Designing A Secure Reliable File System for Sensor Networks

Neerja Bhatnagar
Storage Systems Research Center
University of California, Santa Cruz
neerja@cs.ucsc.edu

Ethan L. Miller
Storage Systems Research Center
University of California, Santa Cruz
elm@cs.ucsc.edu

ABSTRACT

Wireless sensor networks are increasingly being used to monitor habitats, analyze traffic patterns, study troop movements, and gather data for reconnaissance and surveillance missions. Many wireless sensor networks require the protection of their data from unauthorized access and malicious tampering, motivating the need for a secure and reliable file system for sensor nodes. The file system presented in this paper encrypts data stored on sensor nodes' local storage in such a way that an intruder who compromises a sensor node cannot read it, and backs it up regularly on to its neighbor nodes. The file system utilizes algebraic signatures to detect data tampering.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems management—*Distributed file systems*; D.4.6 [**Operating Systems**]: Security and Protection—*Authentication, Cryptographic controls*

General Terms

security, reliability

Keywords

secure, reliable, sensor network file system

1. INTRODUCTION

A wireless sensor network is composed of sensor nodes that have extremely constrained resources in terms of memory, computation, storage, and energy. Most sensor nodes follow a sleep-awake-sense-sleep duty cycle, and are typically up only about 1% of the time. Wireless sensor networks are increasingly being used for various applications, such as enemy reconnaissance, battlefield surveillance, studying troop movements, monitoring habitats, analyzing traffic patterns, and for observing the structural movement and integrity of buildings and bridges for purposes such as seismic retrofitting and general reinforcement. In many of these applications, it is important that the observations recorded by sensor nodes are protected from unauthorized access and tampering. It is also important for these applications to ensure the persistence of these observations despite individual node failures. To address this need, we are developing a secure and reliable file system for sensor nodes. This file system encrypts stored data in order to prevent unauthorized data access; even the sensor node itself cannot read the data once it has been stored. The file system uses algebraic signatures to detect and prevent tampering. Each sensor node backs up its data by dividing it into small chunks and distributing these chunks to its *back-up buddies*—a combination of its one-hop and multi-hop neighbors that help back-up its data. Periodically, each originating sensor node checks to ensure that its back-up buddies are storing the chunks designated to them correctly. By encrypting data, backing it up on to other nodes, and periodically checking its integrity, the sensor network can ensure that the data it has recorded remains both secure and available despite the compromise of individual nodes.

The rest of this paper is organized as follows: Section 2 discusses the factors that motivated the development of the file system, and the file system's usage scenarios; Section 3 presents the threat model; Section 4 presents the design of the reliable and secure file system, including its advantages and disadvantages; Section 5 compares and contrasts the file system presented in this paper with other research; and Section 6 presents conclusions and future work.

2. MOTIVATION

Several factors, including energy efficiency in storage, availability of high-capacity storage, inability to transmit to base station, and increased deployment of sensor networks in specialized and hostile environments, motivate the design of a secure and reliable file system for sensor nodes. These factors are discussed in detail in the following paragraphs.

Storage Efficiency. Transmitting data over radio consumes 200 times more energy than storing the same amount of data locally on a sensor node, and radio reception uses 500 times more energy than reading the same amount of data from local storage [15]. Moreover, according to Moore's law, the cost of storing data on to a flash device will continue to decrease [4]. This is in contrast to a minimal amount of energy, dictated by physics, that will always be required for

StorageSS'07, October 29, 2007, Alexandria, Virginia, USA.

radio operations. These statistics, coupled with the availability of gigabyte-scale storage on sensor nodes promote the "sense and store" [17] paradigm in sensor networks, as opposed to "sense and relay". A secure and reliable file system will prove to be valuable in "sense and store" sensor networks because it will enable the secure, long-term persistence of data.

Inability to Transmit to Base Station. Sensor nodes may not always be able to transmit their observations to a base station because the base station may not be available due to failure or power depletion. A base station is a single point of failure, and highly vulnerable to detection. The failure of a base station is especially troublesome in "sense and relay" sensor networks since in such networks, sensor nodes do not store data locally. Moreover, heavy transmission traffic to a base station might cause power depletion in sensor nodes that are one-hop away from the base station. A secure and reliable file system for sensor nodes can be helpful in the case of both "sense and store" and "sense and relay" sensor networks since it will allow the long-term secure survival of data within the network.

Even if a base station were available, sensor nodes may be unable to transmit all their data to it due to power constraints. Scientists, military personnel in the intelligence and national defense communities, and engineers–typical intended audiences of sensor network data—will likely prefer raw observations based on time series rather than aggregated and representative values. Raw observations can help scientists develop more accurate models.

Specialized and Hostile-Environment Deployments. Wireless sensor networks deployed in hostile environments cannot use a base station due to its high vulnerability to detection. A base station is easily detectable by traffic analysis, or simply due to the fact that being more powerful, it is more visible. Additionally, wireless sensor networks that study long-term phenomena, such as the changes in stress and strain in a bridge over time (perhaps for seismic reinforcement) or a simple understanding of material fatigue, do not require a base station to be installed along with the sensor nodes. This is because such networks do not need to transmit any data to a base station immediately upon sensing. Such sensor networks can continue to record their observations on local storage for long periods of time—six months or longer. Data can be collected manually as required to build models and study the patterns reflected in the collected data. A secure and reliable file system for sensor nodes can be very helpful in such deployments because it can ensure the secure long-term survival of data despite individual node failures. Other examples of specialized deployments include sensor networks that study animal or bird migration, or those that monitor the presence or absence of pollutants over a long period of time, or those that study inactive volcanoes.

Although both "sense and store" and "sense and relay" wireless sensor networks can benefit from the secure and reliable file system presented here, wireless sensor networks deployed in hostile and specialized sensor networks can especially benefit from such a file system. Deployments of wireless sensor networks in hostile environments include networks that engage in battlefield reconnaissance, critical facilities such as nuclear power plants or armed forces base surveillance, or spying on embassies. It may not be possible to install a base station, due to its greater vulnerability for detection and it being a single point of failure, along with such a wireless sensor network. Therefore, a secure and reliable file system such as the one described in this paper becomes especially important for the success of the mission for which the sensor nodes are deployed. A secure file system can ensure the prevention of unauthorized access of data stored on the sensor nodes. The reliability portion can ensure the long-term survival of the data despite the failure of the sensor node that collected the data originally.

3. THREAT MODEL

The main goals of the secure and reliable file system presented in this paper are to prevent unauthorized access to and tampering with data stored on a sensor node's local storage, and to ensure the survival of data, with some degree of probability, despite node failures. The secure and reliable file system presented in this paper should be successful in achieving all of these objectives.

In order to prevent unauthorized data access, the file system encrypts the data collected by each sensor node. The file system uses a block cipher for encryption with CBC or CTR mode within each block. Since CBC or CTR mode is used within each block, the effects of a block getting garbled or deciphered are restricted to a single encryption block, typically 128-bits long. Each key is used only once: to encrypt the current file system block, and to generate the next key in the sequence. The key is then discarded after use. The seed used to kick-start key generation is kept secret and is not stored on the sensor node after the first block is stored. Therefore, sensor nodes are only able to generate new keys; keys that have been used and discarded cannot be regenerated at the sensor node since the node does not know the seed used to kick-start the key generation. This ensures that encrypted data remains secure at the sensor node by ensuring that neither the sensor node nor an adversary, who subverts or physically captures the sensor node, is able to decrypt the encrypted data. Unencrypted data, however, remains vulnerable to unauthorized access. An adversary, upon capturing a sensor node, is free to inject spurious traffic into the wireless sensor network. He or she will also be able to encrypt this spurious data, and generate the corresponding algebraic signatures, which the file system uses to detect data tampering, but the adversary cannot use this information to retrieve already-stored data.

In order to ensure the survival of data collected by sensor nodes despite node failures, the file system divides its encrypted blocks of data into smaller chunks, and distributes these chunks among an optimal combination of its one-hop and multi-hop neighbors (also referred to as its back-up buddies). as determined by using one of three methods: a fixed list of back-up buddies, choose m out of a list of n buddies, or dynamically choosing back-up buddies. The first method assigns each sensor node a fixed list of backup buddies. The second method allows a sensor node to choose m out of n buddies for a particular back-up session. The choice of these m buddies may be based on which of them are up at the time or which ones have sufficient remaining energy. The third method might allow a sensor node to dynamically choose its back-up buddies based on which ones are up, and which ones have sufficient power available. These back-up buddies will provide reliability via redundancy to the sensor network. Periodically, the originating sensor node can query its back-up buddies for algebraic signature corresponding to a random

data chunk specified by start and end offsets, and verify the stored data using the technique described by Schwarz and Miller [20]. This approach is collusion-resistant; moreover, severely constrained resources—power, processing, and storage—also restrict the sensor nodes' ability to collude.

An adversary, upon physically capturing a sensor node, can impersonate the sensor node, and prompt the originating sensor node to repeatedly back-up its data on to itself. Consider a scenario where the originating sensor node (say, node A), backs up its data on to its buddy, node B. Node B, having been captured by an adversary, throws away the chunks sent to it. When node A queries node B for algebraic signatures corresponding to some portion of the original data chunk, node B replies with the wrong answer or no answer at all. Node A, realizing that a particular data chunk is not being stored correctly, resends the chunk for back-up. Node B can, in this way, cause node A to repeatedly resend its data for back-up, ultimately causing the power depletion of several sensor nodes (especially node A) in the process.

The secure and reliable sensor node file system does not attempt to protect the sensor network from denial of service (DoS) attacks. It also does not protect sensor nodes from getting squished by a person or an animal. This is because adding hardware tamper-proofing to sensor nodes makes them prohibitively expensive. Node failures in sensor networks are common, and the failure or destruction of a large number of nodes can adversely affect the overall reliability achieved in the network. Chunks for back-up are already encrypted, therefore, sensor nodes do not need to worry about data tampering or confidentiality.

4. FILE SYSTEM DESIGN

This section presents the basic design of the secure and reliable file system. We first describe the assumptions made by the file system, including the storage that is available on individual sensor nodes. We then detail the file system design and discuss its security features, focusing on advantages and disadvantages of the design.

Assumptions. Since base stations are highly vulnerable to detection, especially in the case of hostile-environment deployments, and are single points of failure, the secure and reliable file system presented here assumes a distributed wireless sensor network architecture in which all sensor nodes are homogeneous. These sensor nodes are randomly deployed in a non-hierarchical manner without the presence of a base station or a data sink, as shown in Figure 1. The sensor nodes follow the "sense and store" paradigm, and continue to store recorded observations on their local storage, securely and reliably, until collection.

Sensor Node Local Storage. The primary local storage on sensor nodes is a NAND flash device. Flash devices, characterized by lower energy consumption, ultra-low idle current, high reliability, high storage capacity and density, and lower cost, are well-suited for the requirements of wireless sensor networks [15]. A NAND flash device is divided into pages, typically 512 bytes in size. Each page has an extra 16 bytes of "out-of-band" storage space to be used for either metadata or error correcting codes. Based on this, the secure and reliable file system assumes that a page (and a block) is 512 bytes in size with 16 bytes of extra storage. The extra 16 bytes of storage can be used to store algebraic signatures, or for error correcting codes. Alternatively, this extra storage can also be used to store a mapping between

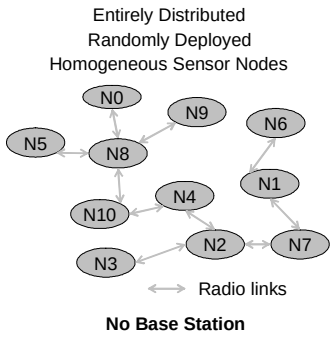

Figure 1: Network Architecture

the key generator and a specific key (*i.e.*, a mapping between a unique id that represents a specific key generator and a specific key number). Such a mapping will allow the use of several key generators in one sensor node. Another possibility is to utilize this space to store information regarding a specific key generator and the key sequence that will be required to decrypt the block.

A battery-backed SRAM or DRAM can be used to buffer writes, and to store the next key to be used for encrypting the next block of data. SRAM is especially useful for storing keys because it is easier to erase old values in SRAM. SRAM and DRAM are unsuitable for sensor nodes because both lose their contents when their power supply is removed. MRAM [16] and phase-change flash [12] may also be used for sensor node local storage. Both MRAM and phase-change flash offer faster read and write speeds over flash, and do not require wear-leveling. Phase-change flash is still largely under development.

Reliability. To ensure that the observations recorded by a particular sensor node survive the originating sensor node's failure, the secure and reliable file system divides the encrypted data stored in its local storage into chunks. It then distributes these chunks to its back-up buddies. The back-up buddies are some optimal combination of its one-hop and multi-hop neighbors. The goal of the file system is to gain reliability through redundancy so that the data collected by the sensor nodes survives failure of individual sensor nodes. The reliability scheme will be similar to the one described in [20], which provides several advantages over RAID-like redundancy schemes. These advantages include error correction, the option to not retain the original copy of the data, and low overhead for verification of correct storage at remote sites. All these advantages will benefit sensor nodes since they have constrained storage and processing resources. Provable data possession [1] cannot be used because of its high performance overhead. Unlike a machine equipped with Intel Pentium IV processor, a sensor node is severely constrained in resources in terms of memory and processing.

The goal of this reliability scheme is not mirroring since mirroring will increase the number of sensor nodes in the network significantly, thereby, increasing the overall cost of deployment [11]. This reliability scheme also does not deploy a back-up scheme such as the one described in [11]. Periodically, the originating sensor node checks to make sure that its back-up buddies correctly store the data designated to them. A POTSHARDS-type [22] group authentication

scheme may be utilized in order to achieve this goal. Several issues related to reliability and group authentication arise, and will be addressed in future work.

Simulation. The behavior of "sense and store" sensor nodes described here will be simulated by adding a middleware layer to J-Sim. A power model for storage, encryption, and key generation will also be added. Due to the overhead of key generation, encryption, computation of algebraic signatures, division of encrypted data into chunks for back-ups, it is expected that the performance of reads and writes experienced by the secure and reliable file system presented here will be slower as compared to that of a generic file system (such as, JFFS2) designed for flash devices.

4.1 Security

Before deployment, at the staging area, each sensor node is assigned a unique seed. The mapping between a sensor node and its unique seed is stored securely at the staging area, as shown in Figure 2. The unique seed never leaves the staging area, and is never stored on the sensor node. The unique seed assigned to each sensor node is used to kick-start key generation for that sensor node. The unique seed is provided as an input to a one-way key generating function. The first ever key could also be computed as a function of the node ID and a single secret key, *i.e.*, $F(nodeId, singleSecretKey)$, assigned to the sensor node. This will avoid the overhead associated with tracking n different initial keys for each key generator. This key generating function may be any one-way function, such as, a pseudo-random number generator (PRNG) or a one-way hash. The key generating function with the unique seed as its input generates the first 128-bit key.

Node Id	Seed
1	xxx
2	yyy
...	...
N	zzz

Figure 2: Node-Seed Mapping

The key generating function used to generate the first key is stored on the sensor node's local storage. The first key generated from it is not stored on the sensor node's local storage. This is important because if keys are stored on the flash device, it will be difficult to erase them later on. Instead, keys (the first and all subsequent ones) are stored on battery-backed SRAM or DRAM, as shown in Figure 3. The first key is used as an input into the one-way key generating function to generate subsequent keys in the sequence. Key generation continues in this manner—the previous key is input into the key generator to generate the next key in the sequence. A key, after the encryption of one block of data and the generation of the next key in the sequence, is permanently discarded.

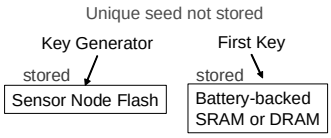

Figure 3: Sensor Node Storage

After deployment, sensor nodes follow their usual duty cycle and continue to record observations based on their mission's objectives. Data recorded by a sensor node on its local storage media remains unencrypted until a block (512 bytes) worth of data has been recorded. Unencrypted data resides on a sensor node's battery-backed SRAM or DRAM, and not on the sensor node's local flash device. This is because, if stored on flash, unencrypted blocks will remain there until they are garbage collected, which might not happen for a long period of time. In the mean time, the unencrypted data remains vulnerable. Moreover, the existence of the same data in both encrypted and unencrypted form may help reveal the encryption key, thus defeating the entire encryption scheme. After a sensor node has collected one block (512 bytes) worth of data, it proceeds to encrypt this block of data, as shown in Figure 4.

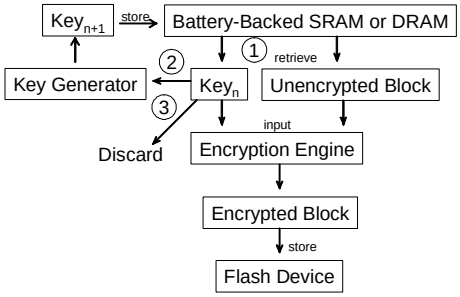

Figure 4: Encryption

The first step in the encryption process is to retrieve, from the sensor node's battery-backed SRAM or DRAM, an unencrypted block of data and the encryption key to be used; and provide them as inputs to the encryption engine. This is shown in step 1 in Figure 4. The encryption engine uses a block cipher, such as RC5 or AES, with CBC or CTR mode within a block. This enables each block to be independent of other blocks. This also restricts the adverse effects of a block getting garbled or deciphered to that particular block itself [5]. Symmetric ciphers are preferred in sensor networks over public key encryption because of the heavy overhead they inflict [28]. The encrypted block of data thus obtained is stored on the sensor node's flash device. The key used to encrypt the data block in step 1 is input into the key generator, as shown in step 2 of Figure 4. The key generator generates the next encryption key in the sequence and saves it on the sensor node's battery-backed SRAM or DRAM. The encryption key used in steps 1 and 2 is then discarded. The encryption of subsequent blocks of data, and the generation of subsequent encryption keys proceeds in a similar manner.

4.2 Analysis

The encryption scheme described above is highly suitable for sensor nodes because sensor nodes typically follow a "write-once, modify-never, access-never" storage and access paradigm. In other words, in "sense and store" wireless sensor networks, observations recorded by the sensor nodes are seldom accessed or modified by the sensor nodes themselves. Most of the access and modification of observations collected by sensor nodes occurs post-deployment at the staging area where data is decrypted for its intended audience. The encryption scheme described here is not suitable for typical file

systems in which users must be able to access and modify the encrypted data.

Encrypting stored data is nothing new for file systems. Several file systems, such as, Cryptfs [27], Plutus [9], and CFS [2], among several others encrypt stored data; the key rotation used by the file system presented here was inspired by Plutus. However, these file systems cannot simply discard an encryption key after one use because they need to ensure that their users are able to access and modify the encrypted data. Unlike the encryption scheme described in this paper, a majority of file systems also cannot function without storing the seed used to kick-start key generation on their local storage. The file system presented here, unlike Cryptfs, does not need to bother with self-authentication since an adversary, upon capturing or subverting a sensor node, will be able to perform exactly the same functions that the sensor node itself can perform. Unlike Plutus, the file system presented in this paper does not need to differentiate between readers and writers. This is because the sensor node, which is the writer, can only encrypt data going forward. It has no way of decrypting data. At the staging area, the data is decrypted by the intended audience.

The security scheme described above offers several advantages. One major advantage of this scheme is that it is simple and easy to understand. Therefore, there are fewer chances of making a mistake in its implementation, and the simple code can easily be run on low-powered sensor nodes. Another major advantage of this scheme is that the sensor node is not burdened with the responsibilities of key management and revocation. Since a key is discarded after one use, neither a sensor node nor an adversary is able to decrypt the already encrypted data. This is because both the sensor node and the adversary are unable to generate keys going backward since they do not possess the unique seed used to kick-start the key generation. Even if an adversary deciphers one cipher text block, it does not compromise or affect other blocks since CBC or CTR mode is used only within a block. However, this holds only if a block's decryption does not reveal the encryption key. If the encryption key is revealed, then all future encryptions are lost because an adversary will be able to generate all keys, going forward, starting from the revealed key.

The encryption scheme used by the secure and reliable file system enables both the sensor node and an adversary to encrypt data going forward since both possess the mechanism to generate keys going forward. Therefore, unencrypted data remains vulnerable to unauthorized access and malicious tampering. An adversary can also inject spurious data into a sensor node and encrypt it. Malicious tampering of encrypted data can be detected with the use of algebraic signatures.

The intended audience of the observations recorded by the sensor network will be able to decrypt all the data recorded by each sensor node because they can regenerate keys from the very beginning. This is because they possess the unique seed assigned to each sensor node. The regeneration of keys at the staging area can be optimized by indexing every n^{th} (say, 500^{th} or 1000^{th}) key generated by a particular key generator. Such indexing can reduce the overhead of regenerating all the keys up to the desired key. As an example, assume that the index stores every 500^{th} key; and that the 520^{th} is required for decryption. The 500^{th} key from the index can be used to kick-start key generation. With the index, only 20 keys (instead of 520 keys) need to be generated to get to the 520^{th} key.

5. RELATED WORK

This section compares and contrasts the features and characteristics of the reliable and secure file system presented in this paper with that of other file systems and security systems for sensor nodes and wireless sensor networks.

Matchbox [7] does not offer sufficient storage to add security to severely constrained Mica motes. It is difficult to add security into ELF [3] and eNVy [25] because these are complex and heavyweight. JFFS2 [24] is a good candidate for a local file system. But, data blocks will still need to be encrypted, and the encryption scheme presented here is a good choice to do so. PRESTO [4], TinyDB [14], Data-Centric Storage [21], Dimensions [6], Directed Diffusion [8], ANSWER [18] and SecureSense [26], and file abstractions designed by Tilak *et al.* [23], and Pisupati *et al.* [19] assume hierarchical network architecture organized around a base station or a data sink. This is in contrast to the distributed, flat network architecture, devoid of a base station, assumed by the file system presented here. A centralized network architecture with a command node, which is similar to a base station, is a single point of failure.

The primary focus of security in TinySec [10] and MiniSec [13] is secure communication between nodes. Both do not focus on storage security. Moreover, both TinySec and MiniSec, unlike the file system presented here, are riddled with the burden of key management. With TinySec, an adversary might be able to decrypt encrypted data upon physically capturing or subverting a sensor node. This is not the case with the reliable and secure file system presented here because both the sensor node and the adversaries can generate keys going forward only. Most of the security systems described above are burdened with key management and safety, and do not focus on secure and reliable storage.

6. CONCLUSIONS AND FUTURE WORK

The secure and reliable file system presented in this paper assumes a flat, distributed network organization devoid of a base station. It relies on "sense and store" paradigm, and utilizes a block cipher (such as, AES or RC5) in CBC or CTR mode to encrypt stored data. Each sensor node is assigned a unique seed before deployment. This unique seed never leaves the staging area, and is used as an input into the key generator to generate the first key in a sequence. This first key and the key generator are stored on the sensor node. Each key is used only once to encrypt data. It is then input into the key generator to generate the next key in the sequence and then discarded. By doing this, sensor nodes as well as adversaries are able to generate keys only going forward. Therefore, both are unable to decrypt already encrypted data. Algebraic signatures are used to prevent malicious tampering of data. Unencrypted data remains vulnerable. For reliability, each sensor node divides its encrypted data into chunks and backs-up these chunks on an optimal combination of single-hop and multi-hop neighbors. Periodically, each originating sensor node verifies whether or not one of its neighbors belongs to its back-up buddies list, and whether or not it stores the chunk assigned to it correctly. The performance of the reliable and secure file system presented in this paper will be evaluated by adding "sense

and store" paradigm to J-Sim. Several open issues (such as determining an optimal mix of single-hop and multi-hop neighbors, and the frequency of incremental back-ups and their verifications) exist and will be addressed in future research efforts.

7. REFERENCES

[1] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D. Provable data possession at untrusted stores. Cryptology ePrint Archive, Report 2007/202, 2007.

[2] Blaze, M. A cryptographic file system for Unix. In *Proceedings of the 1st ACM Conf. on Computer and Communication Security* (Nov. 1993), pp. 9–15.

[3] Dai, H., Neufeld, M., and Han, R. ELF: an efficient log-structured flash file system for micro sensor nodes. In *SenSys '04* (New York, NY, USA, 2004), ACM Press, pp. 176–187.

[4] Desnoyers, P., Ganesan, D., Li, H., Li, M., and Shenoy, P. PRESTO: A predictive storage architecture for sensor networks. In *Proc. of the 10th Workshop on Hot Topics in OS* (June 2005).

[5] Ferguson, N., and Schneier, B. *Practical Cryptography*. Wiley, 2003.

[6] Ganesan, D., Greenstein, B., Estrin, D., Heidemann, J., and Govindan, R. Multiresolution storage and search in sensor networks. *Trans. Storage 1*, 3 (2005), 277–315.

[7] Gay, D. Matchbox: A Simple Filing System for Motes. http://www.tinyos.net/tinyos-1.x/doc/matchbox-design.pdf, August 2003.

[8] Intanagonwiwat, C., Govindan, R., and Estrin, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking* (2000), pp. 56–67.

[9] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. Plutus: scalable secure file sharing on untrusted storage. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)* (San Francisco, CA, Mar. 2003), USENIX, pp. 29–42.

[10] Karlof, C., Sastry, N., and Wagner, D. TinySec: a link layer security architecture for wireless sensor networks. In *SenSys '04*, ACM, pp. 162–175.

[11] Koushanfar, F., Potkonjak, M., and Sangiovanni-Vincentelli, A. Fault tolerance techniques in wireless ad-hoc sensor networks. In *Proc. of IEEE Sensors '02*, vol. 2, pp. 1491– 1496.

[12] Lai, S. Current status of the phase change memory and its future. *IEDM '03 Technical Digest*, 10.1.1–10.1.4.

[13] Luk, M., Mezzour, G., Perrig, A., and Gligor, V. Minisec: a secure sensor network communication architecture. In *IPSN '07* (New York, NY), ACM, pp. 479–488.

[14] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30*, 1 (2005), 122–173.

[15] Mathur, G., Desnoyers, P., Ganesan, D., and Shenoy, P. Ultra-low power data storage for sensor networks. In *IPSN '06* (New York, NY), ACM, pp. 374–381.

[16] Miller, E. L., Brandt, S. A., and Long, D. D. E. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Schloss Elmau, Germany, May 2001), pp. 83–87.

[17] Mitra, A., A., B., W., N., Zeinalipour-Yazti, D., V., K., and Gunopulos, D. High-Performance Low Power Sensor Platforms Featuring Gigabyte Scale Storage. In *IEEE/ACM 3rd Int'l Workshop on Measurement, Modelling, and Perf. Analysis of Wireless Sensor Networks* (2005).

[18] Olariu, S., Eltoweissy, M., and Younis, M. ANSWER: Autonomous Wireless Sensor Network. In *Q2SWinet '05* (New York, NY, 2005), ACM, pp. 88–95.

[19] Pisupati, B., and Brown, G. File System Framework for Organizing Sensor Networks. In *SAC '06:* (New York, NY, 2006), ACM Press, pp. 935–936.

[20] Schwarz Thomas, S. J., and Miller, E. L. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *IEEE International Conference on Distributed Computing Systems (ICDCS '06)* (2006).

[21] Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., and Estrin, D. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev. 33*, 1 (2003), 137–142.

[22] Storer, M. W., Greenan, K. M., Miller, E. L., and Voruganti, K. POTSHARDS: secure long-term storage without encryption. In *USENIX '07* (June 2007), pp. 143–156.

[23] Tilak, S., Pisupati, B., Chiu, K., Brown, G., and Abu-Ghazaleh, N. A file system abstraction for sense and respond systems. In *Workshop on End-To-End, Sense-and-Respond Systems, Applications and Services* (2005), USENIX, International Conference On Mobile Systems, Applications And Services.

[24] Woodhouse, D. JFFS : The Journalling Flash File System, 2001.

[25] Wu, M., and Zwaenepoel, W. eNVy: a non-volatile, main memory storage system. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Oct. 1994), ACM, pp. 86–97.

[26] Xue, Q., and Ganz, A. Runtime security composition for sensor networks (SecureSense). In *IEEE Vehicular Technology Conference* (October 2003), vol. 5, pp. 2976– 2980.

[27] Zadok, E., Badulescu, I., and Shender, A. Cryptfs: A stackable vnode level encryption file system. Tech. rep., Columbia University, 1998. http://www.am-utils.org/docs/cryptfs/cryptfs.html.

[28] Zhou, L., Ni, J., and Ravishankar, C. V. Efficient Key Establishment for Group-Based Wireless Sensor Deployments. In *WiSe '05* (New York, NY, 2005), ACM, pp. 1–10.