# SAKE: Software Attestation for Key Establishment in Sensor Networks

Arvind Seshadri, Mark Luk, and Adrian Perrig

Carnegie Mellon University, Pittsburgh PA 15213, USA,
arvinds@cs.cmu.edu, mark.luk@gmail.com, perrig@cmu.edu

**Abstract.** This paper presents a protocol called SAKE (Software Attestation for Key Establishment), for establishing a shared key between any two neighboring nodes of a sensor network. SAKE guarantees the secrecy and authenticity of the key that is established, without requiring any prior authentic or secret information in either node. In other words, the attacker can read and modify the entire memory contents of both nodes before SAKE executes. Further, to the best of our knowledge, SAKE is the only protocol that can perform key re-establishment after sensor nodes are compromised, because the presence of the attacker's code in the memory of either protocol participant does not compromise the security of SAKE. Also, the attacker can perform any active or passive attack using an arbitrary number of malicious, colluding nodes. SAKE does not require any hardware modification to the sensor nodes, human mediation, or secure side channels. However, we do assume the setting of a computationally-limited attacker that does not introduce its own computationally powerful nodes into the sensor network.
SAKE is based on ICE (Indisputable Code Execution), a primitive we introduce in previous work to dynamically establish a trusted execution environment on a remote, untrusted sensor node.

**Key words:** Key establishment, software attestation, sensor networks

## 1 Introduction

Sensor networks are expected to be deployed in the near future in many safety-critical applications such as critical infrastructure protection and surveillance, fire and burglar alarm systems, home and office automation, inventory control systems, and medical applications such as patient health monitoring. Therefore, securing sensor networks is of paramount importance.

Many security mechanisms proposed for sensor networks rely on cryptography, which makes the problem of key establishment one of the central issues in sensor network security. The problem of key establishment can be described as instantiating secret keys in the nodes of a sensor network in a manner that the authenticity and confidentiality of keys can be guaranteed. Even though key establishment in sensor networks is a well-studied problem, a major body of research that addresses the problem assumes the presence of some secret information in the nodes [1–6]. The techniques that do not assume prior secret information in the nodes namely Integrity Codes [7], Smart-Its Friends [8], Are You with Me [9], Shake Them Up [10], Key Infection [11], and Message in a Bottle (MiB) [12] all either require human mediation, extra hardware, or are insecure.

In this paper, we present SAKE, a key establishment protocol for establishing shared keys between any two neighboring nodes in a sensor network. SAKE is immune to all active and passive attacks without requiring any prior secrets or authentic information in either of the nodes. In order words, the attacker is free to read and modify the entire memory contents of both participants before the protocol executes. In addition, SAKE does not require any secure side channels between the principals. The secrecy and authenticity of the key that is established are guaranteed against an attacker that can perform any active or passive attack using a arbitrary number of compromised, colluding nodes. One consequence of this is that SAKE guarantees that a good node will never establish a key with a malicious node i.e., SAKE is fail-safe.

Another interesting property of SAKE is that the attacker's code can exist in the memory of the participating nodes during the execution of SAKE. In other words, the protocol participants can be compromised. Yet the attacker will not be able to compromise the confidentiality of the key that is established. To the best of our knowledge, this property is unique to SAKE. All other research in the area of sensor network key establishment assumes a network-based adversary, whereby the protocol participants are assumed to be uncompromised. This property of the SAKE makes it suitable for key re-establishment after node compromise since the presence of the attacker's code in either (or both) participants does not compromise the security of SAKE.

The results we present in this paper appear even more surprising since we assume commodity sensor nodes, i.e., no hardware modification is required to the sensor nodes, and do not require human mediation or secure side channels. However, we do assume that the attacker does not introduce its own computationally powerful nodes into the network. Even with this assumption, we have dramatically increased the level of difficulty for an attacker. In order to circumvent our protocol, the attacker's hardware must always be present and must participate actively in the SAKE protocol, which increases the chance of detection. For example, researchers have recently shown that it is possible to use radio fingerprinting to uniquely identify the origin of messages in a sensor network [13]. Such techniques can be used to create a list of nodes that are authorized members of the network, thereby detecting the presence of the attacker's devices. Compared to today's security scenario, where an attack could be launched remotely, SAKE presents a substantial hurdle for the attacker by requiring physical presence. As we discuss in the related work section, the SAKE protocol covers a new point in the design space, offering an appealing trade-off between ease-of-use and security.

SAKE is based on a primitive called ICE (Indisputable Code Execution) that we introduced in our earlier work [14]. Using ICE, a challenger can verify if the execution of an arbitrary piece of code on a remote sensor node will be untampered by any malicious code that might be present. In SAKE, either principal can use ICE to verify if the execution of the SAKE protocol code will be untampered, thereby obtaining the guarantee that the key that is established will not be revealed to malicious code. We implement ICE and SAKE on Telos sensor nodes.

## 2 Problem Definition, Attacker Model, and Assumptions

This section presents our problem definition, attacker model, and sensor network architecture and assumptions, in that order.

### 2.1 Problem Definition

When nodes are newly deployed or when they are compromised, they need a mechanism to establish pairwise shared keys with their neighbors. Once pairwise shared keys between neighbors are established, it is trivial to establish shared keys between non-neighboring nodes. Therefore, establishing pairwise shared keys is the fundamental problem in key establishment.

The protocol used to establish shared keys must guarantee the authenticity and confidentiality of the keys that are established. This requires the protocol to be resilient to passive and active attacks. Also, the protocol cannot assume the existence of any secret or authentic information in the participants, since such information may not exist in newly deployed nodes or might have been compromised in the case of compromised nodes.

### 2.2 Attacker Model

We assume an attacker that can compromise an arbitrary number of sensor nodes and introduce its own sensor nodes into the network. Upon compromising a sensor node, the attacker can read and modify its entire memory contents. Also, the attacker's code can exist in the memory of the SAKE protocol participants during the execution of the SAKE protocol.

The malicious nodes controlled by the attacker can arbitrarily collude. However, the attacker does not introduce computationally powerful devices, such as laptop computers, into the network. Note that introducing new hardware into the network increases the attacker's chance of being detected. The attacker's hardware must always be present in the network and must actively participate in the SAKE protocol. For example, researchers have recently shown that it is possible to use radio fingerprinting to uniquely identify the origin of messages in a sensor network [13]. Such techniques can be used to create a list of nodes that are authorized members of the network, thereby detecting the presence of the attacker's devices. In our future work, we will consider an attacker who is present in the sensor network.

An example attacker that fulfills these requirements would be a remote attacker. Such an attacker exploits software vulnerabilities, such as a buffer overflow vulnerability, to compromise a sensor node. Once a node is compromised, the attacker can gain full control over the node. For example, the attacker can inject and run arbitrary code, and steal cryptographic keys.

### 2.3 Sensor Network Architecture and Assumptions

We consider a wireless sensor network consisting of one or more base stations and several sensor nodes. All sensor nodes have identical hardware configurations. The sensor nodes communicate among themselves and the base station using wireless communication links. Every sensor node has an unique, immutable identifier, hereafter referred to as its *Silicon ID*. Such Silicon IDs are available on commodity motes. Examples include the 64-bit DS2401 silicon serial number chip present on the Mica2 motes [15], and the 48-bit serial ID chip present on the Telos motes [16]. The immutability of the Silicon ID implies that an attacker cannot modify it, even when it compromises the sensor node. The sensor network is assumed to have a reliable transport layer protocol like PSFQ [17]. The SAKE protocols require that all protocol messages be reliably delivered between the participants. Finally, the nodes are assumed to have a hardware source of randomness. This source of randomness is used to generate the shared key that is established and for other cryptographic operations required by SAKE. A hardware source of randomness ensures that the attacker cannot know the output of the random source during the SAKE's execution, even if it knows the entire memory contents of the sensor node before SAKE beings executing.

The base stations are the gateway between the sensor network and the outside world. Other sensor networks or computers on the Internet can send network packets directly to the sensor nodes but these packets will always pass through at least one of the base stations. This assumption is justified since the nodes have short-range wireless radios. We also assume that the base stations are trusted. This assumption is commonly made by most sensor network security researchers since compromise of the base stations in most cases means the compromise of the network.

## 3 The SAKE Protocol

In this section we describe the SAKE protocol. Section 3.1 presents an overview of SAKE. Section 3.2 describes the construction of the SAKE in detail. In Section 3.3, we discuss how we select cryptographic primitives for SAKE and present our evaluation.

### 3.1 Overview

SAKE uses the well-known Diffie-Hellman (DH) key exchange protocol to establish a shared key between the two participating neighboring nodes. The DH protocol is immune to passive attacks. However, it is vulnerable to the man-in-the-middle (MitM) attack, unless the two participating nodes have some means of authenticating each other's DH protocol messages. Then, the key question we face is: how to authenticate the DH protocol messages without relying on the existence of prior secret or authentic information in both participating nodes?

We use the Guy Fawkes protocol by Anderson et al. [18] to authenticate the DH protocol messages. This protocol uses one-way hash chains. We choose the Guy Fawkes protocol rather than constructs based on asymmetric cryptography, such as digital signatures. Hash chains, which are based on symmetric cryptography, are computationally less expensive to generate. This is an advantage since SAKE has to execute on resource-constrained sensor nodes.

To use the Guy Fawkes protocol, each of the two participants has to "commit" to the other participant's hash chain. That is, each participant needs to obtain at least one authentic value of the other participant's hash chain. The Guy Fawkes protocol does not mention how the participants commit to each other's hash chains. Instead it assumes that an out-of-band mechanism exists for this purpose. In SAKE, we use the ICE primitive to build a mechanism that allows the two participating nodes to commit to each other's hash chains, thereby addressing this shortcoming of the Guy Fawkes protocol.

All the above constructions ensure that a network-based active or passive attacker cannot compromise the confidentiality of the key that is established. The final issue that needs to be addressed is: how can we prevent the attacker that might be present on one of the participants from learning the key? This question is particularly relevant in the case of key re-establishment after node compromise. Even in the case of newly introduced nodes, we cannot be sure that there is no malware present, unless we can trust the distribution channels through which the new nodes are obtained. What we need is a mechanism that prevents the attacker's code, if present on either participating node, from compromising the confidentiality of the key. The ICE primitive provides this mechanism.

### 3.2 Constructing SAKE

In this section, we progressively construct SAKE by first describing how we use the ICE primitive to achieve two goals: one, prevent an attacker that may be present on one of the participating nodes from compromising the confidentiality of the key and two, build a hash chain commitment mechanism for the Guy Fawkes protocol. Then, we describe how we use the Guy Fawkes protocol to authenticate the DH protocol messages, thereby completing our construction of SAKE.

#### Using ICE for SAKE ("SAKE on the Rocks")

We start off with a short overview of ICE. Then, we describe how ICE can be used to prevent the attacker's code, that might be present on either participating node, from compromising the confidentiality of the shared key. After that, we discuss the asymmetry in the ICE primitive that we use as the basis of SAKE's hash chain commitment mechanism. Finally, we describe SAKE's hash chain commitment mechanism.

**ICE overview.** ICE is a challenge-response protocol through which the challenger can verify if an arbitrary piece of code (*target code*), that it invokes on the responder, will execute untampered by any malicious code, that might be present on the responder. In other words, when the ICE protocol succeeds, the challenger obtains an indisputable guarantee that the expected target code executes unmolested by any potential malware present on the responder.

The verification in ICE is done using two checks. First, check that the integrity of the target code is intact before it is invoked for execution. This check ensures that the correct target code image is invoked for execution. Second, check that the execution environment for the target code is set up so that no malicious code will execute concurrently with the target code. Given that the correct target code is invoked for execution, the second check verifies that malicious code will never execute as long as the target code is executing. This prevents the malicious code from attacking the target code during the latter's execution, thereby rendering the malicious code harmless.

ICE is based on a special checksum function that sets up the execution environment for the target code, and then computes a checksum over the memory region containing itself, the target code and the CPU state corresponding to the execution environment. The checksum constitutes a proof of integrity of the target code and the correct instantiation of its execution environment. By verifying the correctness of the checksum it obtains, the challenger can satisfy itself of the indisputable execution of the target code. However, the attacker can try to tamper with the checksum code to forge the correct checksum. To detect such tampering, we construct the checksum function so that its execution time increases on any attempt to tamper with its execution. Therefore, if the challenger obtains the correct checksum within the expected period of time, it obtains the guarantee of indisputable execution of the target code. This overview skips several details of ICE, which can be found in our earlier work [14].

**Achieving key confidentiality using ICE.** The indisputable execution property of ICE offers a direct mechanism to prevent any malware, that may be present on either SAKE participant, from compromising the confidentiality of the key that is established. Each of the two participants acts as the ICE challenger in turn and verifies that the SAKE protocol code will execute untampered by malware on the other participant. If the verification fails, the challenging node refuses to establish a key with the responding node and terminates the SAKE protocol. This ensures that the SAKE protocol is fail-safe. The confidentiality of the shared key is thus ensured, as long as the SAKE protocol is executing.

Further, as we discuss in our earlier work on ICE [14], we can use ICE to undo the attacker's modifications to the sensor node software. By undoing the attacker's modifications after the shared key is established, we can guarantee the confidentiality of the shared key even after the SAKE protocol terminates. This enables SAKE to re-establish shared keys after node compromise.

**Asymmetry in ICE.** We use the asymmetry in the checksum value and the computing time of the ICE checksum, between the genuine checksum function and a modified checksum function, as the basis of the mechanism for hash chain commitment. The key observation here is that if we include the Silicon ID of a node as part of the input used to compute the checksum, only the node with the

Silicon ID will be able to return the the correct ICE checksum within the expected time. In other words, we can use the ICE checksum as a short-lived secret. No other node in the network will be able to compute the correct checksum until sometime after the node with the correct Silicon ID finishes its computation.

The above conclusion assumes that all nodes in the network have the same computing power and that a malicious node does not ask a remote colluding device, with greater computing power, to compute the checksum on its behalf. The first assumption is valid since the attacker does not introduce its own computationally powerful nodes into the network. To justify the second assumption, first note that one of the base stations has to act as the gateway between any malicious node and a remote collaborator. Since the expected time taken to compute the ICE checksum is around 3 seconds [14], by delaying all outgoing packets by 3 seconds, the base stations can ensure that the checksum response computed by a remote collaborator arrives too late at the malicious node.

If the ICE checksum function directly reads the Silicon ID during checksum computation, to incorporate it into the checksum, then there is possibility of the following attack. Since the Silicon ID is a constant, the attacker could place the correct Silicon ID in the memory of a node with a different Silicon ID and use this in-memory copy during checksum computation. Since reading the in-memory copy of the Silicon ID is faster than reading the Silicon ID off the chip, the attacker could actually compute the checksum faster, despite using a node with an incorrect Silicon ID to compute the checksum!

To detect this attack, we design the ICE checksum function so that it reads the Silicon ID into memory before starting checksum computation and refers to the in-memory copy during the checksum computation. Now, we need a mechanism to check whether the correct Silicon ID was read into memory before the checksum computation started. We construct a function, called the *ID check function*, that performs this check. After the checksum function sends the checksum to the challenger, it invokes the ID check function. This function reads the Silicon ID of the node off the chip and compares it against its in-memory copy. If the comparison does not succeed, the ID check function sends an error message to the challenger.

The above approach works because the ICE checksum function computes the checksum over itself, the ID check function, the SAKE protocol code, and the execution environment. Therefore, as long as the challenger obtains the correct ICE checksum within the expected time, it knows that the attacker has not modified the ID check function and cannot tamper with its execution. Then, if the challenger does not receive an error from the ID check function on the responder, it obtains the guarantee that the responder used the correct Silicon ID to compute the ICE checksum.

**Using ICE for hash chain commitment.** Now that we have a mechanism that guarantees that the ICE checksum, computed by the node with the correct Silicon ID will be a short-lived secret, we can use ICE to build the hash chain commitment mechanism for SAKE. Figure 1 shows SAKE's hash chain commitment mechanism based on ICE.

Each of the two nodes participating in the SAKE protocol acts as a ICE challenger, and requests the other node to compute the ICE checksum using the responder's Silicon ID as one of the inputs. After computing the checksum, the responding node uses the computed checksum and a random value to generate its hash chain. Using a random value for generating the hash chain ensures that the attacker will not know the hash chain even if it knows the entire memory contents of the sensor node before the SAKE protocol executes.

The responding node then sends the first element of its hash chain along with a Message Authentication Code (MAC) of this element, computed using the checksum as the key, to the challenger. After sending this packet to the challenger, the responding node invokes the ID check function. If the ID check function finds that the in-memory copy of the responding node's Silicon ID does not match the actual value of its Silicon ID, it sends an error message to the challenger and terminates the SAKE protocol.

If the MAC verifies correctly, the challenger obtains the guarantee that the hash chain element it received was computed by a node with the correct in-memory copy of the Silicon ID. This is the case since a node with an incorrect in-memory copy of the Silicon ID will either return an incorrect checksum or will return the correct checksum outside the expected time window. Further, if the ID check function does not return an error, the challenging node knows that the responding node has the correct hardware Silicon ID as well. Together, these two checks assure the challenger that it has obtained an authentic value of the responder's hash chain. In this manner, the challenger commits to the responder's hash chain.

**Authenticating DH Protocol Messages**

In the previous section, we described how the two nodes participating in SAKE, commit to each other's hash chains. We now describe how we use these hash chains in the Guy Fawkes protocol to authenticate the DH protocol messages. We start off with a brief overview of the Guy Fawkes protocol.

**The Guy Fawkes protocol.** The Guy Fawkes protocol enables two participants that have previously committed to each other's hash chains to exchange authenticated messages. Each message exchanged between the participants is accompanied by the MAC of the message. The sending participant computes the MAC using the next undisclosed element of its hash chain, as the key. Once the receiving participant acknowledges the receipt of the packet containing the message and its MAC value in an authentic manner, the sending participant discloses element of its hash chain that it used to compute the MAC. The receiving participant then authenticates this hash chain element using the a previously obtained authentic hash chain element. The receiving participant then verifies the authenticity of the message by verifying its MAC using the newly received (now authentic) hash element as the key. In this manner, both principals can use their respective hash chains to exchange authenticated messages with each other, provided they can acknowledge each others messages in an authentic manner.

$A \rightarrow B$ : $\langle$ICE Challenge$\rangle$
$A$ : $T_1$ = Current time
$B$ : Compute ICE checksum over memory region
containing the SAKE protocol code,
Check ID function, ICE code, and
in-memory copy of Silicon ID
$C_B$ = ICE checksum
$r_B \xleftarrow{R} \{0,1\}^{128}$
Generate one-way hash chain,
$b_3 = H(C_B \parallel r_B)$,
$b_2 = H(b_3), b_1 = H(b_2), b_0 = H(b_1)$
$B \rightarrow A$ : $\langle b_0, MAC_{C_B}(b_0)\rangle$
$A$ : $T_2$ = Current time
Verify $(T_2 - T_1) \leq$ Time allowed to compute
ICE checksum
Compute MAC of $b_0$ with
ICE checksum computed by A
If MAC of $b_0$ computed by self equals MAC
of $b_0$ sent by B
then B's ICE checksum is correct
$B$ : Invoke ID check function
$B \rightarrow A$ : $\langle$Result of ID check function$\rangle$
$A$ : If ID check function on B sends error, abort
Now B acts as the challenger and obtains the first element
of A's hash chain in a similar manner

**Fig. 1.** Using ICE to construct the hash chain commitment mechanism for SAKE. $A$ and $B$ are two neighboring nodes. This commitment mechanism enables $A$ and $B$ to commit to each other's hash chains without requiring any prior secret or authentic information, and without the existence of secure side channels. $A$ and $B$ can then use their respective hash chains in the Guy Fawkes protocol to authenticate the DH protocol messages.

The Guy Fawkes protocol can be used for sending authenticated acknowledgements as well. The receiving participant simply sends the next undisclosed element of its hash chain to acknowledge the receipt of a message from sending participant. The sending participant verifies the authenticity of the receiving participant's acknowledgement by authenticating the hash chain element it receives, using a previously received authentic hash chain element. If the authentication is successful, then the sending participant knows that the receiving participant has correctly received the message, and therefore, it is now safe to release the hash chain element that the sending participant used to compute the MAC of the message. If the sending participant were to disclose the hash element without requiring an authenticated acknowledgement, then the attacker could suppress the sending participant's packet before it reaches the receiving participant, and later on use the disclosed hash chain element to generate fake messages with the correct MAC value.

As can be seen from the above description, the Guy Fawkes protocol proceeds in a lock-step manner between the two participants, with each of them successively disclosing the next element of their respective hash chains. The sending participant first sends the message along with the MAC. The receiving participant then discloses the next undisclosed element of its hash chain by way of acknowledging the message. Finally, the sending participant discloses the element of its hash chain that it used to compute the MAC of the message.

**Using the Guy Fawkes protocol.** We now describe how SAKE uses the Guy Fawkes protocol to authenticate the DH protocol messages.

Figure 1 shows how the two nodes participating in the SAKE protocol commit to each other's hash chain, which is a requirement for using the Guy Fawkes protocol. Figure 2 shows how the two nodes now use the Guy Fawkes protocol to authenticate the DH protocol messages. Once both nodes have the authentic first element of each other's hash chain, they send acknowledgements to each other indicating that they have successfully committed to each other's hash chains and are now ready to perform a DH key exchange. After verifying the other participant's acknowledgement, each of node generates its DH half-key. The nodes then exchange their respective half keys and authenticate this exchange using the Guy Fawkes protocol. Finally both nodes compute their shared secret using the half keys they receive.

### 3.3 Cryptographic Primitive Selection and Evaluation

In this section, we first discuss how we select the cryptographic primitives for use in SAKE, that are suitable for resource-constrained sensor nodes. Then, we present our evaluation that shows the time taken to execute SAKE and energy consumed.

**Selection of cryptographic primitives.** We have designed SAKE for sensor nodes with limited computing resources. Hence, the choice of the cryptographic primitives requires careful thought. To save program memory, we suggest code reuse by using a block cipher to implement all required cryptographic primitives. We suggest the use of RC5 [19] as the block cipher because of its small code size and efficiency. In a prior work in sensor network security [20], Perrig et al. stated that an optimized RC5 algorithm can encrypt an eight byte block in 120 cycles on an 8-bit microcontroller. A MAC function can be obtained using the CBC-MAC construction. A hash function can also be constructed with a block cipher as follows: $h(x) = C(x) \oplus x$, where $C(x)$ stands for the encryption of $x$ using the block cipher. The encryption uses a publically known value as the key. Both the MAC function and the hash function are constructed out of the RC5 block cipher with 128 bit blocks, 8 rounds, and 128 bit keys.

**Diffie-Hellman parameters** Generally, it is considered impractical to perform expensive asymmetric cryptographic operations on sensor nodes because of re-

$A \rightarrow B:\ \langle a_1 \rangle$

$B \rightarrow A:\ \langle b_1 \rangle$

$A:$ Verify $b_0 = H(b_1)$

$B:$ Verify $a_0 = H(a_1)$

With the above acknowledgements A and B indicate that they are now ready to perform the DH key exchange protocol

$A:$ $x \xleftarrow{R} \{0,1\}^{112}$

Compute $(g^x)\ mod\ p$

$A \rightarrow B:\ \langle (g^x)\ mod\ p, MAC_{a_2}((g^x)\ mod\ p) \rangle$

$B \rightarrow A:\ \langle b_2 \rangle$

$A:$ Verify $b_1 = H(b_2)$

$A \rightarrow B:\ \langle a_2 \rangle$

$B:$ Verify MAC of $(g^x)\ mod\ p$ using $a_2$

$B \rightarrow A:\ \langle b_3 \rangle$

$A:$ Verify $b_2 = H(b_3)$

At this point A knows that B has obtained A's authentic DH half-key

Now B sends its DH half key $(g^y)\ mod\ p$ to A in a similar manner

$A:$ Compute $(g^y mod\ p)^x mod\ p$

$B:$ Compute $(g^x mod\ p)^y mod\ p$

**Fig. 2.** Using the Guy Fawkes protocol to authenticate DH protocol messages. $A$ and $B$ are two neighboring nodes. This protocol uses the hash chains that $A$ and $B$ committed to in the protocol shown in Figure 1.

source constraints. In our work, by carefully picking the parameters, it is possible to carry out the DH key exchange protocol on the Telos motes.

The DH key exchange protocol uses an exponentiation operation given by $g^x$ mod $p$. The security of the DH key exchange protocol is based on the length of the exponent $x$ and the length of the prime $p$. For SAKE, we use 512 bit modulus $p$ and 112 bit exponent $x$. According to Lenstra and Verheul, these parameters are deemed to be secure in the year 1990 considering the state of the art technology at that time [21]. Since we are dealing with low cost, mass produced sensor nodes, 1990 levels of security are sufficient. Of course, the attacker can break the keys of the sensor nodes with considerable effort by employing technology that is state-of-the-art. Greater security can be obtained by increasing the sizes of $x$ and $p$. The cost, though, is the computation time and energy. Since $g$ is unimportant for the security of the DH key exchange protocol, we set $g$ to be two in order to speed up computation. Using these parameters, the Telos motes were able to perform $g^x \mod p$ in 6.19 seconds.

## 4 Related Work

In this section, we review related work in the area of sensor network key establishment.

Many researchers have considered key establishment protocols, however, all these efforts assume the presence of prior secret information to prevent man-in-the-middle attacks [3, 5, 2, 4, 20, 22].

There are very few key establishment protocols that prevent man-in-the-middle attacks without assuming the presence of authentic or secret information. Some of these protocols require human-mediation or special hardware [8–10, 12]. SAKE requires neither but does assume that the attacker does not introduce its own computationally-powerful nodes into the network. In this manner, SAKE is an interesting point the key establishment space, offering a unique trade-off between security and ease-of-use.

Integrity-code [7] attempts to perform key establishment without prior secrets. In this scheme, the presence of an RF signal represents a binary '1', while its absence represents a '0'. Assuming an attacker cannot remove radio energy, an attacker would only be able to modify messages by changing a '0' into a '1', but not the other way around. A carefully selected encoding scheme renders it impossible for an attacker to modify the encoded messages. This ensures message authenticity. However, the drawback to this approach is that the threshold energy value that differentiates a '0' from a '1' needs to be known to the protocol participants in an authentic manner. When posed with this question, the authors declined to specify such a method.

Like SAKE, Key Infection [11] also performs key establishment without assuming prior secret or authentic information. However, since the key is initially transmitted in the clear, the scheme is insecure.

SAKE is the only protocol we are aware of that can also perform key reestablishment after node compromise. This is the case since the presence of the attacker's code in the memory of the protocol participants does not affect the security of SAKE.

## 5 Conclusion

We present SAKE, a new protocol for key establishment in sensor networks. SAKE's unique feature is that it can establish new keys without requiring any secret values in any node, yet, active or passive attacking sensor nodes cannot perform man-in-the-middle attacks, as long as the attacker is remote and does not insert its own computationally more powerful nodes into the network. The key idea behind our approach is to set up a short-lived shared secret between the protocol participants through the use of the ICE primitive [14].

The main application domains we envision for SAKE are sensor networks that are deployed in physically secure environment, such as in nuclear power plants, financial institutions, military operations, or other critical infrastructures. In these domains, SAKE provides a powerful primitive to establish keys upon network setup and to re-establish secret keys after a remote attacker has compromised nodes, without requiring any human intervention. Hence, SAKE represents a useful new primitive in the key establishment design space, offering a unique trade-off between security and ease-of-use.

## References


1. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.D.: SPINS: Security protocols for sensor networks. Wireless Networks **8**(5) (September 2002) 521–534
2. Eschenauer, L., Gligor, V.: A key-management scheme for distributed sensor networks. In: Proceedings of Conference on Computer and Communication Security. (November 2002) 41–47
3. Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: IEEE Symposium on Security and Privacy. (May 2003)
4. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: Proceedings of ACM Conference on Computer and Communications Security (CCS). (October 2003) 52–61
5. Du, W., Deng, J., Han, Y., Varshney, P.: A pairwise key pre-distribution scheme for wireless sensor networks. In: Proceedings of ACM Conference on Computer and Communications Security (CCS). (October 2003) 42–51
6. Karlof, C., Sastry, N., Wagner, D.: TinySec: A link layer security architecture for wireless sensor networks. In: Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys). (November 2004)
7. Cagalj, M., Capkun, S., Rengaswamy, R., Tsigkogiannis, I., Srivastava, M., Hubaux, J.P.: Integrity (I) codes: Message integrity protection and authentication over insecure channels. In: IEEE Symposium on Security and Privacy. (May 2006)
8. Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., Gellersen, H.W.: Smart-its friends: A technique for users to easily establish connections between smart artefacts. In: Proceedings of Ubicomp. (2001)
9. Lester, J., Hannaford, B., Borriello, G.: Are you with me? Using accelerometers to determine if two devices are carried by the same person. In: Proceedings of Pervasive. (2004)
10. Castelluccia, C., Mutaf, P.: Shake them up! a movement-based pairing protocol for cpu-constrained devices. In: Proceedings of ACM/Usenix Mobisys. (2005)
11. Anderson, R., Chan, H., Perrig, A.: Key infection: Smart trust for smart dust. In: Proceedings of IEEE Conference on Network Protocols (ICNP). (October 2004)
12. Kuo, C., Luk, M., Negi, R., Perrig, A.: Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes. Proceedings of the ACM Conference on Embedded Networked Sensor System (SenSys) 2007 (2007)
13. Rasmussen, K., Capkun, S.: Implications of radio fingerprinting on the security of sensor networks. In: Proceedings of the Third International Conference on Security and Privacy for Communication Networks (SecureComm). (September 2007)
14. Seshadri, A., Luk, M., Perrig, A., van Doorn, L., Khosla, P.: SCUBA: Secure code update by attestation in sensor networks. In: ACM Workshop on Wireless Security (WiSe). (September 2006)
15. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: Architectural Support for Programming Languages and Operating Systems. (2000) 93–104

16. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling ultra-low power wireless research. In: Proceedings of International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS). (April 2005)
17. Wan, C.Y., Campbell, A.T., Krishnamurthy, L.: PSFQ: A reliable transport protocol for wireless sensor networks. In: Proceedings of ACM Workshop on Wireless Sensor Networks and Applications (WSNA). (September 2002)
18. Anderson, R., Bergadano, F., Crispo, B., Lee, J., Manifavas, C., Needham, R.: A new family of authentication protocols. ACM Operating Systems Review **32**(4) (October 1998) 9–20
19. Rivest, R.: The RC5 encryption algorithm. In: Proceedings of Workshop on Fast Software Encryption. (1994) 86–96
20. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.D.: SPINS: Security protocols for sensor networks. In: Proceedings of Conference on Mobile Computing and Networks (Mobicom). (July 2001)
21. Lenstra, A., Verheul, E.: Selecting cryptographic key sizes. In: Journal of Cryptology: The Journal of the International Association for Cryptologic Research. (1999)
22. Zhu, S., Setia, S., Jajodia, S.: LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In: Proceedings of ACM Conference on Computer and Communications Security (CCS). (October 2003)