# SASHA: A Distributed Protocol for Secure Application Deployment in Shared Ad-hoc Wireless Sensor Networks

Jef Maerien, Sam Michiels, Christophe Huygens, Wouter Joosen
*IBBT-DistriNet*
*Department of Computer Science*
*Katholieke Universiteit Leuven*
*B-3001, Leuven, Belgium*
*Email: firstname.lastname@cs.kuleuven.be*

***Abstract*—Wireless ad-hoc sensor networks in industrial settings often consist of multiple independent parties, each owning a subset of the nodes. In order to reduce costs, minimize time to market and increase coverage and functionality, these parties must share the capabilities of their individual sensor nodes; this creates a multi-owner and multi-application environment that requires advanced and secure mechanisms to control the deployment and operation of sensor applications. Although there is clear demand like in transport & logistics, state-of-the-art on secure application deployment in WSNs is lacking support for this sharing of sensor nodes. This paper presents SASHA: a proof-of-concept protocol that enables lightweight and secure deployment of multiple applications on heterogeneously owned sensor nodes.**

***Keywords*-Code Distribution, Wireless Sensor Networks, Security, Shared Infrastructure**

## I. Introduction

Wireless Sensor Networks (WSNs) have many applications in a wide array of fields such as logistics, healthcare and agriculture. As more real-world applications are being developed and deployed, new use cases and challenges are being discovered. WSNs are moving away from static single-owner single-application platforms toward dynamic multi-owner multi-application environments where long-lived mobile nodes form ad-hoc networks and adapt based on evolving requirements. Many platform owners (POs) each own part of the network and want to share their sensor node capabilities [5] in order to reduce cost or to increase coverage and functionality of the network. The application owners (AOs) want to deploy applications on these shared networks, while not necessarily owning part of the network. Clearly security is a primary concern in this multi-actor setting as requirements such as node integrity, application confidentiality, and resource control need to be maintained.

Much research has been done on secure management of WSNs. Several protocols have been proposed to deploy code updates efficiently and securely. However, most of this research assumes a static sensor network where a single owner has total control of both objectives and platform.

The security challenge tackled in this paper is to enable an AO to securely install a distributed application across a WSN made up of nodes that are owned by many different POs. The AO must be able to deploy applications on the WSN while the POs must be certain that only approved applications can be installed and that the capabilities and lifetime of these applications are limited.

The contribution of this paper is the definition and realisation of SASHA: a light weight distributed protocol for Secure Application deployment in SHared Ad-hoc wireless sensor networks, which provides this necessary multi-actor support by using node specific tokens. These tokens also contain deployment parameters and runtime limitations for the deployed application.

The remainder of this paper is structured as follows: Section II details the use case this paper employs and lists the functional and security requirements. Section III gives a brief overview of related work. Section IV proposes the SASHA protocol. Section V evaluates the prototype implementation. Section VI summarizes the main contributions of this paper and highlights future work.

## II. Context and requirements

### A. Use Case

This section presents a case study in logistics. The two main actors present are (1) the logistics providers and (2) the infrastructure provider as shown in figure 1.

A logistics provider (LP) transports goods using containers. Some goods, such as pharmaceuticals and electronics, are valuable and sensitive to damage. The LP wants to monitor these goods to ensure that the conditions of the transport are within limits. Thus he equips his containers with sensor nodes and runs a monitoring application. He is both a platform owner (PO) and an application owner (AO). Depending on the type of transport, the LP might install a different application that checks different variables.

In a harbor or on a ship multiple LPs are present who each have their own containers and sensor nodes. The LPs need to work together to form an ad-hoc wireless network so messages of all nodes can be routed toward the gateway. Further cooperation is desired in order to provide additional

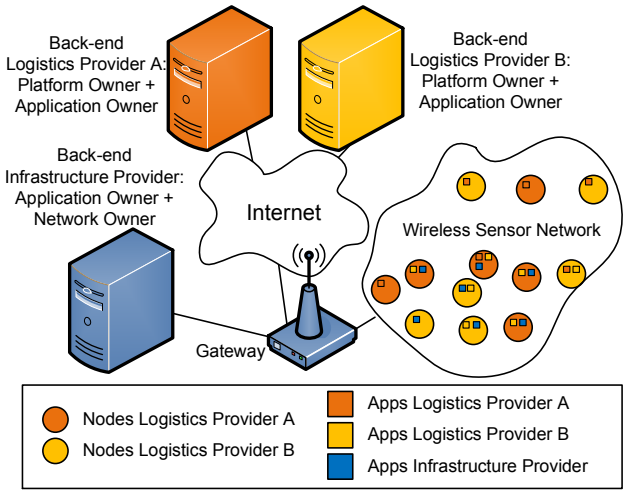


Figure 1. Network representation of the logistics use case.

functions for example to check whether reactive chemicals are stored at a safe distance apart.

The infrastructure provider (IP) provides the infrastructure for the containers and Internet gateways for the nodes. Some examples are the operators of harbors, ships and warehouses. The IP also wants to monitor container attributes while present on his site. He can either deploy a costly private sensor network or reuse the devices of the LPs. This paper assumes that the LPs present on the site of the IP allow the IP limited use of their sensor nodes. The IP is another example of an AO.

In the above scenario security is clearly of vital importance. LPs want that only trusted parties can install verified applications onto the sensor nodes and that the resource usage and lifetime of these applications is limited. An application deployed by an IP should only remain on the node as long as a node is located in the site of the IP. Once it leaves the site, the application should be removed.

### B. Requirements

This section lists the five main requirements of the secure deployment protocol.

**Light weight:** Any protocol in WSNs must be light weight since nodes only have a limited amount of energy, processing and communication resources. This is especially important in logistics since containers are often far away from their owners.

**Per node permissions:** The PO wants to have a clear view of which applications run on each node. By giving permission on a per use and per node basis, the PO keeps the overview of the current state of his sensor nodes.

**Multiple nodes:** Applications must be deployed onto multiple sensor nodes of different companies. If each PO wants to deploy the application onto his sensor nodes separately, the application would have to be sent over the network several times, causing significant communication overhead. Transmitting the application to all relevant sensor nodes at once brings less overhead and is thus preferred.

**Resource usage limits:** The PO needs to be able to deploy and enforce some limitations on the resource consumption of the applications of the sensor nodes, for example restricting the number of times an application runs and the frequency and amount of data it sends.

**Lifetime limitations:** In ad-hoc network scenarios applications should only live as long as a node is a part of the network or only for a limited amount of time.

### C. Attacker model

This paper considers 2 types of attackers.

**The external attacker** is a malicious outsider. He can intercept, modify and inject messages into the WSN. This paper considers 2 types of attacks: (1) a DOS attack, where the attacker attempts to use the deployment protocol to force the node in spending unnecessary energy, and (2) a man in the middle attack where an attacker intercepts and alters the communication between the AO and the node.

**The internal attacker** is a malicious application owner. He has been granted the right to deploy an application onto a sensor node. He wants to exploit this permission to deploy other applications, or redeploy the same application after it has been removed.

### D. Security requirements

This section lists the security requirements for the secure deployment of applications based on the attacker model.

**Confidentiality:** The AO must be able to confidentially deploy an application. Since this requires additional bandwidth and computing power, this should be optional.

**Authenticity:** The node must be able to verify that the permission for deployment is created by its PO.

**Integrity:** The node must be able to verify that the application it received is the application that the PO approved.

**Freshness:** The approval of the deployment should only remain valid for a certain amount of time.

**Limited use:** A permission to deploy an application should only apply to one application for single use.

**Survivability:** Due to a lack of physical security in WSNs, an external attacker can break open and probe a node, revealing all the key material contained in the node. The consequences of such node capture must be minimal.

## III. Related Work

This section evaluates and compares the related work in the area of secure code dissemination protocols and secure service access protocols (see Table I).

Many protocols have proposed security measures for the Deluge code dissemination protocol [4]. Deluge is delivered as a part of the TinyOS operating system, making it the de facto programming standard. Most propose a variation of

| | SASHA | Sluice | Tan2009 | Kerberos |
|---|---|---|---|---|
| Light weight | + | + | + | - |
| Multi-party support | + | - | - | + |
| Confidentiality | + | - | + | + |
| Service parameters | + | - | - | + |
| Temporary lifetime | + | - | - | + |

Table I
COMPARISON OF THE FEATURES OF SASHA, KERBEROS AND A SELECTION OF DEPLOYMENT PROTOCOLS FROM THE RELATED WORK.

a series of hash chains or hash trees where the start of the chain is signed with the base-station's private key to provide integrity and authenticity. Some examples are Sluice [7] and Seluge [6]. Tan et al. [9] also proposed to use the first L bytes of the hash to encrypt the binary to provide confidentiality.

While these works address the problem of securing the application deployment, none of these offer multi-party support. This paper identifies 4 points that are missing: (1) support for multiple parties deploying applications, (2) optional support for confidentiality, (3) resource consumption limitations, and (4) time-limited deployment.

SASHA is inspired by the Kerberos authentication scheme [8]. In Kerberos, the requester of a service first logs into an Authentication Service and receives a Ticket Granting Ticket (TGT). When the user wants to access a service, he contacts the Ticket Granting Server (TGS) using his TGT. The TGS then returns a service token with which the user can interact with the service. While Kerberos provides secure service access, it is too heavy for usage in WSN due to the ticket size: hundreds to even thousand of bytes.

## IV. PROTOCOL

This section presents the secure code deployment protocol. SASHA identifies three roles: the Network Operator, the Platform Owner and the Application Owner.

The Network Operator (NO) operates the network: he ensures the security of communication between nodes. He also provides a gateway for nodes to communicate with the Internet and a registry which lists the nodes currently in the network.

The Platform Owner (PO) offers the resources of his nodes to be used by the AOs.

The Application Owner (AO) wants to deploy distributed applications onto the WSN.

SASHA is based on the following assumptions:

- Each node has a symmetric encryption algorithm and a hashing algorithm installed.
- Each node shares a unique secret symmetric key with its owner. Since each node starts in the physical possession of the owner, the owner can securely deploy this key.
- The SASHA protocol assumes a component based architecture for the applications running in the WSN.

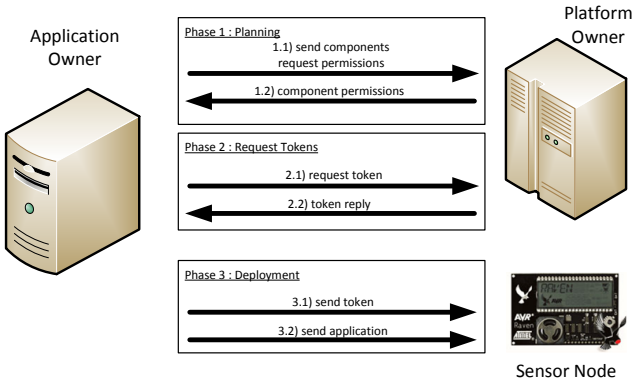


Figure 2. Overview of the SASHA application deployment protocol.

- When a node enters a network, it registers itself with the NO.
- A component uses the resources of a platform. This paper divides these resources into two kinds: energy consuming resources and data resources.
- Energy consuming resources are those resources which cause significant energy usage such as the amount of CPU time a component uses, and the rate and frequency with which it sends. This paper assumes that the node middleware allows for parametrization of these resources.
- Data resources provide access to live data or data logs. Not all components should have access to all data resources. This paper assumes that a node cannot perform the access control because it does not provide memory protection but it can be enforced by code verification on a back-end server.

The SASHA deployment protocol exists out of 3 phases as shown in figure 2 and discussed in the following sections : (1) planning the deployment, (2) requesting the tokens, and (3) enacting the deployment.

### A. Phase 1: Planning the deployment

Multiple applications are active in a WSN at the same time. In order to efficiently use the network, it is important that the deployment of these applications is carefully planned. In a multi-owner environment, this planning has an additional layer of complexity due to the limitations that are enforced by the different POs.

The planning phase can be divided into three steps: (1) The AO sends a request to each PO in the WSN stating which components he wants to deploy. He can acquire the identities of the POs in the network from the NO. (2) The POs analyse the components and send back a contract stating which nodes can run which components with which parameters. (3) The AO plans the actual deployment given the limitations received by the different POs.

### B. Phase 2: Requesting the deployment

Once the deployment plan is created, the AO has to request and receive the tokens necessary to deploy the components on the nodes. The POs have to provide this token request service.

This phase has two steps: the request and the reply step.

**Step 1: The AO sends a deployment request.** The AO sends a request to the POs stating which components the AO wants to deploy on which nodes and with which parameters. The parameters currently supported are: (1) must the component be sent confidential, (2) the maximum component transmission frequency, (3) the maximum send rate, (4) the wake-up interval, (5) delete component when leaving the network, and (6) delete component after a certain time.

It is clear that the AO has to contact the PO securely. The SASHA protocol uses an SSL connection with mutual authentication to ensure the confidentiality and authenticity of the communication between the AO and the POs.

**Step 2: The PO sends the deployment reply.** The PO checks whether or not the AO has received permission to deploy the component with the requested parameters. If the component or parameters do not match the contract agreed in phase 1, the deployment fails.

Because each component has to be approved by the PO and has clearly defined parameters, the PO can create an accurate view of the load on his nodes and have a good estimate of their lifetime.

If the AO is allowed to deploy the component on the requested node, the PO generates a token which the AO can use to access the component deployment service. The token consists out of the following fields (see figure 3):

- Application Hash - 32B: SHA-2 hash of the component.
- TokenId - 4B: A unique id for each token.
- Timestamp - 4B: From when the token is valid.
- Timeout - 2B: Lifetime of the token.
- ComponentLength - 2B: Length of the component.
- Flags - 1B: A list of the following binary flags:
  - Confidential: Is the component sent confidential.
  - DeleteOnTimeOut: Delete the component once the token times out.
  - DeleteOnLeaving: Delete the component once the nodes leaves the current network.
  - 5 flags are currently unused.
- SendFrequencyLimit - 1B: Number of messages a component can send in an hour.
- SendDataLimit - 1B: Rate with which a component can send data.
- WakeupTime - 1B: Time between when a component wake-ups.
- Deployment Key - optional - 16B: Deployment specific AES-128 encryption key.

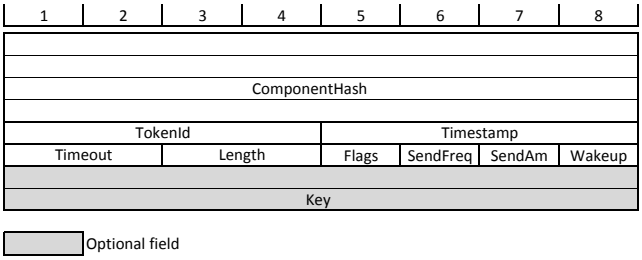


Figure 3. Representation of the token byte format.

The token is 48 (not confidential) or 64 bytes (confidential). It is encrypted with the symmetric key known only to the specific node and his PO. This ensures only the PO and the node can read or modify the token. The PO returns the token to the requesting AO.

If the AO wants to deploy the same component to multiple nodes, he must request a unique token for each node. This process however requires only Internet communication that does not burden the WSN.

### C. Phase 3: Enacting the deployment

The third and last phase is enacting the deployment. This phase contains 3 steps: (1) sending the token, (2) sending the component and (3) installing the component.

**Step 1: AO sends tokens to nodes.** When the AO is ready to deploy (when he has all the necessary tokens), he sends the tokens to the nodes. Since each node requires a unique token, it has to be sent to each node directly. Each node then verifies that the token is valid by decrypting it and checking the timestamp, timeout and tokenId field. Once the node verified the token, it saves the deployment parameters and the hash. If the token shows that the binary is confidential, it decrypts the deployment key and prepares to decrypt the binary. Finally, it listens to further communication to receive the component it needs to install.

**Step 2: AO sends component to nodes.** Each node receives the component, decrypts if needed and checks the hash. The security protocol is independent of the exact means of code distribution: the component can be multicasted over the network, forwarded from one node to the next or send directly to each node.

At this time, the node calculates the hash over the entire binary. The protocol could work as proposed in previous research and use a per page hash. The token would then contain the first hash of the hash chain, encrypted with the secret key. However this paper focuses on securing the code deployment process in a multi-actor environment, rather than optimizing the code distribution process.

**Step 3: Nodes install component** Once the entire component is received and the hash has been verified, the nodes can install the component. The token is saved in a token store for as long as the token is valid.

### D. Removing deployed components

Depending on the Delete flags of the token, the node will remove the components either when the node leaves the current network or when the timeout value has been reached. It is of course possible to not set these flags, resulting in component removal only when a remove command is sent to the node.

## V. Evaluation

This section evaluates the implementation and revisits the requirements.

### A. Implementation

We implemented the protocol on AVR Ravens [1] running the Contiki OS [2] and the LooCI middleware [3]. Contiki is a modular and light weight OS. LooCI is a component based middleware platform and allows for easy development and deployment of new application components. Since neither Contiki nor LooCI support multi-casting or hop by hop code deployment, the current implementation works by sending each component directly to the node over a TCP connection. The planning phase and the enforcement of the limitations are currently not implemented.

SASHA uses the AES-128 for encryption and SHA-2 for hashing. These algorithms are chosen for their security and standardisation.

This section evaluates the implementation and compares it with Sluice [7] (see table II). A comparison with the Tan2009 protocol cannot be made due to absence of implementation details.

**Communication overhead:** The only extra communication overhead is the token that has to be transmitted before the actual code deployment. The token has a size of 48 to 64 bytes, which is sufficiently small for use in WSNs. The token used by Sluice has a similar size of 44 bytes.

**ROM overhead:** The binary for the AVR raven is 68.6kB large and contains 4 parts: (1) Contiki OS: 40kB, (2) LooCI platform: 21.4kB, (3) crypto libraries: AES 3kB - SHA-2 1.8kB, and (4) deployment protocol: 2.3kB. The total protocol overhead is ca 7.1kB. A significant part of the overhead is caused by the implementation of the crypto libraries. Sluice has a comparable ROM overhead of 9kB.

**RAM overhead:** Only the additional RAM overhead of using the secure deployment protocol instead of the unsecured protocol is considered. The protocol requires: (a) buffers for AES (200B) and SHA-2 (40B), (b) buffers to store the token and the secret key (100B), (c) list of recently used tokens (84B), (d) deployment buffers (80B).

This sums up to a total additional RAM overhead of ca 510B over an unsecured deployment protocol. This is comparable to Sluice which has a RAM overhead of 2kB.

**Delay:** The additional deployment delay is minimal. The ticket request is done over the wired Internet between servers, so this poses no WSN overhead. The SASHA protocol further requires the additional transmission of a message of 48B. Since most applications are several hundreds to thousands of bytes, the increase in time is small. No measurements were performed of these delays. Note however that SASHA requires a unique token for each node, which significantly increases overhead if each node requires the component. However this paper argues that many use cases require that components are only deployed on a very limited subset of nodes, thus not requiring network wide flooding.

| | SASHA | Sluice |
|---|---|---|
| Comm overhead (bytes) | 48 | 44 |
| ROM (kB) | 28 | 9 |
| RAM (bytes) | 510 | 2000 |
| Computational delay (s) | 0.160 | 30 |

Table II
Comparison of the proposed protocol with Sluice.

Next the time overhead of the computation on the node is considered. The AES decryption operation of one 64B block takes 3.5ms. Adding a 64B block to the total hash takes 6.5 ms. A small component update of 1kB takes 1024/64*(3.5+6.5) ms = 160ms. In comparison, Sluice takes 30 to 35 seconds to verify a program due to the ECDSA verification.

The PO's server (which manages and grants the tokens) and the AO's server (which requests the tokens and deploys the applications) are both implemented in Java. The planning service is currently unimplemented.

### B. Discussion

This section briefly discusses the different security aspects of the deployment protocol.

**Confidentiality:** By setting the confidentiality flag in the token and adding a deployment key, a component can be confidentially deployed. If this is not necessary the flag can be set to off and the additional resources can be saved.

**Authenticity:** A node can verify that the PO created the token for the specific node since the node has to decrypt the token with a symmetric key that is unique for each sensor node and only known by the PO and the node.

**Integrity:** Only a component with the same hash as the PO approved and stated in the token can be installed. Assuming that SHA-2 is collision resistant, only the exact component that the PO approved can be installed.

**Freshness:** Because the token contains a timestamp and a timeout, the sensor can verify that the token is valid. This requires a level of synchronisation, which is likely also needed for other applications such as logging.

**Limited use:** Every token contains a tokenId. Since the node verifies that a tokenId has not been used before, the token can only be used once. This requires that the sensor node keeps a list of tokens that have recently been used.

However, it is only needed to keep the tokenId as long as the timeout has not passed.

**Survivability:** If a malicious party manages to capture one node, he only knows the key for that particular node. He cannot influence any other node using that key.

### C. Discussion of potential attacks

This section briefly discusses potential attacks on the protocol and how the protocol can counter these attacks.

**Denial of service attacks:** The protocol can undergo denial of service attacks by an attacker sending invalid tokens or hijacking a session and sending more data than required. The main operation to verify a token only requires one symmetrical decryption operation. Because the component length is stated in the token, the node knows how many bytes it should receive, and closes the connection after it has received the expected amount of bytes.

**Man in the middle attacks:** Entities in the network can try to alter the token or the binary to install malicious components. Modifications in the token will be detected through invalid tokenIds and timestamps. Modifications in the binary will be detected when the hash is calculated and a mismatch occurs. The deployment process will fail, but node integrity remains assured.

## VI. Future work and conclusion

### A. Future work

This paper presented a basic protocol for the secure deployment of applications onto a shared WSN. We envision two extensions: (1) extend the enforcement infrastructure: add more parameters to the deployment token such as RAM usage, file system usage or sensor usage, and (2) research the first phase of the deployment protocol: planning a deployment given a heterogeneous set of owners each with their own associated policies, parameters and costs.

### B. Conclusion

It is clear that WSNs are moving toward multi-party multi-application ad-hoc environments where several parties have a need to share their sensor node infrastructure. In order to share the infrastructure, it needs to be possible to deploy new applications on sensor nodes owned by others. Naturally, in such use cases, security is of vital importance.

This paper presents SASHA: a light weight distributed protocol for the secure deployment of applications on a shared multi-party WSN. The protocol requires an Application Owner (AO) to request permission from all the Platform Owners (POs) on whose infrastructure he wishes to install an application. Each PO validates this request and, if approved, grants a token with which the AO can use to deploy his application. This token also contains parameters to limit the amount and frequency a component can send and the CPU time it can use.

We developed a prototype using the AVR Raven and the LooCI middleware. The prototype demonstrates that the overhead of this protocol is limited and comparable to other protocols, whilst offering multi-actor support and supporting resource consumption policies.

## Acknowledgements

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT and the Research Fund K.U. Leuven. This work was conducted in the context of the ITEA2 "Do-it-Yourself Smart Experiences" project, ITEA2 08005, and is supported by funding from the Flemish agency for Innovation by Science and Technology (IWT).

## References

[1] Atmel. Avr raven, sep 2010. Available as http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=4291.

[2] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[3] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. D. Cid, S. Michiels, C. Huygens, W. Joosen, and J. Ueyama. Building wireless sensor network applications with looci. In *International Journal of Mobile Computing and Multimedia Communications vol:2 issue:4*, pages 38–64, Hershey, PA 17033, USA, Oct 2010. IGI Global.

[4] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY, USA, 2004. ACM.

[5] C. Huygens and W. Joosen. Federated and shared use of sensor networks through security middleware. In *ITNG '09: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 1005–1011, Washington, DC, USA, 2009. IEEE Computer Society.

[6] S. Hyun, P. Ning, A. Liu, and W. Du. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pages 445 –456, Apr. 2008.

[7] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 53, Washington, DC, USA, 2006. IEEE Computer Society.

[8] B. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33 –38, sep 1994.

[9] H. Tan, D. Ostry, J. Zic, and S. Jha. A confidential and dos-resistant multi-hop code dissemination protocol for wireless sensor networks. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 245–252, New York, NY, USA, 2009. ACM.