System Security*, Harvard Technical Report, TR-05-87, August 1988.
14. R. L. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Comm. of the ACM*, Vol. 21, No. 2, February 1978, pages 120–126.
15. J.J. Tardo, K. Alagappan, SPX: Global Authentication Using Public Key Certificates, *Proceedings of the IEEE Conference on Research in Security and Privacy 1991*.
16. R. Yahalom, B. Klein, Th. Beth, Trust Relationships in Secure Systems—A Distributed Authentication Perspective, *Proceedings of the IEEE Conference on Research in Security and Privacy* 1993.

Appendix A: Proof of Proposition 5

PROPOSITION 5: If A receives a *pk-response* message then $A \ni K_{B+}$ and $A \models \overrightarrow{K_{B+}}$.

PROOF:

(a) If A receives a *pk-response* message then there exists a path of entities $[A, P_1, ...P_m]$ such that P_m possesses a value K_{B+} it believes is B's public key. Each entity on that path (except for A) received a *pk-request* message from its predecessor, and each entity on the path (except for P_m) received a *pk-response* which contains K_{B+} from its successor:

According to the protocol specifications, the first time a *pk-response* message is sent is when an entity possesses an item it believes to be B's public key. It sends it to the entity from whom it received a *pk-request* message. From that point each entity generates a *pk-response* message only if it receives a *pk-response* message and always forwards it (with B's public key) to the last entity that sent it the *pk-request* message. Because each path of *pk-request* messages is acyclic and starts from A, the *pk-response* message will eventually be forwarded to A.

(b) Each entity P_i on the above path (except for P_m) may believe when it receives the *pk-response* message that P_{i+1} possesses $\mathcal{S}p$ and $\mathcal{S}t$ and regards them as path and target constraints, and possesses τ and regards it as the set of entities which may be identified:

Each *pk-request* message has the following form

$$\{\{S_n, ts, [A, P_1, \ldots, P_i, seq, P_{i+1}], K_{B+}?, \mathcal{S}p'', \mathcal{S}t'', \tau''\}_{K_{P_i-}}\}_{K_{P_{i+1}+}}$$

The last three fields are interpreted respectively as *the current path and target constraint sets* and *the current set of entities which may be identified.* The message also explicitly includes the path traversed thus far and the identifiers P_i and P_{i+1}.

P_i trusts P_{i+1} with respect to following protocol steps and to maintaining synchronized clocks (because only such entities are chosen as a Next-Step). Thus P_i believes that P_{i+1} will only consider recent messages as valid, and consider the last three fields as defined above. Consequently P_i believes that if P_{i+1} sent it a message implying *I received a recent message from you* that indeed it received the *pk-request* message which P_i sent.

The *pk-response* is associated with an implicit interpretation: *I received your recent pk-request message (identified by the S_n field) and acted consistently with it.* P_i's possession of an item it believes is P_{i+1}'s public key (assumption about trust relations) allows it to conclude that P_{i+1} made that statement. Its trust expressions allow it to conclude that it made the statement recently (in the current execution), that it received its message, and acted according to the protocol specifications. Consequently, P_{i+1} received $\mathcal{S}p$ and $\mathcal{S}t$ and regarded them as path and target constraints, and received τ and regarded it as the set of entities which may be identified.

(c) Each entity in the above path (except for P_m) which receives a *pk-response* message may possess K_{B+} and believe it is B's public key:

We prove this claim by induction, considering paths of length n. With each *pk-response* message there is an associated implicit statement by the sender P_{i+1} to the intended recipient P_i: *I believe that you are allowed to trust P_m with respect to B's public key and K_{B+} is the value it provided* (the values P_m, B, K_{B+} are explicitly included, among others, in the message).

The case $n = 1$ is trivial—A receives a *pk-response* from P_m (which it trusts—or else it would not have communicated with it) and applies rule 1 when it receives the response to deduce the validity of the key.

However, because our general proof assumes paths of at least length 2, we need to establish that the claim holds for $n = 2$. P_1 received a *pk-response* from P_m. P_1 trusts P_m and also believes that it is consistent with A's constraints (which it received in the request message). Consequently it may convey the *pk-response* message to A with the above meaning. A can apply rule 4 for which all the conditions hold: It trusts the recommendations of P_1—or it would not have sent a request to it. It believes P_1 possesses $\mathcal{S}p$ and $\mathcal{S}t$ and regards them as path and target constraints and similarly τ (claim (b)). It received a message encrypted with P_1's private key implying that P_1 states that A should believe in the trustworthiness of P_m. Applying rule 4, A may acquire a trust expression in P_m (and its ability to identify entities in a set which contains B). As P_1 testified that $\overrightarrow{K_{B+}}$, and A trusts P_1, A may, using a variant of rule 1, believe that K_{B+} is B's public key:

$$A \cdot \textit{trusts.rec}^{seq}_{y_\tau}\,(P_1)\ \textit{when.path}[\mathcal{S}p]\ \textit{when.target}[\mathcal{S}t]$$

$$A \models P_1 \ni (\mathcal{S}p, \mathcal{S}t, \tau)$$

$$A \lhd \{ts,\ \text{A is allowed to } \cdot \textit{trust}_{y_{\tau'}} P_m, seq'\}_{K_{P_1}}$$

$$\overset{rule\ 4}{\Longrightarrow}\ A \cdot \textit{trusts}^{seq'}_{y_{\tau'}}\,(P_m) \Longrightarrow A \models \overrightarrow{K_{B+}}$$

We now assume that the claim holds for a sequence of *pk-response* messages (corresponding to a path starting from P_m) of length k. That is, assume that after

P_{m-k} receives its *pk-response* message it may trust P_m with respect to the provision of B's public key, and believe that P_m claimed that K_{B+} is B's public key:

$$P_{m-k} \cdot \textit{trusts}^{seq}_{y_\tau}(P_m) \text{ and } P_{m-k} \models \overrightarrow{K_{B+}}$$

We now prove that the above assumption implies that the same holds for P_{m-k-1}. P_{m-k-1} may believe that P_{m-k} possesses the appropriate path constraints and believes them to be these appropriate constraints (proved in claim (b)). P_{m-k-1} trusts the recommendations of P_{m-k} - or else it would not have chosen it as a next entity for a *pk-request* message. In particular it trusts P_{m-k}, while following the protocol steps, to verify that the path traversed (between P_{m-k} and P_m) is consistent with the above appropriate constraint sets. According to the induction assumption, P_{m-k} gains a trust in P_m (and consequently believes that it possesses B's key), and can send a *pk-response* message, with its implied meaning, to P_{m-k-1}. Thus, upon receiving that message, by using rule 4, P_{m-k-1} acquires a trust expression in P_m. That trust together with the belief that P_m stated that K_{B+} is B's public key, leads P_{m-k-1} to believe that indeed it is.

$$P_{m-k-1} \cdot \textit{trusts.rec}^{seq}_{y_\tau}(P_{m-k})\ \textit{when.path}[\mathcal{S}p]\ \textit{when.target}[\mathcal{S}t]$$

$$P_{m-k-1} \models P_{m-k} \ni (\mathcal{S}p, \mathcal{S}t, \tau)$$

$$P_{m-k-1} \lhd \{ts,\ P_{m-k-1} \text{ is allowed to } \cdot \textit{trust}_{y_{\tau'}} P_m, \textit{seq}'\}K_{P_{m-k}}$$

$$\overset{rule\ 4}{\Longrightarrow}\ P_{m-k-1} \cdot \textit{trusts}^{seq'}_{y_{\tau'}}(P_m) \implies P_{m-k-1} \models \overrightarrow{K_{B+}}$$

□