RAIM: Redundant Array of Independent Motes

Dominik Schürmann, Felix Büsching, Sebastian Willenborg, Lars Wolf
Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany
{schuermann, buesching, willenborg, wolf}@ibr.cs.tu-bs.de

***Abstract*—Wireless Sensor Networks in real world context are per se error-prone; motes can be lost, destroyed, corrupted, or stolen. Some scenarios demand a high level of confidentiality and integrity so that an attacker cannot access or alter the secret data. Other setups may require a high level of redundancy so that data from many motes can be recovered by only collecting a few.**

In this paper we show how both can be achieved at the same time by designing a protocol based on Shamir's Secret Sharing, symmetric ciphers, and Message Authentication Codes to distribute, encrypt, and authenticate the data. Our implementation RAIM supports the whole bandwidth from full redundancy to full confidentiality. RAIM is the first security implementation for Contiki OS, which uses secret sharing techniques. It has been deployed on physical motes, where its real world energy consumption and throughput has been measured. In addition to real world experiments, we provide a simulation to evaluate all possible configurations.

***Index Terms*—Wireless Sensor Networks, Shamir's Secret Sharing, Contiki OS, redundancy, replication**

I. Introduction

"Whatever can go wrong will go wrong." – Murphy's Law is true for nearly every system ever built, and particularly for Wireless Sensor Networks (WSNs) or Cyber-Physical System (CPS). In theory, thousands to millions of Sensor Nodes – so-called motes – are installed over wide areas [1]. In practice, at least a few to hundreds of motes are deployed to observe wildlife [2], prevent fire [3], monitor the structural health of buildings [4], or the well-being of humans [5] with (wireless) Body Area Networks (BANs).

By virtue of the sheer number of motes alone, there is a high risk of single failures on single motes. This is a known problem and handled by default by many techniques and/or protocols such as dynamic routing for changing topologies or simple hardware watchdogs for failing devices. Especially in outdoor deployments, a high level of robustness is needed due to the mostly harsh and varying environmental conditions.

If such networks are constantly connected to a sink and data is forwarded to this sink right after being recorded, there is usually no critical loss of data to be expected. Additionally, in such constantly connected networks, privacy and security issues could be handled by a PKI or by using pre-deployed keys. But, in reality, many scenarios do not have a constant end-to-end connection of every mote to the sink. Especially in wildlife observation or health monitoring scenarios, there might not even be a sink and motes have to be collected to gather the recorded data. Moreover, additional things may occur to motes besides the "normal" failures. Motes can be destroyed – either by mistake or by intention. They can be corrupted – either by bad programming skills or by an attacker's intention. Depending on the specific scenario, data stored on howsoever lost motes is at least lost or in the hand of an attacker which may be even worse. We discuss two scenarios to introduce these challenges.

A. Scenario 1: Wildlife Monitoring – Maximum Redundancy

In many wildlife monitoring scenarios, such as ZebraNet [2], motes are attached to the observed animals or deployed in the environment where the observation takes place. Obviously, in real life outdoor scenarios that include wild animals, sensors get lost or destroyed and not every mote can be detached and readout in the end – perhaps the specific animals are missing as well. Nevertheless, it would be desirable to have the recorded data of all sensors. Thus, storing the locally generated data on as many motes of other animals as possible leads to a high level of redundancy. With an increasing level of redundancy, the number of motes that will have to be collected for data analysis decreases, e.g., with "full redundancy" only one animal has to be found and caught.

B. Scenario 2: Personal Health Monitoring – Maximum Confidentiality

In a BAN, motes attached to the human body measure vital parameters or activity data. Thus, high confidentiality is required to protect this sensitive information, e.g., by using One Time Pads (OTPs) [6]. Not every sensor node in a BAN may have a dedicated storage, some will send the recorded data instantly to other more powerful nodes. On the other hand, BANs may also not constantly be connected to some backend network and recorded data has eventually to be stored locally for a longer period of time. In such scenarios sensors can get lost, stolen, or destroyed, e.g., by accidentally putting them into the washing machine. In contradiction to the first scenario, a pure redundancy is neither needed nor intended. Thus, confidentiality may be the higher target in BANs, but some redundancy for single lost nodes would be beneficial.

C. Different Levels of Redundancy and Confidentiality

In comparison to wildlife monitoring and BAN, the requirements for many other use cases like production monitoring [7] or structural health monitoring [4] can be located somewhere between these scenarios. Thus, the ideal level of redundancy at maximum possible confidentiality differs from use case to use case and has to be adjusted according to the individual needs.

D. Contributions

In the following sections, we show how both can be achieved at the same time by utilizing RAIM. We combine existing methods for confidentiality, data integrity and redundancy with a well-defined storage format. Besides symmetric ciphers and Message Authentication Codes, we utilize a (k, n)-threshold Shamir's Secret Sharing (SSS) scheme to provide the security property that an attacker needs to get at least k SD cards to access the data collection. No amount of less motes reveal any data. While it is possible that one SD card is physically stolen, it is more unlikely that k SD cards will be stolen.

In this paper, we provide the first SSS-based implementation for combined secure data distribution and redundancy. This allows us to actually deploy RAIM on physical motes and measure its energy consumption and throughput. We are able to show that it provides low energy consumption due to the usage of hardware AES, while also having low overhead resulting in a negligible decrease in throughput.

In Section 2 we present related work regarding security schemes based on secret sharing schemes in WSNs. RAIMs design and attack model is discussed in Section 3. In Section 4, the implementation of RAIM is given with a focus on its storage layout. RAIM's energy consumption and throughput is evaluated in Section 5. Section 6 concludes the paper.

II. Related Work

WSNs are increasingly deployed in malicious environments and, thus, exposed to external attackers. Due to the motes' limited memory and computing power, routing protocols are often kept stateless and simple. This makes them an easy target for a range of attacks such as Sybil, Sinkhole, and Wormhole attacks [8]. To achieve data integrity, confidentiality, and authentication of senders, several key management schemes have been proposed in the literature [9]. Real-world security architectures have been designed and implemented for TinyOS, Contiki, and other WSN operating systems [10]–[14]. While these systems protect against passive eavesdropping and active man-in-the-middle attacks, attacks against data availability by destroying motes or interfering with their memory are not considered. In the following we look at data redundancy protocols and architectures that provide both confidentiality and data redundancy.

Di Pietro *et al.* describe a local distribution protocol that randomly selects neighbors to share their data with [15]. It is based on SSS [16] to ensure confidentiality and redundancy properties. Their simulation assumes mobile unattended motes and shows the probability of data recovery based on time depending on the selection of the threshold and number of shares. Besides SSS, erasure codes can be utilized to achieve encrypted data redundancy. Wang *et al.* propose an efficient data redundancy scheme based on secret sharing and erasure coding [17]. Their design is sound, but has not been evaluated by an implementation or simulation besides their storage and communication overhead analysis. Chessa *et al.* introduce protocols using erasure codes based on Redundant Residue Number Systems (RRNS) [18], [19]. For this, an abstract model has been defined and theoretically evaluated, i.e., it has been shown that the RRNS requirement of assigning modulis to motes is NP-complete and thus a random assignment is appropriate [19]. Similar to Di Pietro *et al.*, they showed a trade-off between confidentiality and availability for WSNs with a specific density and redundancy level. Girao *et al.* [20] propose tinyPEDS, a storage framework for static nodes that subdivides a network into clusters. A balance between security and overhead is achieved by using asymmetric (PH_a) and symmetric (PH_s) additive privacy homomorphism functions. The authors analyze the performance of the underlying cryptographic operations and simulate their protocol using GloMoSim. While their simulation shows that their protocol succeeds to recover data even when $40\,\%$ of all motes are exhausted, it is important to note the complexity of their protocol potentially complicating a real implementation. Abdullahi *et al.* [21] introduce a protocol that encrypts payload using Elliptic Curve Integrated Encryption Scheme (ECIES) and divides it into fragments, which are replicated across the network. Their threat model includes collusion, pollution, and data dropping attacks as well as guarantees forward and backward secrecy. A performance evaluation has been done with Matlab. A protocol based on Homomorphic Encryption and Homomorphic Secret Sharing has been proposed by Ren *et al.* [22]. The homomorphic property of their proposed protocol allows for minimization of storage overhead and transmission cost. Their numerical evaluation shows that their assumptions hold for scenarios where the probability of data modification and destruction attacks is less than $40\,\%$. Chen *et al.* [23] implement a distributed file system based on secret sharing. Instead of using SSS, they use Blakley's, which is based on hyperplanes [24]. In comparison to other publications, they discuss their file system design in detail and provide a experimental performance analysis of their implementation, which uses matrix calculations on GPUs. Bagci *et al.* [25] propose Fusion, a storage extension for existing IPSec and DTLS architectures. Their main contribution is to decrease the number of cryptographic operations by mapping incoming encrypted packets to stored blocks instead of re-encrypting them for storage. They provide a real-world implementation for the Contiki OS. However, no data redundancy for recovery is considered.

Most described protocols considering data redundancy have only been evaluated theoretically or by simulations. Only two real world implementations have been described [23], [25]. However, the hyperplane calculations of Chen *et al.* require GPUs which are hardly available on motes [23], while the Fusion framework does not protect against data destruction attacks [25]. Our goal is to provide an implementation and evaluation on real mote hardware using a well chosen combination of confidentiality, data integrity and redundancy. To the best of the authors' knowledge, we are the first to not only implement a redundancy scheme based on SSS for Contiki OS, but also deploy this implementation on physical motes to evaluate real world energy consumption and throughput.

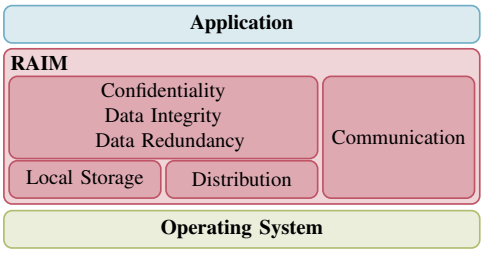


Fig. 1: RAIM's modular architecture.

TABLE I: Cipher suite configurations

Cipher Suite	Properties
CRC only [a]	Error Detection
CBC-MAC only	Data Integrity
AES	Confidentiality
OTP	Confidentiality
OTP + CBC-MAC [b]	Confidentiality + Data Integrity
AES + CBC-MAC [b]	Confidentiality + Data Integrity

[a] Only checksum calculation, no security properties.
[b] Encrypt-then-Authenticate (EtA) [26]

III. RAIM

RAIM is designed to provide protection against an external attacker, i.e., an attacker who can intercept and alter communication but is not part of the network. Also, protection against non-complex internal attacks is provided. In addition, it allows to recover data in case of hardware failures, which can happen frequently in low-cost WSNs. Finally, RAIM is low on energy consumption and network overhead (cf. Section V).

A. Design

As depicted in Figure 1, RAIM's architecture consists of a modular layer between applications and the operating system that allows the secure distribution of data across several motes. It provides data redundancy, unified storage/transport confidentiality, as well as data integrity. Furthermore, RAIM provides an additional communication layer to allow distribution of redundancy data and communication with other RAIM-enabled motes.

Due to differing requirements arising from variable scenarios, RAIM can be configured in a flexible way. The encryption layer can be configured for usage with the OTP streaming cipher using a locally stored key-file requiring external storage or for usage with the Advanced Encryption Standard (AES) in Cipher Block Chaining Mode (CBC) Mode. CBC-MAC is available as a Message Authentication Code (MAC) for authenticating stored and transmitted data. If no confidentiality and integrity is required, Cyclic Redundancy Check (CRC) checksums can provide error detection only. A list of sensible cipher suite configurations is depicted in Table I. If CBC-MAC is combined with an encryption algorithm, it is always applied via Encrypt-then-Authenticate (EtA) [26], providing protection against chosen-ciphertext attacks. If hardware support is available, we recommend AES + CBC-MAC, otherwise OTP + CBC-MAC is also a valid option (cf. Section V). RAIM's redundancy can be configured for either full redundancy or with a chosen k for a (k, n)-threshold SSS scheme.

Data is encrypted and authenticated on the sender before handing it to the communication layer. For AES a configured pre-shared key is used for the whole network, for OTP a key-file is generated from random separately for each mote, which is consumed from an external storage when deployed. The communication layer broadcasts the data to all neighboring motes for distribution. On receiving motes, the CBC-MAC is verified and a share is calculated directly from the encrypted data using the (k, n)-threshold SSS redundancy scheme. We decided to execute SSS on the receiving motes to allow for broadcasting. Executing the scheme on the sending side requires unicasting the encrypted data which would result in a huge amount of network overhead. On the receiving mote, shares are directly saved in the assigned storage, that is limited in its capacity per neighboring mote. If an external storage medium is attached, the shares are saved there, but RAIM also works with integrated flash storages. To protect against replay attacks, packets contain a counter which is increased over time. A packet is only accepted if the mote has not already received a packet with this counter and the counter is greater than the least received counter value. It is important to note that data is never decrypted on motes. Decryption and restoring of shares is done on external hardware after collecting the motes in case of wildlife monitoring or periodically using powerful sink motes in case of BANs.

To reduce energy consumption, RAIM's AES implementation is based on hardware-backed symmetric cryptography and directly processes encrypted data without separation between storage and transport confidentiality. A detailed evaluation of RAIM's energy consumption can be found in Section V-C2.

B. Attack Model

As presented before, the security of our architecture includes transport and storage confidentiality as well as data integrity. To maintain a proper balance between security measures and the limited battery and processing power of motes, no public key infrastructure has been implemented or deployed. Still, RAIM protects against external attacks and common internal attacks as outlined in this section.

1) External Attacker: First, an external attacker is assumed with no physical access to the motes.

Eavesdropping/man-in-the-middle: If RAIM has been properly configured for OTP/AES + CBC-MAC according to Table I, transmitted data is encrypted and authenticated. Thus, the data cannot be read by simple eavesdropping. It is important to note that for OTP mode, each mote gets its own key-file to prevent an attacker from guessing messages when eavesdropping ciphertexts c_1, c_2 and applying $c_1 \oplus c_2$. Also, due to the CBC-MAC, a man-in-the-middle attacker is not able to replace parts of the encrypted data to change the meaning of the content.

Insert data: In this attack, an attacker sends random or fake data to motes of the WSN. If RAIM has been configured for OTP/AES, the attacker does not possess the network's pre-shared key and is, thus, not able to properly encrypt data for the network. In addition, RAIM's communication layer accepts only packets suitable for the current firmware configuration to prevent downgrade attacks.

Replay attacks: An attacker who records communication between neighboring nodes could replay this recorded data to exhaust the storage of the originally receiving mote. This attack is prevented because RAIM includes an index that is authenticated using CBC-MAC and receiving nodes verify that this index has not been used before (cf. Section III-A).

2) Internal Attacker: In the following scenarios, an internal attacker is assumed, e.g., a compromised mote that is still part of the WSN. Most of these attacks are hard to execute in practice as they require physical access to the motes and special hardware equipment as well as knowledge.

Stolen external storage: In case of AES encryption, stealing the external storage, e.g., the microSD, does not lead to an exposure of the encryption key which is configured in the mote firmware. Thus, the data cannot be decrypted and is still confidential. In case of OTP, already consumed parts of the local key file are overwritten, i.e., already encrypted data and old communication is still secure. However, when an attacker has physical access to re-attach this storage medium, remaining key blocks can be extracted. Further considerations in case an AES key is stolen in regards to the SSS scheme are described in the "compromised AES key" scenario.

Storage exhaustion: In this attack, a number of compromised motes continuously transmit random data to all other motes filling their storage to hinder RAIM's data distribution. To weaken this attack, each "$mote_id" file (cf. Section IV-A) is limited to prevent storage exhaustion.

Pollution/data dropping: In this attack, a number of compromised motes modify or drop their stored shares [21]. When using (k, n)-threshold SSS for redundancy, restoring will still work as long as no more than $n-k$ motes have been compromised.

Stolen/destroyed mote: If a mote gets broken or is deliberately destroyed, RAIM's redundancy layer prevents data loss. When using (k, n)-threshold SSS, if no more than $n-k$ motes are gone, the data can still be fully recovered. As discussed in the next attack, as long as no firmware/memory extraction has been performed, the data is still confidential.

Compromised AES key: In case of AES encryption, the pre-shared key could be compromised by extracting the firmware/memory or recording the SPI communication to the radio. This attack requires direct physical access as well as hardware equipment and knowledge. A compromised key allows for decryption of future communications and addition of new motes to the network. Still, past data collections can only be recovered when k motes/SD cards are stolen, where k is defined by the (k, n)-threshold SSS scheme. No amount of less motes than k provide *any* data recovery.

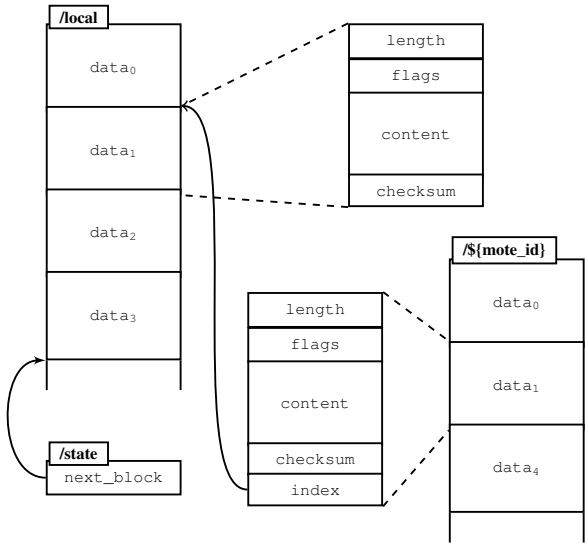


Fig. 2: Layout for data storage depicting complete local data blocks as well as appended and incomplete data blocks for "$mote_id".

IV. Implementation

The presented architecture has been fully implemented for the Contiki OS. Some methods are based on hardware support of our open source mote called *Inexpensive Node for General Applications (INGA)* [27]. RAIM has been published on https://www.ibr.cs.tu-bs.de/projects/raim and licensed under a 3-clause BSD-style license. In the following, we describe RAIM's modular layers in detail.

A. Storage

The local storage is implemented using a special directory structure based on a FAT32 file system supporting internal and external storage. Due to the ease of exchanging microSDs, these are preferable in comparison to an integrated flash storage. As depicted in Figure 2, three types of files are utilized. The file "local" contains all data blocks generated on the mote itself. These are consecutively written to this file. When utilizing OTP encryption, this file is also the key file used as a one time pad in conjunction with "state" holding a relative offset to the beginning of the next data block. This allows the system to append new data correctly and prevents overwrites even after restarting the system. Besides the actual content (+ checksum), "local" contains a length field for the whole block including the length of the field itself. The flags field encodes the parameters how the data has been secured, i.e., the encryption, data authentication, and redundancy methods. Data received from other motes is stored in separate files named by mote ID, i.e., "$mote_id". Additionally, an index is appended to each block pointing to the corresponding data block in "local". This index is used as an ID in the restoration process. Because received data blocks are not modified, the layout for single data blocks is not only used on storage but also on transport layer.

If, instead of using the index field, data would be stored at the corresponding position inside the file, attackers could overwrite existing storage positions. In addition, storage space would be allocated without actually using them on motes that do not receive all shares. The difficulty of restoring the data would then increase due to the created unused storage spaces, which must be detected beforehand.

B. Confidentiality

Confidentiality is achieved on transport as well as on storage layer by encrypting data prior to sending (cf. Section III-A). RAIM supports OTP, which can achieve high throughput rates even in software, as well as hardware-backed AES.

OTP has been implemented to operate in-place, i.e., the targeted storage location within the local file is read as an OTP key, the data is encrypted in-memory, and then overwrites the consumed key locally. Thus, sensed data is always encrypted directly without being written to a flash storage in clear. During the recovery process, OTP-encrypted data can be handled in the same way as unencrypted data due to the usage of the same file layout.

While OTP always requires a large storage medium, AES can be configured for motes without additional storage hardware. Due to the usage of an AT86RF23X radio on our INGA mote, AES-128 has been implemented using hardware support. As described before, the utilized pre-shared AES key must be configured before firmware compilation. To prevent correlation of plaintext to ciphertext blocks, AES is operating in CBC mode. The Initialization Vector for CBC is read from the storage address of the local file. Prior to encryption, the plaintext data is padded using null bytes to 16 byte data blocks.

C. Integrity

Data integrity is achieved by configuring RAIM for the usage with CBC-MAC to protect against malicious tampering of transmitted data using pre-shared keys. The CBC-MAC tag is calculated using the hardware-backed AES methods with a pre-shared key, which can be configured independently from the AES key. The output of the last AES pass is used as the CBC-MAC tag which is appended to the transmitted data for verification on the receiver. To prevent concatenation attacks, the data length is included together with the first data block. The scheme can be described by

$$AES(data) \parallel index \parallel CBC\text{-}MAC(AES(data) \parallel index).$$

While we recommend AES + CBC-MAC, it is difficult to deploy pre-shared keys in legacy WSNs. Thus, RAIM also provides the possibility to use CRC checksums, providing Error Detection only without shared keys. RAIM utilizes a CRC-8 checksum with $x^8 + x^2 + x + 1$ polynomial which is stored in one byte.

D. Data Redundancy and Distribution

Using the data format described in Section IV-A, data is distributed accross motes inside the WSN. To achieve data redundancy, RAIM can be configured for either full redundancy or with a chosen k for a (k, n)-threshold SSS scheme. When using SSS, receiving motes calculate their share using their own ID. Based on the configured threshold, k defines directly how many bytes are used for calculating a single share. An example for a $(3, 4)$-threshold SSS scheme is depicted in Figure 3.

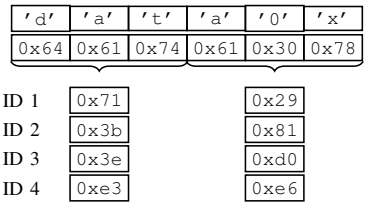
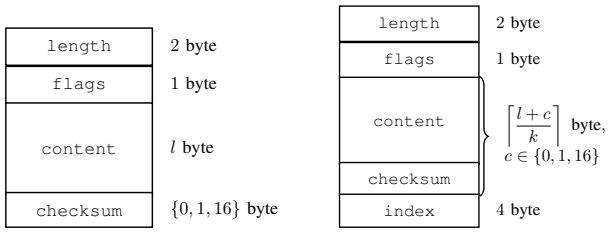


Fig. 3: Example distribution by dividing 6 bytes of data into 2 chunks where each 3 byte long chunk is distributed among 4 motes. Every mote gets a 1 byte share and, thus, for each original 3 byte chunk to recover, 3 out of 4 motes are required.



(a) Local storage of sensing mote.

(b) Storage of neighboring remote mote.

Fig. 4: Storage requirements for l byte of data.

V. Evaluation

We evaluated RAIM's storage requirements on local and remote motes, simulated and physically tested its functionality and throughput. Finally, its energy consumption has been measured in our laboratory.

A. Storage Requirements

Storing data on the local mote is different to storing replicated data received from neighboring motes. Thus, these scenarios are evaluated separately. As depicted in Figure 4a, besides the payload of length l, either 1 byte (CRC) or 16 byte (CBC-MAC) is appended. When configuring for AES, the storage requirement increases to the next size that is divisible by 16. Using these values, equations for lower (Equation 1) and upper bound (Equation 3) can be derived.

$$f_{min,local}(l) = l + 3 \tag{1}$$

$$f_{max,local}(l) = \left\lceil \frac{l+16}{16} \right\rceil \cdot 16 + 3 \tag{2}$$

$$= \begin{cases} l + 16 + 3 & \text{for } l \bmod 16 = 0 \\ l + 16 - (l \bmod 16) + 16 + 3 & \text{otherwise} \end{cases} \tag{3}$$

TABLE II: Zebra scenario: Percentage of recoverable data after the loss of one node at different levels of redundancy.

	full redundancy	SSS $k=2$	SSS $k=3$	SSS $k=4$
mean	96.5	88.0	72.8	52.9
σ	3.9	12.2	21.1	25.7

On neighboring remote motes, the storage requirements depend on SSS with a (k, n)-threshold for payloads of length l with an additional index appended. SSS reduces the payload and yields to a significantly reduced lower (Equation 4) and upper bound (Equation 5) for remote motes.

$$f_{min,remote}(l) = l + 7 \tag{4}$$

$$f_{max,remote}(l) = \left\lceil \frac{\lceil \frac{l+16}{16} \rceil \cdot 16}{k} \right\rceil + 7 \tag{5}$$

In the worst case, an overhead of 38 byte still exists, but even when using SSS with $k = 2$, it is reduced to 19 byte.

B. Simulation of Scenarios

The two scenarios described in the introduction define a baseline for maximum redundancy (Wildlife Monitoring – Section I-A) and maximum confidentiality (BAN – Section I-B). So, we simulated these scenarios using Contiki's Cooja.

1) Wildlife Monitoring: In this dynamic scenario 20 of 30 nodes have a simulated storage. All nodes are constantly moving on random but previously determined paths and record respectively transmit their position. We simulated four different configurations for (k, n)-threshold SSS which differ in the number of shares required for data recovery ($k = 1, \ldots, 4$).

In Table II can be seen that in case of full redundancy, the recovery rate for single lost nodes has a mean value over all nodes of $96.5\,\%$. The configuration with $k = 4$ is not very well suited for this scenario, because nearly $50\,\%$ of data cannot be recovered; if SSS should be applied to such a scenario, the configuration $k = 2$ is much better suited.

2) Personal Health Monitoring: We simulated eight motes of a BAN which randomly generate data. Only a subset of four nodes (mote IDs 1-4 in Table III) have a simulated storage; the others are just able to send data. As expected on a human body, every node is within the radio range of every other node. (k, n)-threshold SSS has been configured with $k = 3$. Additionally, AES + CBC-MAC has been used to provide confidentiality and integrity. After running the simulation for 13 hours for simulating 4 hours, each node generated more than 100 packets. In Table III it can be seen which amount of data is recoverable when a single node is missing. Thus, in this scenario between $97.5\,\%$ and $100\,\%$ of the generated data can be recovered – with a storage overhead of just $33\,\%$.

3) Throughput Simulation: In Figure 5, the mean duration and the specific standard deviation for several configurations at 2,000 simulation runs is given. It is obvious that simple storing on the mote without sending any redundant information to other motes is faster than anything else. It is interesting to note that there is no difference if just a SSS share of data is sent and stored on other motes or if a full data-set is sent.

TABLE III: Recoverable data after the loss of one node.

mote ID	stored packets	restored packets	[%]
1	118	115	97.5
2	113	112	99.1
3	125	125	100.0
4	119	117	98.3
5	105	105	100.0
6	109	109	100.0
5	118	118	100.0
8	135	134	99.3

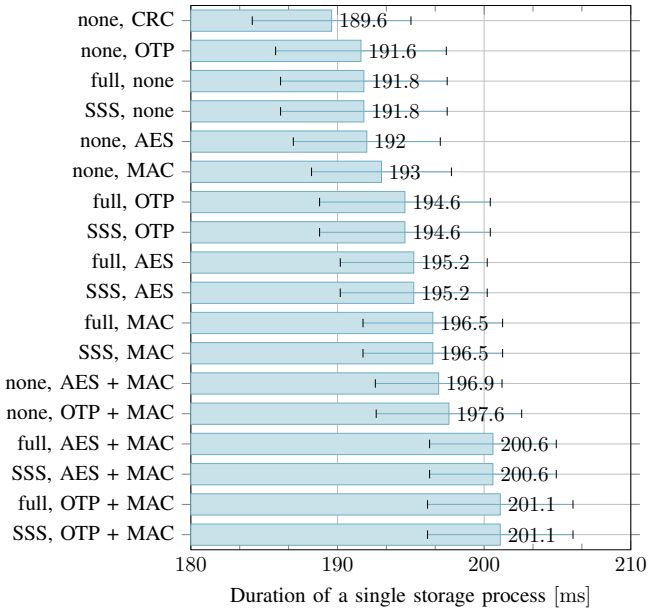
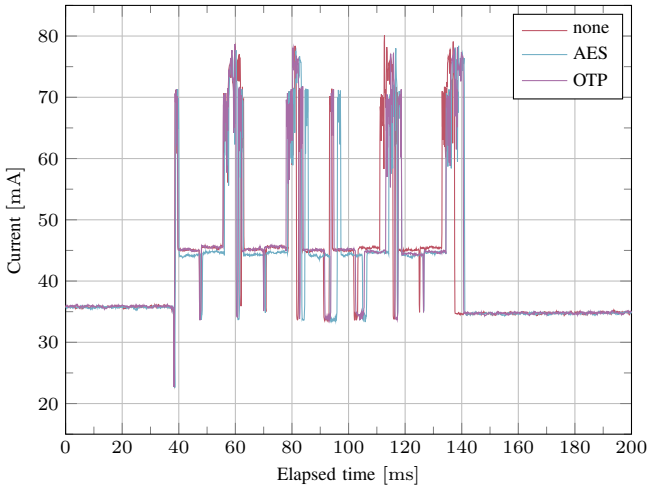


Fig. 5: Storage durations depending on (redundancy, cipher suite)-configurations.

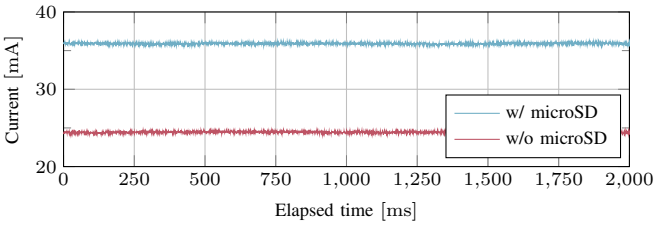
Looking at the different encryption methods, surely "no encryption" is fastest; followed by AES. This may be confusing at first glance, because OTP encryption is just a simple XOR and AES needs several rounds; but, OTPs need another local storage access for reading the specific pads which surely has an impact on the overall performance.

C. Measurements Using Physical Motes

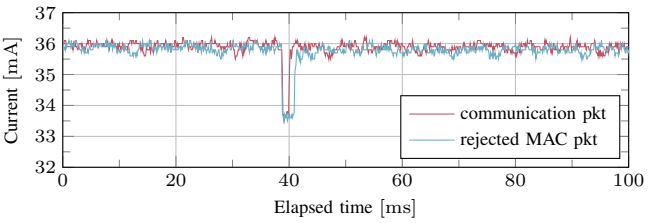
We used a static setup of 9 motes which have been deployed in our lab (Figure 6) for at least 1.5 days. Motes 3 and 6 have been placed outside the building, mote 8 was located in a climate chamber. During the experiments the motes measured and sent temperature and barometric pressure every $60\,\mathrm{s}$.



(a) Consumption on storing and transmitting data using different symmetric cipher configurations (w/ microSD).



(b) Additional consumption due to microSD.



(c) Consumption on receiving packets (w/ microSD).

Fig. 8: Energy consumption measured using the physical motes.

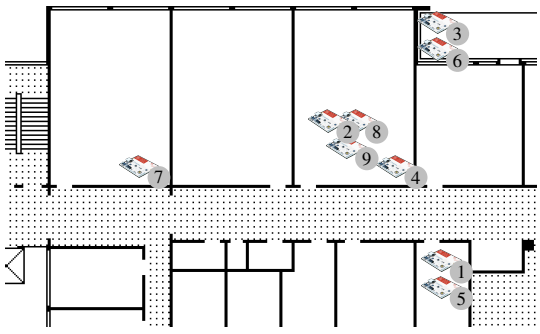


Fig. 6: Location of motes in our lab experiment.

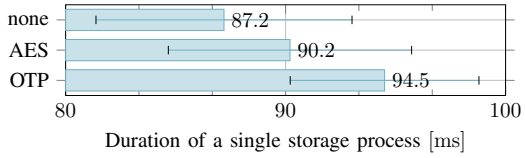


Fig. 7: Measured throughput on physical motes depending on the chosen symmetric encryption.

1) Real Throughput: In Figure 7, the real throughput on a physical mote is shown. In comparison to our simulation results in Figure 5, our real-world measurements are 100 ms faster. Thus, while the effects predicted by the simulator can be observed in the physical implementation, the absolute values of the simulation results must be adjusted upwards by 100 ms.

2) Energy Consumption: To analyze the energy consumption of RAIM, the consumption of the mote itself is measured by using a shunt resistor of 5 Ω with a digital storage oscilloscope. The consumption has been observed while the motes where connected to a laboratory power supply without using the voltage regulator of the mote itself.

A baseline measurement in the mote's "idle" mode has been executed first. As shown in Figure 8b, a stable 24.4 mA is consumed without microSD and 36.0 mA with microSD attached, i.e., using the microSD consumes additional 11.6 mA.

Energy consumption during storing and transmitting data via RAIM, i.e, read and write access, can be observed in Figure 8a. Read access can be seen as a short peak with 70 mA followed by a phase with 45 mA where 512 byte test data is transmitted. Write access behaves inverse: A longer 45 mA phase can be observed followed by am 75-80 mA peak. The order of read and write accesses conform to the expected behavior, i.e, depend on the limitation of the mote's memory on one sector. Every sector needs to be loaded for access and after editing again written to the storage. Of special interest is the time span between 80 ms and 120 ms in Figure 8a: The observed double access is due to an additionally required access to the FAT32 cluster table. In case of AES, the encryption is executed before read and write access. For OTP, the local key file needs to be read prior to its execution and write access. We found no significant difference in energy consumption between RAIM's AES and OTP mode.

The energy consumption for an incoming communication packet and due to a rejected CBC-MAC packet has been measured and compared in Figure 8c. It shows that CBC-MAC verification has a similar energy footprint as a default communication packet and thus works as a mechanism against attackers trying to exhaust the mote's energy. Compared to Figure 8a, no further energy is consumed since no storage encryption and writing process is executed.

VI. Conclusion

RAIM utilizes a combination of Shamir's Secret Sharing together with symmetric ciphers and MACs to distribute, encrypt, and authenticate the data within a WSN. RAIM is flexible and configurable – according to the specific and countless application scenarios in which data is gathered and stored within the WSN. Thus, data can be transmitted securely over insecure wireless channels and be stored on neighboring motes. We discussed the design of RAIM, attacks to be considered, and corresponding implementation details. The evaluation, using simulations as well as physical motes, shows that RAIM provides a suitable approach for data distribution in WSNs, while being low on energy consumption. RAIM can be accessed at https://www.ibr.cs.tu-bs.de/projects/raim.

References

[1] J. M. Kahn, R. H. Katz, and K. S. Pister, "Next century challenges: mobile networking for Smart Dust," in *Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking*, ACM, 1999, pp. 271–278.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet," in *Proc. of the 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, California: ACM, 2002, pp. 96–107.

[3] M. Hefeeda and M. Bagheri, "Wireless Sensor Networks for Early Detection of Forest Fires," in *IEEE Int. Conf. on Mobile Adhoc and Sensor Systems (MASS 2007)*, Oct. 2007, pp. 1–6.

[4] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks," in *6th Int. Symp. on Information Processing in Sensor Networks (IPSN 2007)*, Apr. 2007, pp. 254–263.

[5] F. Büsching, M. Bottazzi, W.-B. Pöttner, and L. Wolf, "DT-WBAN: Disruption Tolerant Wireless Body Area Networks in Healthcare Applications," in *The International Workshop on e-Health Pervasive Wireless Applications and Services (eHPWAS'13)*, Lyon, France, Oct. 2013.

[6] F. Büsching and L. Wolf, "The Rebirth of One-Time Pads – Secure Data Transmission from BAN to Sink," *IEEE Internet of Things Journal*, vol. 2, no. 1, pp. 63–71, Feb. 2015.

[7] T. O'Donovan, J. Brown, F. Büsching, A. Cardoso, J. Cecelio, O Jose do, P. Furtado, P. Gil, A. Jugel, W.-B. Pöttner, U. Roedig, J. sa Silva, R. Silva, C. J. Sreenan, V. Vassiliou, T. Voigt, L. Wolf, and Z. Zinonos, "The GINSENG System for Wireless Monitoring and Control: Design and Deployment Experiences," *ACM Sen. Netw.*, vol. 10, 4:1–4:40, Nov. 2013.

[8] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," *Ad Hoc Networks*, vol. 1, no. 2–3, pp. 293–315, 2003.

[9] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A survey of key management schemes in wireless sensor networks.," *Computer Communications*, vol. 30, no. 11-12, pp. 2314–2341, Nov. 2, 2007.

[10] C. Karlof, N. Sastry, and D. Wagner, "TinySec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, MD, USA: ACM, 2004, pp. 162–175.

[11] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," in *6th Int. Symp. on Information Processing in Sensor Networks (IPSN 2007)*, Apr. 2007, pp. 479–488.

[12] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the limits of Elliptic Curve Cryptography in sensor networks," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, R. Verdone, Ed., vol. 4913, Springer, 2008, pp. 305–320.

[13] L. Casado and P. Tsigas, "ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System," in *Identity and Privacy in the Internet Age*, ser. Lecture Notes in Computer Science, vol. 5838, Springer, 2009, pp. 133–147.

[14] L. B. Oliveira, D. F. Aranha, C. P. Gouvêa, M. Scott, D. F. Câmara, J. López, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," *Computer Communications*, vol. 34, no. 3, pp. 485–493, 2011, Special Issue of Computer Communications on Information and Future Communication Security.

[15] R. Di Pietro and S. Guarino, "Confidentiality and availability issues in Mobile Unattended Wireless Sensor Networks," in *14th Int. Symp. and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2013, pp. 1–6.

[16] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[17] Q. Wang, K. Ren, S. Yu, and W. Lou, "Dependable and Secure Sensor Data Storage with Dynamic Integrity Assurance," *ACM Trans. Sen. Netw.*, vol. 8, no. 1, 9:1–9:24, Aug. 2011.

[18] S. Chessa, R. Di Pietro, and P. Maestrini, "Dependable and Secure Data Storage in Wireless Ad Hoc Networks: An Assessment of DS2," in *Wireless On-Demand Network Systems*, ser. Lecture Notes in Computer Science, R. Battiti, M. Conti, and R. Cigno, Eds., vol. 2928, Springer, 2004, pp. 184–198.

[19] M. Albano and S. Chessa, "Replication vs erasure coding in data centric storage for wireless sensor networks," *Computer Networks*, vol. 77, no. 0, pp. 42–55, 2015.

[20] J. Girao, D. Westhoff, E. Mykletun, and T. Araki, "TinyPEDS: Tiny persistent encrypted data storage in asynchronous wireless sensor networks," *Ad Hoc Networks*, vol. 5, no. 7, pp. 1073–1089, 2007.

[21] M. Abdullahi, G. Wang, and F. Musau, "A Reliable and Secure Distributed In-network Data Storage Scheme in Wireless Sensor Networks," in *10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Nov. 2011, pp. 548–555.

[22] Y. Ren, V. Oleshchuk, and F. Li, "A distributed data storage and retrieval scheme in unattended WSNs using Homomorphic Encryption and secret sharing," in *2nd IFIP Wireless Days (WD)*, Dec. 2009, pp. 1–6.

[23] S. Chen, Y. Chen, H. Jiang, L. T. Yang, and K.-C. Li, "A Secure Distributed File System Based on Revised Blakley's Secret Sharing Scheme," in *11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2012, pp. 310–317.

[24] G. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the 1979 AFIPS National Computer Conference*, Monval, NJ, USA: AFIPS Press, 1979, pp. 313–317.

[25] I. E. Bagci, S. Raza, U. Roedig, and T. Voigt, "Fusion: coalesced confidential storage and communication framework for the IoT," *Security and Communication Networks*, Apr. 2015.

[26] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL?)" In *CRYPTO 2001*, Springer, 2001, pp. 310–331.

[27] F. Büsching, U. Kulau, and L. Wolf, "Architecture and evaluation of inga-an inexpensive node for general applications," *Sensors*, pp. 842–845, 2012.