Whisper: A Local Secret Maintenance Protocol

Vinayak S. Naik, Anish Arora, Sandip S. Bapat, and Mohamed G. Gouda

***Abstract*—A challenge in resource-constrained networks is to provide secure communication in an efficient manner. In this paper, we present a simple protocol for secret maintenance between a pair of network neighbors. We prove that Dolev Yao model based adversaries cannot compromise the current secret shared by the neighbors. Moreover, we show that if the current secret between the pair is somehow disclosed, previous secrets are not compromised nor can future secrets be compromised.**

***Index Terms*—Computer network security, Network fault tolerance, Network reliability, Protocols.**

I. INTRODUCTION

A basic step in providing secure communication in a network despite the activity of intruders is to empower authentic network entities with secrets. In this paper we address the problem of maintaining secrets in resource-constrained networks, e.g. sensor networks.

Desired properties of secret maintenance in resource-constrained networks are forward secrecy, backward secrecy, scalability, tolerance to loss of synchronization, and tolerance to state corruption of the entities. Forward secrecy means that compromise of the current session key[1] does not imply compromise of future session keys. Backward secrecy means that compromise of the current session key does not imply compromise of past session keys. The issue of scalability is of primary concern as network of small devices offer the opportunity to deploy a large number of low powered devices as opposed to a small number of high powered devices. Lack of manual configuration and the high frequency of faults motivate the need for tolerances.

Manuscript received January 27, 2003. This work was partially sponsored by DARPA NEST contract OSU-RF program F33615-01-C-1901, NSF grant CCR-9972368, an Ameritech Faculty Fellowship, and two grants from Microsoft Research. This paper has been cleared through author affiliations

V. S. Naik is with the Computer and Information Science Department, Ohio State University, Columbus, OH 43210 USA (phone: 614-688-4650; e-mail: naik@ cis.ohio-state.edu).

A. Arora is with the Computer and Information Science Department, Ohio State University, Columbus, OH 43210 USA (e-mail: arora@ cis.ohio-state.edu).

S. S. Bapat is with the Computer and Information Science Department, Ohio State University, Columbus, OH 43210 USA (e-mail: bapat@ cis.ohio-state.edu).

M. G. Gouda is with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712 USA (e-mail: gouda@cs.utexas.edu).

[1] Key and secret are used interchangeably in this paper.

Extant protocols such as Diffie-Hellman key agreement [3] and ones using asymmetric cryptography that deal with secret establishment and exchange involve operations such as exponentiation and multiplication, which consume a lot of computational power, memory and energy. For example, on Palm V device 512-bit modulo exponentiation takes 96.91 seconds [18]. Hence these solutions are not suitable for nodes that have limited resources. An exception is SPINS [16], which as discussed in Section VI, does not deal with secret maintenance. The contribution of this paper is to present "*Whisper*", a protocol with the above mentioned properties.

Overview of Whisper *Whisper* limits its use of cryptographic constructs to one–way (pre-image resistant) hash functions, which can be computed efficiently. The SHA-1 hash function gives a throughput of 17,429 Bps for a message of size 2KB on Palm V [18]. The protocol is thus suitable for execution on resource constrained devices. To provide forward secrecy, we derive each session key from two distinct "key–parts" known only to the two neighboring principals sharing the session key. To move to their next session, they chose their new key–parts, encrypt these new key–parts using their previous key–parts, and exchange the encrypted new key–parts in a predetermined order. To assure that compromise of the current key does not reveal the current key–parts, the function selected to compute the key from the two key–parts is one–way. (This approach may be contrasted to extant solutions that use long term keys to update session keys, under the assumption that long term keys are secure.)

More precisely, secret maintenance in *Whisper* proceeds as follows. Let A and B be two neighboring principals that share a key. Principal A's key–parts are stored in an array X_A and those of principal B in Y_B. A and B know the two key–parts $X_A[i-1]$ and $Y_B[i-1]$ of the key $C_{AB}[i-1]$ at the end of $(i-1)^{th}$ session of the protocol. To update the key, A sends a request to B which contains one key–part ($X_A[i]$) for the new secret, obscured with $X_A[i-1]$ and authenticated using $C_{AB}[i-1]$. B can retrieve $X_A[i]$ as it knows $X_A[i-1]$ and it can verify the authenticity of the message since it knows $C_{AB}[i-1]$. B then responds by contributing the second key–part ($Y_B[i]$) for the new secret, obscured with $Y_B[i-1]$ and authenticated using the new secret ($C_{AB}[i]$) which it computes by using $X_A[i]$ and $Y_B[i]$. After receiving B's reply, A can retrieve $Y_B[i]$ as it knows $Y_B[i-1]$ and it can compute $C_{AB}[i]$, which it also uses to verify

the authenticity of the message.

We summarize the protocol below. h is a one-way hash function. f is also a one-way function, which enables forward secrecy. Function *rand* returns a positive random integer.

$$A \rightarrow B: \quad X_A[i] + h(X_A[i\text{-}1], B),\ h(C_{AB}[i\text{-}1], X_A[i])$$
$$B \rightarrow A: \quad Y_B[i] + h(Y_B[i\text{-}1], A),\ h(C_{AB}[i], Y_B[i])$$

where $X_A[i] = rand(\ )$;
$Y_B[i] = rand(\ )$;
$C_{AB}[i] = f(X_A[i], Y_B[i])$

For reason of space, we assume that the initial secrets are bootstrapped before hand. We discuss one way of bootstrapping the secrets in [13].

Organization of the paper In Section II, we describe the network, intruder and fault model that we consider in this paper. In Section III, we give a brief introduction to the Abstract Protocol Notation (APN) [6] and recall definitions of security concepts. We formalize *Whisper* in APN in Section IV. In Section V, we mention security and fault-tolerance properties of *Whisper*. In Section VI, we discuss related work and make concluding remarks.

II. System Model

A. Network Model

The network consists of nodes that are small devices which communicate with each other and with a more powerful base station. In turn, the base station may be connected to an outside network. By design, nodes are inexpensive and have limited computational and communicational resources. Communication is radio based and is an energy-consuming function for these nodes. Each principal has a unique ID and shares a secret with the base station. For simplicity, we assume that each non-malicious node represents a unique principal.

We model the broadcast radio-based communication by two channels between two principals: one is from *A* to *B*, and the other one is from *B* to *A*. Each message sent from *A* to *B* remains in the channel from *A* to *B* until it is eventually received by *B* or is lost. Messages that reside simultaneously in a channel form a set and so they are received or lost, one at a time, in any order and not necessarily in the same order in which they are sent.

B. Intruder Model

The intruder model assumed here is the one proposed by Dolev and Yao [4]. Informally, all the communication channels are accessible to an intruder for reading and writing. The intruder can also intercept the messages, store them in encrypted and decrypted form (in case it knows the keys), and construct messages using the stored and known values. An intruder can be a malicious node in the network and hence can engage in sessions with other neighboring nodes.

The primitive data types that may occur in any message are IDs of the processes and keys. Compound fields are constructed by concatenation and hashing. The concatenation of fields X and Y is the field (X,Y). The hash of a field X is h(X). The sets of primitive data types and compound field are disjoint. Formally, the fundamental operations on a set S of message fields that are possible for an intruder are *parts*(S), *analz*(S) and *synth*(S) as defined by Paulson [14]. Briefly, *parts*(S) is the set of all the subfields of fields in the set S, including components of concatenations and the plaintext of encryptions (but not the secret keys). *analz*(S) is the subset of *parts*(S) consisting of only those subfields that are accessible to an intruder. These include components of concatenations and the plaintext of those encryptions where the secret key is in *analz*(S). Finally, *synth*(S) is the set of fields constructible from S by concatenation and encryption using fields and keys in S. We use the following two results from [14]:

- The set transformers *parts*(S), *analz*(S), and *synth*(S) are closure operators
- The *fake*(S) operator models an intruder

$$fake(S) = synth(analz(S))$$

Let the actions of an intruder be called F_W. We do not consider the jamming of radio signals in this paper. Modeling an intruder as a set of actions does not reduce the capabilities of an intruder. Note that, F_W includes complex operations such as *parts*(S), *analz*(S), and *synth*(S).

C. Fault Model

In addition to the faults captured by the intruder model above, there are corruption faults that corrupt the values of variables in the volatile memory of a node. These faults result in garbage values in the corrupted variables.

III. Programming Notation and Concepts

A. Abstract Protocol Notation Syntax

In this section, we briefly recall APN. In this notation, each process in a protocol is defined by a set of constants, ał set of variables and a set of actions. Let A be a process in a protocol. The variables of process A can be read and updated by the actions of process A. Each ⟨action⟩ has a unique name and is of the form:

⟨name⟩ :: ⟨guard⟩ → ⟨statement⟩

The guard of an action of A has one of the following three forms: a boolean expression over the constants and variables of A, a receive guard of the form **rcv** ⟨message⟩ **from** B where B is another process, or a timeout guard that

contains a boolean expression over the constants and variables of every process and the contents of the channels in the protocol.

Executing an action consists of executing all the statements of this action atomically. Executing the actions of different processes in a protocol proceeds according to the following three rules. First, an action is executed only when its guard is true. Second, the actions in a protocol are executed one at a time. Third, an action whose guard is continuously true is executed eventually.

The ⟨statement⟩ of an action of process *A* is a sequence of ⟨skip⟩, ⟨assignment⟩, ⟨send⟩, ⟨receive⟩ or ⟨selection⟩ statements of the following forms:

⟨skip⟩ : skip
⟨assignment⟩ : ⟨variable in *A*⟩ := ⟨expression⟩
⟨send⟩ : **send** ⟨message⟩ **to** *B*
⟨receive⟩ : **rcv** ⟨message⟩ **from** *B*
⟨selection⟩ : **if** ⟨boolean expression⟩ → ⟨statement⟩
⟨boolean expression⟩ → ⟨statement⟩
fi

The ⟨send⟩ statement puts a message in the channel between *A* and *B*. The ⟨receive⟩ statement removes a message from the channel between *A* and *B*. Note that, the receive statement does not authenticate the sender of the message.

B. Semantics

Let *p* be a protocol. A *state* of *p* is defined by a value for each variable of *p*, chosen from the predefined domain of the variable. A *state predicate* of *p* is a boolean expression over the variables of *p*. An action of *p* is enabled in a state iff its guard (state predicate) evaluates to true in that state.

Let *s* and *s′* be the two states of *p*. (*s*,*s′*) is called a *state transition* of *p* iff there exists an action ⟨guard⟩ → ⟨statement⟩ such that (*s* ∧ guard) holds and after executing statement *s′* holds. $<s_1,s_2,s_3,\ldots,s_{n-1},s_n>$ is called a *state sequence* of a protocol *p* iff $\forall i{:}1 \le i \le n\text{-}1$: (s_i,s_{i+1}) is a state transition of *p*. A state sequence $<s_1,s_2,s_3,\ldots,s_{n-1},s_n>$ is called a *computation* of *p* iff s_1 is a starting state of *p*.

Let *S* be a state predicate of *p*. *S* is *closed* in *p* iff for each action (guard) → (statement) in *p*, executing statement starting from a state where (*S* ∧ guard) holds results in a state where *S* holds. *S* is an invariant of *p* iff *S* is true at all the initial states of *p* and *S* is closed in *p*.

Let us partition the variables of *p* into *C* (for "critical" variables) and *NC* (for "non-critical" variables); as these names suggest, the sequence of changes on the critical variables is material for correctness, the changes on the non-critical variables are not. In other words, *SPEC*, the specification that *p* satisfies, depends only on C.

Let *s* be a state of *p*. Let *v* be a subset of the variables of *p*. $s|_v$ is a set of states of *p* with the same values for each variable in *v*.

C. Definition of Security

In this paper, we use the concepts of closure, convergence and protection [7] to explain and verify the security properties of interest. Let *SPEC* be a system specification describing the allowed computations of the system in the absence of any intruder and fault actions. Intuitively speaking, for a system to be secure, certain "critical system variables" identified in *SPEC* must be protected; that is, modifications on these variables must be the same whether or not any intruder or fault actions occur. In other words, any mismatching state transition on the critical variables in the absence and the presence of an intruder or faults implies violation of the *SPEC* for that intruder and fault model.

More specifically, given a protocol that satisfies *SPEC*, the states reached by the system in the absence of any intruder or fault actions satisfy an "invariant" state predicate. Also, the states reached by the system in the presence of the intruder and fault actions satisfy a potentially weaker invariant, which we call the "fault span". Our approach to protection is to establish that for every state transition on critical variables in the fault span states, the same state transition exists in the invariant states.

Formally, let *S* be a closed state predicate of protocol *p* and *F* be a set of actions of an intruder. We say ***p* is *F*-secure for *SPEC* in *C* from *S*** iff there exists a state predicate *T* that satisfies the following conditions:

- *S* is true at any of the initial states of *p*.
- At any state where *S* is true, *T* is also true. (In other words, $S \Rightarrow T$)
- Starting from any state where *T* is true, if any action in *p* or *F* is executed, the resulting state is also one where *T* is true. (In other words, *T* is closed in *p* and *T* is closed in *F*)
- For any state *t* where *T* is true and any action of *p* or *F* that whose execution in that state changes the values assigned to one or more *C* variables, if there exists a state *s* in *S* such that $t|_C = s|_C$, then there exists an action of *p* which yields the same change of values to the *C* variables from *s* (the values of the *NC* variables may be different in the witness step). (In other words, the critical variables *C* are protected inside the state predicate *T*.)

Should we wish liveness in the presence of an intruder, we add one more clause to the definition above

- Starting from any state where *T* is true, every computation of *p* alone eventually reaches a state where *S* is true.

Our work can be regarded as invariant based approach using forward search, in Meadows' classification of formal

methods in cryptographic protocol analysis [12]. Our approach is related to that of Paulson's [15] and Cohen's [2]. The essential difference between our approach and theirs is that we limit the verification check of protection condition to the critical variables of the system. This definition has been used to prove the security of the PING protocol [8], Compaq's Millicent and IBM's Micropayments digital cash protocols, and Secure Socket Layer (SSL) protocol [11].

IV. The Protocol in Abstract Protocol Notation

In this section, we formalize *Whisper* as outlined in Section I using Abstract Protocol Notation. Process A has arrays X_A and X_B for storing key–parts. Similarly process B has variables Y_A and Y_B. A's key–parts are stored in X_A in process A and in Y_A in process B. Similarly B's key–parts are stored in X_B in process A and in Y_B in process B. For optimization, processes A and B cache the computed secrets in arrays C_{AB} and C_{BA}**.** For all $i \geq 0$, $C_{AB}[i] = f(X_A[i], X_B[i])$ and $C_{BA}[i] = f(Y_A[i], Y_B[i])$.

process A
inp C_A : **integer**
var k, n, i_A, hX_A, $tempX_B$, $tempC_{AB}$: **integer**, {initially, $i_A = 1$}
X_A, X_B, C_{AB} : **array** [**integer**] of **integer** {initially, $X_A[i]=X_B[i]=C_{AB}[i]= \perp$ for all $i \geq i_A$}
begin
A_0 :: $(X_A[i_A] = \perp \wedge X_B[i_A] = \perp) \rightarrow X_A[i_A] := rand( )$;
$hX_A := h(C_{AB}[i_A - 1], X_A[i_A])$;
send $m_0(X_A[i_A] + h(X_A[i_A-1],B), hX_A)$ **to** B
A_1 :: **rcv** m_1(k,n) **from** B $\rightarrow$ $tempX_B := k - h(X_B[i_A-1],A)$;
$tempC_{AB} := f(X_A[i_A], tempX_B)$;
if$(h(tempC_{AB},tempX_B) = n) \rightarrow X_B[i_A] := tempX_B$;
$C_{AB}[i_A] := tempC_{AB}$;
$i_A := i_A + 1$
$(h(tempC_{AB},tempX_B) \neq n) \rightarrow$ skip
fi
A_2 :: **timeout** $(X_A[i_A] \neq \perp \wedge$ ch.A.B = $\langle\ \rangle \wedge$ ch.B.A = $\langle\ \rangle) \rightarrow$ **send** $m_0(X_A[i_A] + h(X_A[i_A-1],B), hX_A)$ **to** B
end

process B
inp C_B : **integer**
var k, n, i_B, hY_B, $tempY_A$: **integer**, {initially, $i_B = 1$}
Y_A, Y_B, C_{BA} : **array** [**integer**] of **integer** {initially, $Y_A[0]=X_A[0], Y_B[0]=X_B[0]$,
$C_{AB}[0]=C_{BA}[0]=f(X_A[0], X_B[0]), Y_A[i]=Y_B[i]=C_{BA}[i]= \perp$ for all $i \geq i_B$}
begin
B_0 :: **rcv** m_0(k,n) **from** A $\rightarrow tempY_A := k - h(Y_A[i_B-1],B)$;
if$(h(C_{BA}[i_B-1],tempY_A) = n) \rightarrow Y_A[i_B] := tempY_A$;
$Y_B[i_B] := rand( )$;
$C_{BA}[i_B] := f(Y_A[i_B], Y_B[i_B])$;
$hY_B := h(Y_B[i_B-1],A)$;
send $m_1(Y_B[i_B]+hY_B, h(C_{BA}[i_B],Y_B[i_B]))$ **to** A;
$i_B := i_B + 1$
$(h(C_{BA}[i_B-1],tempY_A) \neq n) \rightarrow tempY_A := k - h(Y_A[i_B-2],B)$;
if$(h(C_{BA}[i_B-2], tempY_A) = n) \rightarrow$
send $m_1(Y_B[i_B-1]+hY_B, h(C_{BA}[i_B-1],Y_B[i_B-1]))$ **to** A
$(h(C_{BA}[i_B-2], tempY_A) \neq n) \rightarrow$ skip
fi
fi
end

Fig. 1. Whisper Protocol in Abstract Protocol Notation.

V. SECURITY AND FAULT-TOLERANCE PROPERTIES

A. Security Properties

The assumption for this proof is, h is a pre-image resistant hash function, i.e. it is computationally infeasible to find any pre-image x such that $h(x) = y$ when given any y for which a corresponding input is not known.

Informally, the *SPEC* of *Whisper* (consisting of processes A and B) is that not only the values of $X_A[i]$, $Y_B[i]$, $C_{BA}[i]$ and $C_{AB}[i]$ are kept secret but also an intruder cannot affect the values of $Y_A[i]$ and $X_B[i]$. The latter can be reduced to one which says that for all i, corresponding values of $X_A[i]$ in A and $Y_A[i]$ in B, and $X_B[i]$ in A and $Y_B[i]$ in B do match.

Formally,

$SPEC \equiv \{C_{AB}[i], C_{BA}[i]\} \cap analz(M) = \{\}$

$\wedge ((X_A[i_A] \neq \perp \wedge Y_A[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_A[i_A] = Y_A[i_B]))$

$\wedge ((X_B[i_A] \neq \perp \wedge Y_B[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_B[i_A] = Y_B[i_B]))$

where M is the set of messages principals A, B exchange over the channels ch.A.B and ch.B.A.

The critical variables C of *Whisper* are X_A, X_B, Y_A, Y_B, i_A and i_B. Recall that *Whisper* starts in a state where ($i_A = i_B = 0 \wedge X_A[i_A] = Y_A[i_B] \wedge X_B[i_A] = Y_B[i_B]$). *Whisper* is F_W *-secure* for *SPEC* in C from S, where $S \equiv S_0 \vee S_1 \vee S_2$,

$S_0 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}1])$

$\wedge (X_A[i_A] \neq \perp) \wedge (Y_A[i_B] = X_B[i_A] = Y_B[i_B] = \perp)$

$\wedge (i_A = i_B)$

$\wedge$ (ch.A.B =

$\langle m_0(X_A[i_A]+h(X_A[i_A\text{-}1],B), h(C_{AB}[i_A\text{-}1], X_A[i_A]))\rangle)$

$\wedge$ (ch.B.A = $\langle\ \rangle$)

$S_1 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}2]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}2])$

$\wedge (X_A[i_A] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A] = \perp) \wedge (Y_B[i_B\text{-}1] \neq \perp)$

$\wedge (Y_A[i_B] = Y_B[i_B] = \perp)$

$\wedge (i_A = i_B - 1)$

$\wedge$ (ch.B.A =

$\langle m_1(Y_B[i_B\text{-}1]+ h(Y_B[i_B\text{-}2],A), h(C_{BA}[i_B\text{-}1], Y_B[i_B\text{-}1]))\rangle)$

$\wedge$ (ch.A.B = $\langle\ \rangle$)

$S_2 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}1])$

$\wedge (X_A[i_A] = Y_A[i_B] = X_B[i_A] = Y_B[i_B] = \perp)$

$\wedge (i_A = i_B)$

$\wedge$ (ch.A.B = $\langle\ \rangle$)

$\wedge$ (ch.B.A = $\langle\ \rangle$)

The last two conjuncts of *SPEC* hold at S. Intuitively speaking, since *Whisper* is F_W *-secure* for *SPEC* in C from S, the same two conjuncts also hold in the presence of faults and intrusion. Hence *Whisper* continues to satisfy its *SPEC* in the presence of faults and intrusion. The complete formal proof appears in [13].

We recall the definitions of backward and forward secrecy [10]. Backward secrecy guarantees that a passive adversary who knows a contiguous subset of secrets cannot discover preceding secrets. Forward secrecy guarantees that a passive adversary who knows a contiguous subset of old secrets cannot discover subsequent secrets.

Intuitively speaking, even if $C_{AB}[i]$ or the key–parts of $C_{AB}[i]$ are leaked, an intruder cannot get $C_{AB}[i\text{-}1]$ or the key–parts of $C_{AB}[i\text{-}1]$ because function h is one-way. Hence, *Whisper* provides backward secrecy. Assuming function f is one-way, even if $C_{AB}[i]$ is leaked, an intruder cannot get $C_{AB}[i+1]$ or the key–parts of $C_{AB}[i+1]$ because function h is one-way. Hence, *Whisper* provides forward secrecy if function f is one-way. Formal proofs of these properties appear in [13].

B. Fault-tolerance Properties

Whisper has a property that, principals A and B are never out of synchronization by more than one session, as we prove in [13]. Hence it is sufficient for a principal to remember the secrets of at the most last two consecutive sessions. Hence the infinite arrays X_A, X_B, C_{AB} in process A can be replaced by arrays with two values each and integer variable i_A by single digit binary number. Similarly for variables Y_A, Y_B, C_{BA} and i_B of process B.

Whisper is self-stabilizing to S with respect to arbitrary corruption of all variables of A and B, except for variables X_A, Y_A, X_B, Y_B, i_A and i_B, as we prove in [13]. Therefore *Whisper* is able to recover to its invariant S upon corruption of all variables of A and B, except for variables X_A, Y_A, X_B, Y_B, i_A and i_B, without compromising security. The corruptible variables can be kept in volatile memory of a principal while all the other variables are kept in non-volatile memory.

VI. RELATED WORK AND CONCLUDING REMARKS

Secret agreement using asymmetric cryptography has a long history in the context of network protocols for telecommunication networks. Diffie-Hellman [3], RSA [17] and ElGamal [5] are prominent examples. Other related work deals with secret agreement in mobile ad-hoc networks, such as Balfanz et al [1] and Hubaux et al [9]; these works also use asymmetric cryptography while we use the less expensive symmetric cryptography. A detailed survey of ad-hoc network related security protocols can be found in [19].

Perrig et al [16] recently proposed SPINS, which has two building blocks SNEP and µTesla. SNEP provides data confidentiality, two-party data authentication, and data freshness, and µTesla provides efficient broadcast authentication. SPINS does not deal with secret maintenance, and in this regard it can be used in

combination with *Whisper*. SPINS does however provide – just as we did– a way to bootstrap the initial secret between two neighboring nodes using a base station as a trusted agent. However, it does not deal with the denial-of-service attack of a malicious node sending spurious key request messages (a malicious node can forge a new request message in SPINS). Also, *Whisper* does not require real-time synchronization in contrast to SPINS.

In conclusion, secret agreement establishment and maintenance in a large-scale resource-constrained sensor has requirements that are not met by classical secret agreement protocols. To the best of our knowledge, *Whisper* is the first piece of work in which neighboring node-to-node local secret maintenance with the property of forward secrecy is achieved using session keys only.

ACKNOWLEDGMENT

We thank Ted Herman, Mikhail Nesterenko, Bill Leal, Prabal Dutta and Vinodkrishnan Kulathumani for helpful discussions and comments.

REFERENCES

[1] D. Balfanz, D. K. Smetters, P. Stewart, and H. Chi Wong, "Talking to strangers: authentication in ad-hoc wireless network," *Symposium on Network and Distributed Systems Security*, San Diego CA, February 2002.

[2] E. Cohen. TAPS, "A first-order verifier for cryptographic protocols.," *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 144-158, 2000.

[3] W. Diffie, and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol 22 , pages 644-654, 1976.

[4] D. Dolev, and A. C. Yao, "On the security of public key protocols," *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350-357, 1981.

[5] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, Vol 31, pages 469-472, 1985.

[6] M. G. Gouda, "*Elements of Network Protocol Design*," John Wiley & Sons, New York, NY, 1998.

[7] M. G. Gouda, "Elements of security: closure, convergence and protection," *Information Processing Letter*, vol 77, pages 109-114, 2001.

[8] M. G. Gouda, C.-T. Huang, and A. Arora, "On the Security and Vulnerability of PING," *Fifth International Workshop on Self-Stabilizing Systems*, 2001.

[9] J. P. Hubuax, L. Buttyan, and S. Capkun, "The quest for security in mobile ad-hoc networks," *ACM MobiHoc* , October 2001.

[10] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," *Proceedings of 7th ACM Conference on Computer and Communications Security*, ACM Press, pages 235-244, November 2000.

[11] D. F. Lee, "A formal presentation of electronic commerce protocols,"ł *The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-02-33*, 2002.

[12] C. Meadows, "Invariant generation techniques in cryptographic protocol analysis," *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 159-167, 2000.

[13] V. Naik, A. Arora, S. Bapat, and M. Gouda, "Whisper: Local Secret Maintenance in Sensor Networks," Technical Report OSU-CISRC-1/03-TR04, The Ohio State University, Columbus OH, January 2003.

[14] L. Paulson, "Proving properties of security protocols by induction," *Proceedings of the IEEE Computer Security Foundations Workshop X* , , IEEE Computer Society Press, pages 70-83, 1997.

[15] L. Paulson, "Proving Security Protocols Correct. The inductive approach to verifying cryptographic protocol," *Journal of Computer Security*, vol 6, pages 85-128, 1998.

[16] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," *MOBICOM 2001*, Rome Italy, June 2001.

[17] R. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol 21, Issue 2 , pages 120-126, 1978.

[18] D. S. Wong, H. Fuentes, and A. H. Chan, "The performance measurement of cryptographic primitives on palm devices,"ł *Proc. of the 17th Annual Computer Security Applications Conference*, December 2001.

[19] L. Zhou, Z. J. Haas, "Securing Ad Hoc Networks," *IEEE Network*, Vol 13, issue 6, 24-30, 1999.