Whisper: Local Secret Maintenance in Sensor Networks[1]

Vinayak Naik Anish Arora Sandip Bapat Mohamed Gouda[†]

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210, USA
{naik, anish, bapat}@cis.ohio-state.edu

[†]Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, USA
gouda@cs.utexas.edu

Abstract. A challenge in resource-constrained sensor networks is to provide secure communication in an efficient manner, even in the presence of denial-of-service attacks. In this paper, we present a simple protocol for secret maintenance between a pair of network neighbors. We prove that Dolev-Yao adversaries cannot compromise the current secret shared by the neighbors, nor can they cause the neighbors to unduly waste resources. 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. Finally, we propose several ways of bootstrapping the initial secrets of the neighbors.

Keywords: Secure communication, Forward secrecy, Backward secrecy, Denial of service, Self-synchronizing, Self-stabilizing, Low cost

[1] This work was partially sponsored by DARPA contract OSU-RF \#F33615-01-C-1901, NSF grant NSF-CCR-9972368, an Ameritech Faculty Fellowship, and two grants from Microsoft Research. This paper has been cleared through author affiliations.

1. 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 establishing and in particular maintaining secrets in sensor networks.

Desired properties of secret establishment and maintenance in sensor networks are forward secrecy, backward secrecy, scalability, tolerance to loss of synchronization, tolerance to state corruption of the entities, and tolerance to denial-of-service attacks. Forward secrecy means that compromise of the current session key[2] 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 sensor network 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. And withstanding denial-of-service attacks, which try to unduly waste resources of sensor nodes, is crucial in determining the lifetime of the network.

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. Hence these solutions are not suitable for sensor nodes that have limited resources. An exception is SPINS [14], which as discussed in Section 9, does not handle denial-of-service attacks. 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 protocol is thus suitable for execution on resource constrained sensor nodes. 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

[2] Key and Secret are used interchangeably in this paper. Both have the same meaning.

the protocol, as described in Table 1. 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.

Table 1. Secret Update

Session	1st part	2nd part	secret
i-1	$X_A[i-1]$	$Y_B[i-1]$	$C_{AB}[i-1] = f(X_A[i-1], Y_B[i-1])$
I	$X_A[i]$	$Y_B[i]$	$C_{AB}[i] = f(X_A[i], Y_B[i])$

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.

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

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

Organization of the paper In Section 2, we describe the network, intruder and fault model that we consider in this paper. In Section 3, 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 4. In Section 5, we give formal proofs of the security and fault-tolerance properties of *Whisper*. Section 6 proposes a variety of ways to bootstrap the initial secrets in the sensor nodes. Given the importance of defending against denial-of-service attacks in the resource constrained environments, we extend the protocol by adding a notion of count in Section 7. In Section 8, we discuss related work and make concluding remarks.

2. System Model

2.1 Network Model

The network consists of sensor nodes that are small battery powered devices which may 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, sensor 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.

2.2 Intruder Model

The intruder model assumed here is the one proposed by Dolev-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 [12]. 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 [12]:

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

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

We do not consider the jamming of radio channel with a strong signal in this paper.

2.3 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 sensor node. These faults result in garbage values in the corrupted variables.

3. Programming Notation & Concepts

3.1 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, a 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

Executing an action of process *A* can cause a message to be sent to process *B*. We model the broadcast radio-based communication by two channels between two processes: one is from *A* to *B*, and the other one is from *B* to *A*. Each sent message sent from *A* to *B* remains in the channel from *A* to *B* until it is eventually received by process *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.

3.2 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 $\langle \text{guard} \rangle \rightarrow \langle \text{statement} \rangle$ such that ($s \wedge$ 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 $\forall i{:}1 \leq i \leq n\text{-}1$: (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) $\rightarrow$ (statement) in p, executing statement starting from a state where ($S \wedge$ 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.

3.3 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 [11]. Our approach is related to that of Paulson's [13] 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.

4. The Protocol in Abstract Protocol Notation

In this section, we formalize *Whisper* as outlined in Section 1 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 $C_{BA}[i] = f(Y_A[i], Y_B[i])$. (We later prove in Corollary 1.1 that $(X_A[i] = Y_A[i]) \wedge (X_B[i] = Y_B[i])$; hence $C_{AB}[i] = C_{BA}[i]$.)

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 Σ 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
$\square$ 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) →
X$_B$[i$_A$] := tempX$_B$;
C$_{AB}$[i$_A$] := tempC$_{AB}$;
i$_A$:= i$_A$ + 1
□ (h(tempC$_{AB}$,tempX$_B$) ≠ n) → skip
fi

□ A$_2$ :: **timeout**
(X$_A$[i$_A$] ≠ ⊥ ∧ ch.A.B = ⟨ ⟩ ∧ ch.B.A = ⟨ ⟩) →
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]= ⊥ for all i Σ i$_B$}
begin
B$_0$ :: **rcv** m$_0$(k,n) **from** A → tempY$_A$:= k – h(Y$_A$[i$_B$-1],B);
if(h(C$_{BA}$[i$_B$-1],tempY$_A$) = n) →
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$_A$) ≠ n) →
tempY$_A$:= k – h(Y$_A$[i$_B$-2],B);
if(h(C$_{BA}$[i$_B$-2], tempY$_A$) = n) →
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$_A$) ≠ n) → skip
fi
fi
end

5. Proofs of Security and Fault-tolerance Properties

5.1 Proof of 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,

$$SPEC \equiv \{X_A[i_A], Y_B[i_B], f(Y_A[i_B], Y_B[i_B]), f(X_A[i_A], X_B[i_A])\} \cap \text{analz}(M) = \Phi$$
$$\wedge ((X_A[i_A] \neq \perp \wedge Y_A[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_A[i_A] = Y_A[i_B]))$$
$$\wedge ((X_B[i_A] \neq \perp \wedge Y_B[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_B[i_A] = Y_B[i_B]))$$

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 $(i_A = i_B = 0 \wedge X_A[i_A] = Y_A[i_B] \wedge X_B[i_A] = Y_B[i_B])$

Lemma 1: The invariant *S* of *Whisper* is $S_0 \quad S_1 \quad S_2$, where

$$S_0 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}1])$$
$$\wedge (X_A[i_A] \neq \perp) \wedge (Y_A[i_B] = X_B[i_A] = Y_B[i_B] = \perp)$$
$$\wedge (i_A = i_B)$$
$$\wedge (\text{ch.A.B} = \langle m_0(X_A[i_A]+h(X_A[i_A\text{-}1],B), h(C_{AB}[i_A\text{-}1], X_A[i_A]))\rangle)$$
$$\wedge (\text{ch.B.A} = \langle\ \rangle)$$

$$S_1 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}2]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}2])$$
$$\wedge\ (X_A[i_A] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A] = \perp) \wedge (Y_B[i_B\text{-}1] \neq \perp)$$
$$\wedge\ (Y_A[i_B] = Y_B[i_B] = \perp)$$
$$\wedge\ (i_A = i_B - 1)$$
$$\wedge\ (\text{ch.B.A} = \langle m_1(Y_B[i_B\text{-}1]+ h(Y_B[i_B\text{-}2],A), h(C_{BA}[i_B\text{-}1], Y_B[i_B\text{-}1]))\rangle)$$
$$\wedge\ (\text{ch.A.B} = \langle\ \rangle)$$

$$S_2 \equiv (X_A[i_A\text{-}1] = Y_A[i_B\text{-}1]) \wedge (X_B[i_A\text{-}1] = Y_B[i_B\text{-}1])$$
$$\wedge\ (X_A[i_A] = Y_A[i_B] = X_B[i_A] = Y_B[i_B] = \perp)$$
$$\wedge\ (i_A = i_B)$$
$$\wedge\ (\text{ch.A.B} = \langle\ \rangle)$$
$$\wedge\ (\text{ch.B.A} = \langle\ \rangle)$$

Note that state predicates S_0, S_1 and S_2 are mutually exclusive.

Proof: S_2 holds in the initial state. In the following table, we enumerate each state predicate of *Whisper*, all actions that are enabled in that state predicate and the state predicate resulting from the execution of those actions.

Current State	Action	Next State
S_0	B_0	S_1
S_1	A_1	S_2
S_2	A_0	S_0

■

Let $T \equiv T_0 \wedge T_1$ be the fault span state predicate, where

$$T_0 \equiv (m_0(k,n)\#\text{ch.A.B} \geq 1 \wedge h(C_{BA}[i_B\text{-}1],k\text{-}h(Y_A[i_B\text{-}1],B)) = n) \Rightarrow$$

$(k\text{-}h(Y_A[i_B\text{-}1],B) = X_A[i_A] \wedge i_A = i_B)$

$T_1 \equiv (m_1(k,n)\#ch.B.A \geq 1 \wedge h(C',k\text{-}h(X_B[i_A\text{-}1],B)) = n) \Rightarrow$

$(k\text{-}h(X_B[i_A\text{-}1],B) = Y_B[i_B\text{-}1] \wedge i_A = i_B - 1)$

where $C' = f(X_A[i_A], k\text{-}h(X_B[i_A\text{-}1],B))$

Note that, T_0 and T_1 cannot both hold non-vacuously in any state due to the conditions on the values of variables i_A and i_B. Hence given T is true in a state and T_0 is holding non-vacuously, then T_1 must be holding vacuously. In that case we write $\boldsymbol{T_0} \wedge T_1$. Similarly, given T is true in a state and T_1 is holding non-vacuously, then T_0 must be holding vacuously. In that case we write $T_0 \wedge \boldsymbol{T_1}$.

Lemma 2: $S \Rightarrow T$

Proof: In S_0, there is one message in ch.A.B which satisfies T_0. Channel ch.B.A is empty, hence T_1 is vacuously true. In S_1, there is one message in ch.B.A which satisfies T_1. Channel ch.A.B is empty, hence T_0 is vacuously true. In S_2, both the channels ch.A.B and ch.B.A are empty; hence T_0 and T_1 are vacuously true. Therefore, $S \Rightarrow T$. ■

Lemma 3: T is closed in *Whisper*

Proof: T holds in the initial state when there are no messages in the channels.

Current State	Action	Next State
T	A_0,A_2	$\boldsymbol{T_0} \wedge T_1$
$\boldsymbol{T_0} \wedge T_1$	B_0	$T_0 \wedge \boldsymbol{T_1}$
$T_0 \wedge \boldsymbol{T_1}$	A_1	T

■

Lemma 4: T is closed in F, where F is the set of intruder's actions as modeled in Section 2.2

Proof: Interception and replay actions do not violate the state predicate T. The case of pre-play of message needs to be verified. Let us consider any two consecutive sessions of *Whisper*. In the $(i_A)^{th}$ session, results of applying *analz* operator to messages m_0 and m_1 are

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

$analz(m_1) = \{A, B, Y_B[i_B] + h(Y_B[i_B\text{-}1],A), h(C_{BA}[i_B], Y_B[i_B])\}$.

$X_A[i_A]$, $Y_B[i_B]$ and $C_{BA}[i_B]$ appear in *parts* of any message for the first time in this session. Still $X_A[i_A] \notin analz(m_0, m_1)$, $Y_B[i_B] \notin analz(m_0, m_1)$ and $C_{BA}[i_B] \notin analz(m_0, m_1)$.

Similarly in the $(i_A+1)^{th}$ session, results of applying *analz* operator to messages m_0' and m_1' are

$analz(m_0') = \{A, B, X_A[i_A+1] + h(X_A[i_A],B), h(C_{AB}[i_A], X_A[i_A+1])\}$

$analz(m_1') = \{A, B, Y_B[i_B+1] + h(Y_B[i_B],A), h(C_{BA}[i_B+1], Y_B[i_B+1])\}$.

$X_A[i_A] \notin analz(m_0', m_1')$, $Y_B[i_B] \notin analz(m_0', m_1')$ and $C_{AB}[i_A] \notin analz(m_0', m_1')$. Hence neither key–parts nor key are revealed in any of the messages.

Since $\{x + h(X_A[i_A],B),\ h(C_{AB}[i_A],x)\} \notin fake(m_0,m_1,m_0')$ such that $x \neq X_A[i_A+1]$, an intruder cannot synthesize a new message m_0' that violates T_0. Similarly since $\{y + h(Y_B[i_B],A),\ h(C',y)\} \notin fake(m_0,m_1,m_0',m_1'))$ such that $y \neq Y_B[i_B+1]$ and $C' = f(X_A[i_A], y)$, an intruder cannot synthesize a new message m_1' that violates T_1. Therefore T is closed in F. ■

Lemma 5: The critical variables C of *Whisper* are protected inside T
Proof: In the following table, we enumerate each state predicate of *Whisper* in the presence of an intruder, the state of the critical variables, an action that is enabled in that state of *Whisper*, the state predicate resulting from the execution of that action and the state of the critical variables in the resulting state. (n/a implies that Current state Critical Variables = false) e.g. When the current state of *Whisper* is $T_0 \wedge \boldsymbol{T_1}$, i_A is equal to i_B - 1 and when the state of the critical variables is S_0, i_A is equal to i_B. Hence the critical variables cannot be in state S_0 when *Whisper* is in state $T_0 \wedge \boldsymbol{T_1}$.

Current State	Critical Variables	Action	Next State	Critical Variables
$\boldsymbol{T_0} \wedge T_1$	S_0	B_0	$T_0 \wedge \boldsymbol{T_1}$	S_1
$T_0 \wedge \boldsymbol{T_1}$	S_0	n/a	n/a	n/a
T	S_0	A_2	$\boldsymbol{T_0} \wedge T_1$	S_0
$\boldsymbol{T_0} \wedge T_1$	S_1	n/a	n/a	n/a
$T_0 \wedge \boldsymbol{T_1}$	S_1	A_1	T	S_2
T	S_1	A_2	T	S_1
$\boldsymbol{T_0} \wedge T_1$	S_2	n/a	n/a	n/a
$T_0 \wedge \boldsymbol{T_1}$	S_2	n/a	n/a	n/a
T	S_2	A_0	$\boldsymbol{T_0} \wedge T_1$	S_0

■

Theorem 1: *Whisper* is *F-secure* for *SPEC* in C from S
Proof: It follows from Lemmas 1-5 and the definition of security. ■

Corollary 1.1: $((X_A[i_A] \neq \perp \wedge Y_A[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_A[i_A] = Y_A[i_B]))$
$\wedge ((X_B[i_A] \neq \perp \wedge Y_B[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_B[i_A] = Y_B[i_B]))$
Proof: From Theorem 1 and the protection condition, for each state t (where T is true) in the computation of *Whisper* there exists a state s in S such that $t|_C = s|_C$. I Since $((X_A[i_A] \neq \perp \wedge Y_A[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_A[i_A] = Y_A[i_B]))$
$\wedge ((X_B[i_A] \neq \perp \wedge Y_B[i_B] \neq \perp \wedge i_A = i_B) \Rightarrow (X_B[i_A] = Y_B[i_B]))$ holds in S, it holds in T. ■

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.

Theorem 2: *Whisper* provides backward secrecy
Proof: Let m_{0i}, m_{1i} be the messages exchanged by A and B during i^{th} session of *Whisper*.

$analz(m_{0i}) = \{A, B, X_A[i] + h(X_A[i-1],B), h(C_{AB}[i-1], X_A[i])\}$,
$analz(m_{1i}) = \{A, B, Y_B[i] + h(Y_B[i-1],A), h(C_{BA}[i], Y_B[i])\}$

Let us assume adversary knows the secrets starting from the K^{th} session up to M^{th} session and i = K-1.
From the proof of Lemma 4, $X_A[i] \notin analz(m_{0i}, m_{1i})$, $Y_B[i] \notin analz(m_{0i}, m_{1i})$, $X_A[i] \notin analz(m_{0i+1}, m_{1i+1})$ and $Y_B[i] \notin analz(m_{0i+1}, m_{1i+1})$. From protocol actions, $(\forall j : j < K - 1: X_A[i] \notin analz(m_{0j}, m_{1j})$ and $Y_B[i] \notin analz(m_{0j}, m_{1j}))$. Also $(\forall j : K \leq j \leq M: X_A[i] \notin analz(m_{0j}, m_{1j},C_{AB}[j])$ and $Y_B[i] \notin analz(m_{0j}, m_{1j},C_{AB}[j]))$. Hence the $(K-1)^{th}$ secret is not revealed. Using induction on i (for all $i \leq K - 2$), secrets for all the sessions less than K are not revealed. ■

Theorem 3: *Whisper* provides forward secrecy if function *f* is one-way
Proof: Let m_{0i}, m_{1i} be the messages exchanged by A and B during i^{th} session of *Whisper*.

$analz(m_{0i}) = \{A, B, X_A[i] + h(X_A[i-1],B), h(C_{AB}[i-1], X_A[i])\}$
$analz(m_{1i}) = \{A, B, Y_B[i] + h(Y_B[i-1],A), h(C_{BA}[i], Y_B[i])\}$

Let us assume adversary knows the secrets starting from the K^{th} session up to M^{th} session and i = M + 1.
Since *f* is one-way, $X_A[i] \notin analz(C_{AB}[i])$ and $Y_B[i] \notin analz(C_{AB}[i])$. From the proof of Lemma 4, $X_A[i] \notin analz(m_{0i}, m_{1i})$, $Y_B[i] \notin analz(m_{0i}, m_{1i})$, $X_A[i] \notin analz(m_{0i+1}, m_{1i+1})$ and $Y_B[i] \notin analz(m_{0i+1}, m_{1i+1})$. From protocol actions, $(\forall j : j > M + 2 : X_A[i] \notin analz(m_{0j}, m_{1j})$ and $Y_B[i] \notin analz(m_{0j}, m_{1j}))$. Also $(\forall j : K \leq j \leq M: X_A[i] \notin analz(m_{0j}, m_{1j},C_{AB}[j])$ and $Y_B[i] \notin analz(m_{0j}, m_{1j},C_{AB}[j]))$. Hence the $(M+1)^{th}$ secret is not revealed. Using induction on i (for all $i \geq M + 2$), secrets for all the sessions greater than M are not revealed. ■

For reasons of space, we omit the proof of liveness of *Whisper* in the presence of an intruder.

5.2 Proof of Fault-Tolerance Properties

Theorem 4: Principals A and B are never out of synchronization by more than one session
Proof: From Theorem 1 and the protection condition, for each state *t* (where *T* is true) in the computation of *Whisper* there exists a state *s* in *S* such that $t|_C = s|_C$. In *T*, $(i_A = i_B) \vee (i_A = i_B - 1)$. Therefore, for each state *t* (where *T* is true) in the

computation of *Whisper*, $(i_A = i_B) \vee (i_A = i_B - 1)$. Therefore principals A and B are never out of synchronization by more than one session. ■

The implication of Theorem 4 is that it is sufficient for a principal to remember the secrets of at the most 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.

The next theorem deals with the ability of *Whisper* 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.

Thorem 5: *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
Proof: From Theorem 1, *Whisper* is *F-secure* for *SPEC* in *C* from *S*. The only variables of A and B in the state predicate *T* are X_A, Y_A, X_B, Y_B, i_A and i_B. The values of all the other variables which we can call "corruptible" can be arbitrary. The corruption of corruptible variables does not violate *T*. Therefore *Whisper* is still *F-secure* for *SPEC* in *C* from *S*.

All the corruptible variables except for those used as cache for optimization, such as hX_A, are calculated afresh during all the executions of every action of *Whisper*. The variable hX_A stores the value $h(C_{AB}[i_A - 1], X_A[i_A])$, which is used across two actions A_0 and A_2. To guarantee liveness of *Whisper*, we can periodically recalculate hX_A using current values of $C_{AB}[i_A - 1]$ and $X_A[i_A]$ which are not corrupted. This would guarantee liveness of *Whisper*. Hence *Whisper* is self-stabilizing to the corruption of all the variables inside the principals except for X_A, Y_A, X_B, Y_B, i_A and i_B. ■

The corruptible variables can be kept in volatile memory of a principal while all the other variables are kept in non-volatile memory.

6. Bootstrapping the Initial Secret

The initial secret between A and B can be bootstrapped using a variety of methods depending upon the level of initial trust in the network and the level of efficiency required. Efficiency is directly proportional to the initial trust in the network. In case of no initial trust in the network, a base station (principal SB) serves as a trusted authority used to bootstrap the initial secret (Multiple base stations can be deployed for load balancing purposes). SB shares a secret C_U with every principal U in the network, e.g. SB shares secrets C_A and C_B with A and B respectively. When A wishes to establish a secret with B, it contacts SB. SB replies with the two key–parts $X_A[0]$, $Y_B[0]$. A can compute $X_A[0]$ by itself and retrieve $Y_B[0]$

from the reply. Similarly, B can compute $Y_B[0]$ by itself and retrieve $X_A[0]$ from the reply. Hence both of them can compute the initial shared secret $C_{AB}[0]$.

$$A \rightarrow SB : B, h(C_A, A, B)$$
$$SB \rightarrow A, B : X_A[0] + Y_B[0], h(X_A[0]), h(Y_B[0])$$

where $X_A[0] = h(C_A, B)$; $Y_B[0] = h(C_B, A)$; $C_{AB}[0] = f(X_A[0], Y_B[0])$

In case of perfect initial trust in the network, we can more efficiently bootstrap the secrets. All nodes can be provided with a common secret that has a limited lifetime t_p, such that once t_p expires a principal will no longer remember that secret. During t_p, this common secret is used to initialize the two key–parts. Once t_p has elapsed, any new principal joining the network has to resort to the base station for initiating communication with other principals. The assumption underlying this optimization is that during t_p, all principals are non-malicious.

In case of partial initial trust in the network a scheme such as key trees [8] can be used. Given is a tree of keys such that each principal is associated with a leaf node and it knows all the keys that are in the path from this leaf to the root of the tree. A and B will use the first key that they share starting from the bottom of the tree as their initial key–parts. This is more efficient than using a base station but less efficient than using a single secret all over the network.

7. Defending against Denial-of-Service Attacks

7.1 Denial-of-Service Attacks

Sensor networks with constrained resources are especially vulnerable to attacks that waste their resources. For example, in the bootstrapping protocol described above where there is a unique secret per node, an intruder E can easily replay old request messages of A and not only force the base station to send replies but also force A and B to switch to the initial secret; this would waste resources of A and B (as they have to form and remember the secret) and the base station. Similarly, during secret update an intruder can replay the old messages of A to force B to send the corresponding replies; this would force B to waste its energy on sending reply messages. These cases apply even when E may not be a part of the network, i.e. it does not even have C_E.

Sometimes a principal is compromised, and its secrets are disclosed. In this case, E can create authenticated messages. In bootstrapping, such an intruder can create valid request messages and during secret update it can send malicious update messages just to waste resources of the participating nodes.

7.2 Measures against Denial-of-Service Attacks

Since communication is radio-based, a node has to listen to all the messages in its receiving range, even if they are from malicious nodes. Hence measures need to be taken to identify and entertain only genuine messages using less energy. For defending against denial-of-service attacks, we present an enhanced version of the bootstrapping and secret update protocols.

A notion of count is introduced in each process. Every process has an internal counter, which is incremented at least once after any program action is executed. Each process maintains its current knowledge of the counter values of all other processes it is communicating with.

Bootstrapping Initial Secret

$A \rightarrow SB$: $B, t_A, h(C_A, A, B, t_A)$

$SB \rightarrow A, B$: $t_S, X_A[0] + X_B[0], h(t_S, X_A[0]), h(t_S, X_B[0])$

where t_A, t_S are current values of counters in A and SB respectively;
$X_A[0] = h(C_A, B, t_S)$; $X_B[0] = h(C_B, A, t_S)$; $C_{AB}[0] = f(X_A[0], X_B[0])$

Secret Update

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

$B \rightarrow A$: $t_B, X_B[i] + h(X_B[i-1], A), h(C_{AB}[i], X_B[i], t_B)$

where t_A, t_B are current values of counters in A and B respectively;
$X_A[i] = rand(\,)$; $X_B[i] = rand(\,)$; $C_{AB}[i] = f(X_A[i], X_B[i])$

The enhanced version described above handles denial-of-service attacks via four mechanisms, viz., self-authorizing request messages [16], synchronization, asymmetry in resource expenditure, and caching of computationally expensive messages [16]. The last two are especially important in case an intruder is a compromised node. Below we offer an intuitive explanation of the use of these mechanisms:

1. Each message contains an authentication of its sender. While bootstrapping the secrets, requester A uses C_A and the base station uses C_A and C_B to authenticate the requests. In case of secret update, requester A uses $C_{AB}[i-1]$ for authentication. The receiver expends resources only when a request is authentic. Hence an intruder can form an attack only if it sends authentic request messages. Since the secrets are not leaked to an intruder, it can only replay authentic messages, which is handled by (2).
2. Each fresh authentic message contains a sequence number which is greater than that in the previous authentic message. The receiver records the sequence number in the last authentic message from the sender. The receiver detects the replay of a message if it contains a sequence number less than or equal to that in record, in which case it discards the replayed message. For example, in bootstrapping, an intruder cannot replay the base

station's messages to force A and B to revert to the initial secret. Similarly, in secret update, an intruder cannot replay A's messages to force B to send reply messages again.

3. One way to discourage denial-of-service attacks is to force an intruder to expend more resources compared to benign nodes for establishing and maintaining malicious sessions. This is especially useful when the intruder is a compromised node and can create authentic messages. Note that in both secret bootstrap and update, an intruder at a compromised node has to do more work than benign nodes to force them to maintain sessions with it. This would deplete the resources of the malicious node.
4. Resending requests and responses causes a principal to waste a lot of resources, if it has to compute them repeatedly. In *Whisper*, A reuses the value $X_A[i] + h(X_A[i-1], B)$ in request if it has to send the request again. B can also reuse the value $X_B[i] + h(X_B[i-1], A)$ in corresponding response. Hence both A and B can cache results of the previously computed values which saves them energy when they are forced to re-send the same messages e.g. due to scrupulous collisions or due to a compromised node.

8. Related Work and Concluding Remarks

The literature on secret agreement is extensive, and we will not attempt to be comprehensive in this compilation of related work. Instead, we will discuss works that most directly influenced ours or are representative of existing ideas.

Secret agreement using asymmetric cryptography has a long history in the context of network protocols for telecommunication networks. Diffie-Hellman [3], RSA [15] 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.

Perrig et al. [14] 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.

9. Acknowledgements

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

10. 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* (February 2002), San Diego CA.
2. E. Cohen. TAPS: A first-order verifier for cryptographic protocols. *Proceedings of the 13th IEEE Computer Security Foundations Workshop* (2000), pages 144-158.
3. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory* Vol 22 (1976), pages 644-654.
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* (1981), pages 350-357.
5. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* Vol 31 (1985), pages 469-472.
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 Letters (2001)*, vol 77, pages 109-114.
8. M. G. Gouda, Chin-Tser Huang, E. N. Elnozahy. Key trees and the security of interval multicast. *Proceedings of the 22nd International Conference on Distributed Computing Systems* (July 2002), Vienna Austria, pages 467-468.
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* (Novemnber 2000), pages 235-244, ACM Press.
11. C. Meadows. Invariant generation techniques in cryptographic protocol analysis. *Proceedings of the 13th IEEE Computer Security Foundations Workshop* (2000), pages 159-167.
12. L. Paulson. Proving properties of security protocols by induction. *Proceedings of the IEEE Computer Security Foundations Workshop X* (1997), pages 70-83, IEEE Computer Society Press.
13. L. Paulson. Proving Security Protocols Correct. The inductive approach to verifying cryptographic protocol. *Journal of Computer Security* (1998), vol 6, pages 85-128.
14. A. Perrig, R. Szewczyk, V. Wen, D. Culler and J. D. Tygar. SPINS: Security protocols for sensor networks. *MOBICOM 2001* Rome Italy (June 2001).
15. 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 (1978), pages 120-126.
16. L. Zhou, F. Schneider and R. van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems* Vol 20 Issue 4 (November 2002), pages 329-368.