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

[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.

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 article, we present a simple protocol for secret maintenance between a pair of network neighbors. We claim that certain adversaries cannot compromise the current secret shared by the neighbors, nor can they cause the neighbors to unduly waste resources. Moreover, 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. 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 article we address the problem of establishing and in particular maintaining secrets in sensor networks. The sensor network provides an inexpensive, scalable, and robust alternative for existing centralized monolithic sensory devices. The applications of sensor network include intrusion tracking and classification, remote habitat monitoring etc.

Extant protocols such as Diffie-Hellman key agreement [1] 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 [2].

Our approach is to use computationally less expensive cryptographic primitives, which are less taxing for the sensor nodes. The drawback of using faster cryptographic primitives is that those primitives are less secure. We provide security by updating the secrets periodically, and by providing backward secrecy and forward secrecy. Forward secrecy means that compromise of the current session secrets does not imply compromise of future session secrets. Backward secrecy means that compromise of the current session secret does not imply compromise of past session secrets. It is proved that re-keying, if done properly, brings significant gain in the security [3].

The desired properties of secret establishment and maintenance in sensor networks are scalability, tolerance to loss of synchronization, tolerance to state corruption of the entities, and tolerance to denial-of-service attacks. The issue of scalability is of primary concern as a sensor network offers 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.

2. System Model

The network consists of sensor nodes 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 have limited computational and communication resources. Communication is radio based and is an energy-consuming function for these nodes.

We assume that each non-malicious sensor node represents a unique principal. Each principal has a unique ID and shares a secret with the base station. 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 are received or lost, one at a time, in any order and not necessarily in the same order in which they are sent.

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 secrets), and construct messages using the stored and known values. The more a secret is used in communications, the greater its chances are of being compromised. An intruder can be a malicious node in the network and

hence can engage in sessions with other neighboring nodes. We do not consider the jamming of radio signals in this article.

The primitive data types that may occur in any message are IDs of the processes and secrets. 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.

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. Mica mote

UC Berkeley's MICA mote, shown in figure 1, is one type of available sensor node. Each mote consists of a 4MHz Atmel microprocessor, 128Kbytes of programmable flash memory, and 4Kbytes of data memory. The network device is a 433MHz, amplitude shift keying RF transceiver capable of delivering up to 38.4Kbaud and emits less than 1mw of transmission power [4]. TinyOS [5] provides a programming environment and a network stack on this platform.

Fig. 1. U.C. Berkeley's mote [4].

4. *Whisper* Protocol and its Security Properties

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 in resource constrained computational model. To provide forward secrecy, we derive each session secret from two distinct "key–parts" known only to the two neighboring principals sharing the session secret. 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 secret does not reveal the current key–parts, the function selected to compute the secret from the two key–parts is one–way.

More precisely, secret maintenance in *Whisper* proceeds as follows. Let A and B be two neighboring principals that share a secret. 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 secret $C_{AB}[i-1]$ at the end of $(i-1)^{th}$ session of the protocol, as described in Table 1. To update the secret, 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	1^{st} part	2^{nd} 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. Mathematical addition is represented using +, but it can replaced by XOR (bit-wise exclusive or) operation.

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

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

The two key−parts are encrypted using the one-way hash of the previous key−parts. The session secret is used only to authenticate the key−parts. Therefore, neither the key−parts nor the secret are revealed. Since the key−parts of the previous session are used to encrypt the key−parts of the current session, compromise of the current key−parts does not reveal the key−parts of the previous sessions. Hence, *Whisper* provides backward secrecy.

The basic argument in the proof of forward secrecy is irreversibility of the hash function *f*, while that for the backward secrecy is irreversibility of *h*. Note that, the compromise of both key−parts will reveal the future session secrets. However the two key−parts are used only while updating a session secret, while the session secret is used in all the communications. Hence the likelihood of compromising them is less than that of compromising the session secrets. The use of tamper-proof hardware to store the key−parts will safeguard the key−parts from the hardware break-ins.

5. Fault-Tolerance Properties of *Whipser*

Principals A and B are never out of synchronization by more than one session. So it is sufficient for a principal to remember the key−parts of at most the last two consecutive sessions. Hence the infinite arrays X_A, Y_B in process A can be replaced by arrays with two values each and integer variable i by a single bit. Similarly this can be done for variables X_A, Y_B and i of process B. This is desirable considering the unreliable nature of the wireless medium and the limited memory available to a sensor node.

Whisper can recover upon corruption of all variables of A and B, except for variables X_A, Y_B, and i, without compromising security. All the corruptible variables are calculated afresh during all the executions of every action of *Whisper*. The corruptible variables can be kept in volatile memory of a principal while all the other variables are kept in non-volatile memory. MICA mote is provided with a flash chip that acts as a non-volatile memory to store data [4].

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 bootstrap the secrets in a more efficient manner. 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 [6] 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

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.

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 [7], synchronization, asymmetry in resource expenditure, and caching of computationally expensive messages [7]. 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 the counter value which is greater than that in the previous authentic message. The receiver records the counter value in the last authentic message from the sender. The receiver detects the replay of a message if it contains the counter value 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 malicious sessions. This is especially useful when the intruder is a compromised node and can create authentic messages. Note that in secret bootstrap the intruder has to create an authentic request message, while each benign neighboring node has to only verify a single response message which anyway is unavoidable.

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 a 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 spurious collisions or due to a compromised node.

8. Related Work and Concluding Remarks

Secret agreement using Diffie-Hellman [1], RSA [8] and ElGamal [9] has a long history in the context of network protocols for telecommunication networks. A recent work [10] by Tal Rabin is closely related in the sense that it utilizes re-keying to update the keys in the RSA-based threshold signature scheme. Other related work deals with secret agreement in mobile ad-hoc networks, such as Balfanz et al [11] and Hubaux et al [12]; these works use asymmetric cryptography while we use the less expensive hash functions. A detailed survey of ad-hoc network related security protocols can be found in [13].

A related paper is by Gong [14] who uses a similar idea of lightweight crypto to solve an authentication problem. Perrig et al [2] 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.

The conventional security protocols leverage security by using computationally expensive cryptographic primitives, which is suitable for the computational paradigm consisting of few computers that are rich in resources. The computational paradigm of sensor network is different primarily due to the lack of resources and the scale of the network. Therefore, we need to rethink about the security protocols to make them apposite to the sensor network. The two new dimensions that can be explored to provide security are temporal and spatial. In temporal dimension sensitive information is spread out in time while in spatial dimension sensitive information is spread out in space. Therefore, an intruder has to work multiple times either in time or space respectively to accomplish a successful attack. *Whisper* is an example of the use of temporal dimension to provide security.

In conclusion, secret agreement establishment and maintenance in a large-scale resource-constrained sensor network 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 secrets only.

References

1. W. Diffie, and M. Hellman. (1976). "New directions in cryptography," *IEEE Transactions on Information Theory*, Vol 22, 644-654.
2. A. Perrig et al. (2001). "SPINS: Security protocols for sensor networks," *MOBICOM 2001*.
3. M. Abdalla, M. Bellare. (2000). "Increasing the lifetime of a key: A Comparitive Analysis of the security of re-keying techniques," *Advances in Cryptology - Asiacrypt 2000 Proceedings*, Lecture Notes in Computer Science Vol 1976, 546-559.
4. J. Hill, and D. Culler. (2002). "Mica: A Wireless Platform for Deeply Embedded Networks," *IEEE Micro.*, Vol 22 Issue 6, 12-24.
5. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. (2002). "System architecture directions for networked sensors," *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*.
6. M. G. Gouda, Chin-Tser Huang, and E. N. Elnozahy. (2002). "Key trees and the security of interval multicast," *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna Austria, 467-468.
7. L. Zhou, F. Schneider and R. van Renesse. (2002). "COCA: A secure distributed online certification authority," *ACM Transactions on Computer Systems*, Vol 20 Issue 4, 329-368.
8. R. Rivest, A. Shamir and L. M. Adleman. (1978). "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol 21 Issue 2, 120-126.
9. T. ElGamal. (1985). "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, Vol 31, 469-472.
10. T. Rabin. (1998). "A Simplified Approach to Threshold and Proactive RSA," *Advances in Cryptology - Asiacrypt 1998 Proceedings*, Vol 1462, 89-104.
11. D. Balfanz et al. (2002). "Talking to strangers: authentication in ad-hoc wireless network," *Symposium on Network and Distributed Systems Security*, San Diego CA.
12. J. P. Hubuax, L. Buttyan, and S. Capkun. (2001). "The quest for security in mobile ad-hoc networks," *ACM MobiHoc*.
13. L. Buttyan, and J. Hubaux. (2003). "Report on a Working Session on Security in Wireless Ad Hoc Networks," *Mobile Computing and Communications Review*, Vol 7, Number 1.
14. L. Gong. (1989). "Using one-way functions for authentication," *Computer Communication Review*, Vol 9, Number 5, 8-11.