End – to – End Secure Data Delivery in Wireless Sensor Networks

Alexandros Zaharis
University Of Thessaly
Volos, Greece
alzahari@inf.uth.gr

Leonidas Perlepes
University Of Thessaly
Volos, Greece
leperlep@inf.uth.gr

George Stamoulis
University Of Thessaly
Volos, Greece
georges@inf.uth.gr

Panagiotis Kikiras
University Of Thessaly
Volos, Greece
kikirasp@inf.uth.gr

***Abstract* — Typical sensor nodes are resource constrained devices containing user level applications, operating system components, and device drivers in a single address space, with no form of memory protection. A malicious user could easily capture a node and tamper the applications running on it, in order to perform different types of attacks. In this paper, we propose a 3-Tier Security Framework composed by physical security schemes, cryptography of communication channels and live forensics protection techniques that allows for secure WSN deployments. Each of the abovementioned techniques maximizes the security levels leading to a tamper proof sensor node. Even if the physical protection of the nodes is bypassed which is common in wireless sensor attacks, the attacker must surpass 2 more security tiers in order to perform a valid attack on the underlying network. By applying the proposed security framework, secure communication between nodes is guaranteed, identified captured nodes are silenced and their destructive effect on the rest of the network infrastructure is minimized due to the early measures applied. Our main concern is to propose a framework that balances its attributes between robustness, as long as security is concerned and cost effective implementation as far as resources (energy consumption) are concerned.**

Keywords - Security Framework; Sand-boxing; live forensics; cryptography; wireless sensor networks.

I. INTRODUCTION

Wireless Sensor Networks (WSN) are emerging as an important tier in the IT ecosystem where active research involving hardware and system design, networking, distributed algorithms, data management and security, is blended to deal with a unique environment with distinctive characteristics and demands. The main function of a sensor network is the utilization of tiny sensing devices which are capable of sensing various types of incidents/parameters and communicating those with other devices. Sensor networks sensing can be applied for many applications such as target tracking, surveillance, environmental monitoring, etc. [30].

Due to the unattended environment on which wireless sensors operate and the resource constrained nature of these devices in the manner of computational capabilities, memory size and available energy, it is a major challenge to employ efficient security schemes coming from the computers or ad hoc wireless networks domain [2][30].

In this paper, the critical security issues in wireless sensor networks are addressed, various types of threats and attacks against them are explored in order an efficient multi-tier security framework to be proposed.

Furthermore, an evaluation of the combination of cryptography of the communication channel is presented along with valid sand-boxing techniques for providing protection in energy constrained embedded sensor nodes

The three layers of the proposed framework are:

a) Physical Sensor Protection
b) Sand-Boxing
c) Crypto-Communication

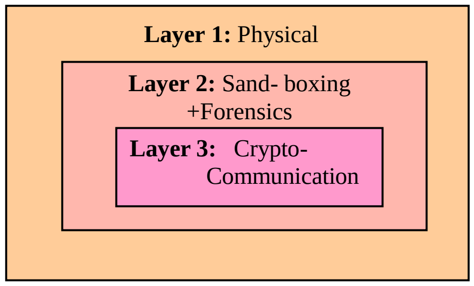
Layer 1: Physical
Layer 2: Sand- boxing +Forensics
Layer 3: Crypto-Communication

Figure 1: Multi layer Security Framework

Framework's primary goal is the effective blending of common security techniques such as physical security or cryptography with more modern ones like sand-boxing [1][2][26][27].

The proposed protection framework is thoroughly presented along with real life use examples that prove its robustness and effectiveness against the most popular WSN security attacks. The overall concept of combining live forensics along with "sand-boxing" techniques and other commonly used security schemes as cryptography in a single framework is, to our knowledge, a unique and out of the box security attempt that can lead to an impenetrable multi-tier security framework.

The remainder of this paper is organized as follows: Section II provides a review of similar security techniques and frameworks. In Section III, we briefly explain the components on which the framework is based on. Section IV describes in details the proposed framework. In Section V, the framework's efficiency against different types on attacks in explained. Finally, Section VI some concluded remarks are presented.

II. Similar Work

Attacks on the sensor network can be classified as:

a) Physical attacks on sensor devices, e.g., destroying, analyzing, and/or reprogramming sensors.
b) Service disruption attacks on routing, localization
c) Data attacks, e.g., traffic capture, spoofing.
d) Resource-consumption and denial-of-service (DoS) attacks.

One of the serious attacks to the sensor networks deployed in an unattended environment is physical tampering with sensors. An adversary can easily capture, reverse-engineer the sensor, and deploy (multiple clones of) manipulated sensors. The compromised sensors will then be exploited by the adversary to mount actual attacks which will facilitate the subversion of the entire network.

Traditionally, the tamper-proofing of programs relies on tamper-resistant hardware [1][2]. However, hardware-based protection will likely fail to provide acceptable security and efficiency on its own because 1) strong tamper-resistance is 'expensive' to be implemented in resource-limited sensor devices and 2) the tamper-resistant hardware itself is not always absolutely safe due to various tampering techniques [1][3][4] such as reverse-engineering on chips, microprobing, glitch and power analysis, and cipher instruction search attacks.

Existing approaches to generating tamper-resistant programs without hardware support can be classified as:

a) Code obfuscation that transforms the executable code to make analysis/modification difficult [5][6][7][8].
b) Result checking that examines the validity of intermediate results produced by the program [9][10][11].
c) Self-decrypting programs that store the encrypted executables and decrypt them before execution [12][13].
d) Self-checking that embeds, in programs, codes for hash computation as well as correct hash values to be invoked to verify the integrity of the program under execution[12][14][15].
e) Software based Attestation to remotely verify the integrity of sensor software [20].

However, most of the above mentioned approaches will more likely fail on sensor networks where a program runs on slow, less-capable microcontrollers.

Software attestation is a challenge-response protocol where a verifier (e.g., base station) sends an attestation command to the attester (the node being attested) asking for certain state information as the evidence of its software integrity. Such state can be computed correctly only if the attester's system meets certain integrity requirement. After receiving the response, the verifier compares it with the known good state to check if the software at the attester has been corrupted. If a sensor node fails to give the correct answer, actions can be taken to revoke this node from the network. Several software attestation schemes have been proposed to attest the static memory regions of the software [17][18][19][20].

Physical hardening of the sensor is the first obstacle an attacker must overcome in order to tamper a wireless sensor. The effectiveness of the physical security on sensors is usually low and a WSN based only on physical security cannot be considered as secure. In our approach physically securing a sensor is the layer of defense mostly used to prevent less determined attackers. Our second defense scheme strips the major functions of a live forensics check on an average system in order to match with the limited resources of a sensor, leading to the important conclusion of whether a sensor is compromised.

The live forensics security layer proposed in this paper verifies the integrity of the program residing in each sensor through a process that has been specifically designed to:

a) Prevent altering / manipulation / reprogramming of the sensor
b) Be purely software-based.
c) Work on sensor devices with severe resource limitations
d) The verification of the different parameters tested does not add large overhead to the communication.
e) Prevent eavesdropping attacks on the communication channel.

III. Live Forensics Framework

As the need for decentralized security emerges in large public wireless sensor networks; new application level security mechanisms aim at providing application developers with appropriate abstractions for designing the security aspects of the target software.

In computer security, a sandbox is a security mechanism for separating running programs. It is often used to execute untested code, or untrusted programs from unverified third-parties, suppliers and/or untrusted users.

It typically provides a tightly-controlled set of resources for guest programs to run in, such as scratch space on disk and memory. In this sense, sandboxes are a specific example of virtualization.

Zaharis et al. [26] proposed a protocol based on sandboxing technique. According to this approach, they divide isolates in two categories:

- The Security-Dedicated Isolates ("SDI")
- The Work-Dedicated Isolates ("WDI")

An Isolate Verification Server plays a key role on verifying the genuine WDIs from the malicious ones while performing all the computational and energy consuming and needy tasks.

The verification of genuine WDIs is based on 1) RAM dumbing and 2) Hashing techniques.

This framework uses a secure RAM dumping technique specially designed for sensors. This technique provides the framework with safer intrusion recognition while complying with the classic digital forensic techniques.

The Hashing technique that is used by the framework is based on the Randomized Hashing Function [16]. This technique is used on the Work-Dedicated isolates in order to acquire highly secure tamper-proofing on sensor-resident programs. The hashing function plays a key role in the effectiveness of the proposed architecture as it is robust technique, used frequently in computer security and digital forensics due to its precision in detecting altered code.

Our goal is to improve this mechanism by enriching it with cryptographic procedures, in order to provide a secure end-to-end data delivery framework.

A. *Cryptography*

The Tiny Encryption Algorithm (TEA) is a cryptographic algorithm designed to minimize memory footprint and maximize speed. It is a Feistel type cipher that uses operations from mixed (orthogonal) algebraic groups [31]. Despite its robustness minor extensions have been published in order to present safer encryption results. In this research, we determine the weaknesses and identify the robustness of TEA, XTEA and XXTEA algorithms in wireless sensor networks and implement them in secure framework to harden security during communication [27][28][29].

The conditions must be met in order the algorithm to bet truly "inseparable" are:

- The distribution of keys must have been to all nodes in a secure manner.
- Each message uses a secure, unique key.
- The key generation has become with a truly random cryptographic way

In order to generate a set of unique, truly random keys, we use the Random Number Generator service designed and operated by the University of Trinity [25]. RANDOM.ORG's source of entropy is atmospheric noise. This noise is obtained by tuning a radio to a radio frequency that no one is using. It is then played into a workstation where a program converts it to an 8-bit mono signal at a frequency of 8 KHz. Then the first seven bits are discarded and the remaining bits are gathered together. This stream of bits has very high entropy.

A possible attack on the generator is to broadcast on the frequencies that the RANDOM.ORG radios use in order to affect the generator. However, radio frequency attacks of this type would be difficult for a variety of reasons. First, the frequencies that the radios use are not published, so an attacker would have to broadcast across all frequencies of all bands used for FM and AM broadcasting. Second, this is not an attack that can be launched from anywhere in the world, only reasonably close to the generator. RANDOM.ORG currently has radio receivers in several different countries, which would make it difficult to coordinate this type of attack. Third, if an attacker actually did succeed at broadcasting highly regular signals (e.g., perfect sine waves) at exactly the right frequencies from the right locations, then the RANDOM.ORG real-time statistics would pick up the drop in quality very rapidly , which would raise an alert [25].

IV. Network Architecture

Our sensor network consists of an Isolate Verification Server (IVS) an Isolate Verification Database (IVDB) and numerous sensors which consist of an SDI and one or more WDIs. The Security-Dedicated Isolate (SDI) is the one executed on start up and conducts the forensics check of the second isolate. The 'SDI" is the one responsible for the communication with the Isolate Verification Server (IVS)

The Role of the Isolate Verification Server is:

- To communicate with the SDI of every sensor in its vicinity.
- To update/manage its local IVDB.
- To act as a trusted authentication third party.

For scalability, we let cluster-heads in a cluster-based hierarchical architecture serve as IVSs. This allows each IVS to maintain a local IVDB that stores SDI_IDs of the sensors belonging to its own cluster.

It is undesirable to equip only one IVS in a network as it becomes a single point of failure and the performance bottleneck and so use multiple IVSs can be deployed over the entire network. We assume that there exists a mechanism for a sensor to learn how to discover, and reach, an IVS.

The proposed architecture leads to a decentralized model of sensor protection where its cluster head / IVS is responsible for its sensors. Of course, all this information must be gathered in a master IVS with the total IVDB of the whole WSN.

A. *Sandboxing In Action*

In order to achieve the maximum tampering protection of our sensors, sand-boxing is applied to achieve safety against malicious code execution. While more than one Security-Dedicated Isolates can run on a sensor in our proposed Framework we will use one per sensor.

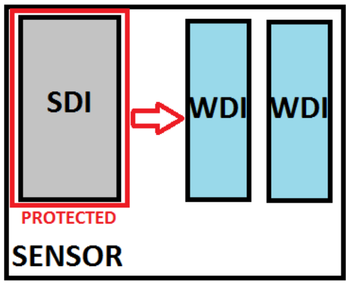

Figure 2: The isolates on a sensor

On the other hand, more than one Work-Dedicated Isolates can run on a sensor performing different tasks. The verification process performed on a single WDI applies for more than one instance with the same results. Failure to verify one of the WDIs leads to locking and blacklisting of the sensors.

1) Security dedicated isolate

The Security-Dedicated Isolate is actually a mini forensics tool case specially created to perform live forensics in a sensor, on demand or periodically in order to specify if the sensor is compromised and react depending on the result.

The Security-Dedicated Isolate has a unique id/key for every sensor, the “SDI_ID” that is used in order to communicate with the Isolate Verification Server.

On the first supervised boot the SDI is the first to execute and perform a mini mapping and state validation of the sensor. These results are stored on the Isolate Verification Database (‘IVDB’) which resides on the Isolate Verification Server (‘IVS’). As in every Digital Forensics case these data are going to be used as a proof of the sensors authenticity on the field.

From now on all data transmitted by the SDI are going to be compared to those stored on the ‘IVDB’ depending on the “SDI_ID”.

The tasks the SDI is responsible for are:

a) Communicating safely with IVS.
b) Checking the Work-Dedicated Isolates.
c) Applying countermeasures upon intrusion detection.

2) Work dedicated isolate

The Work-Dedicated Isolate performs the everyday tasks of a typical sensor. Due to the sand-boxing technology, more than one WDI can be executed simultaneously on a sensor, performing different tasks. Execution of non verifiable WDIs will lead to the activation of countermeasures by the SDI.

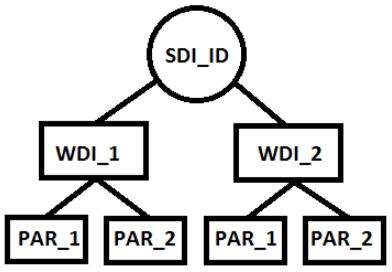

Figure 3: More than one WDI and their check parameters

The fingerprinting of the performance of every isolate on different parameters is stored on Isolate Verification Database along with the SDI_ID of the sensor on which the WDIs belong.

The fingerprinting parameters of a WDI can depend on:

a) The hash value of the isolate.
b) The RAM dump of the isolate.

B. State-Transition Diagram

Each sensor device is associated with one of four states:

a) “LOCKED”
b) “VERIFYING”
c) “ACTIVATED”
d) “COUNTERMEASSURES”

When a sensor starts its execution, it is in the LOCKED state. Upon deployment a sensor device will remain in LOCKED state until it securely authenticates with IVS. No other tasks can be performed until it is authenticated.

After a valid authentication, it makes a transition to the VERIFYING state by executing the SDI verification checks. The stripped results are transmitted back to the IVS where: If the verification fails, it returns to the LOCKED state, causing the network to deny this sensor’s access to the network. Otherwise, it transitions to the ACTIVATED state, in which the WDIs code is normally executed. Periodic re-verification by the SDI during ACTIVATED state can lead to LOCKED state or COUNTERMEASSURES state. COUNTERMEASSURES is the state in which a sensor is already accepted on the network and then compromised. In order to avoid denial of service attacks on which the attacker can lead all sensors to LOCK state, the COUNTERMEASSURES state can be used. In this state the compromised sensor tries to identify the type of attack on which it has been subjected through a different type of live forensics process. All other nodes ignore the compromised node through an alarm message send by the IVS. Finally it returns to the LOCKED state.

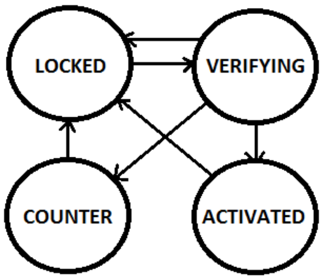

Figure 4: State Diagram

C. Authentication Protocol

The proposed protocol is consisted by three phases where certain actions must take place. These phases are divided into actions prior to deployment, during the “initialization” phase and, while in regular operation.

1) Pre-deployment phase

During the phase prior to deployment a set of random keys is generated by the base station. This set is stored to the tamper resistant storage area and it is the same for each of the network’s nodes. This set will act as the key repository from where the nodes and the base station will choose their encryption keys during the operational life of the network.

The generation of the keys prior to deployment allows for significant gains in the energy consumed by the nodes, due to the fact that in order to compute a strong cryptographically key, a number of complex mathematical operations and a sequence of iterations are required, which are energy consuming and computational demanding operations.

2) Initialization phase

After the deployment of the network the following actions are taking place:

a) Initiation: This step starts the authentication protocol between the IVS and the sensor by transmitting the SDI_ID. The sensor, after receiving the IVS_ID, asks for authentication.
If the authentication fails the protocol is terminated.
b) If authentication succeeds, SDI is executed.
c) The result of SDI is transmitted back to IVS. IVS checks the IVDB and validates the results. The received hash value and Ram dump are checked. If it passes the test, the IVS registers the sensor in the IVDB. Then, the IVS notifies the sensors SDI of the verification result.
d) Based on the verification result, the sensor is either activated or locked. The sensor state will be changed to either ACTIVATED or LOCKED, accordingly.

Step 1 ensures sensor security, i.e., a malicious device can neither passes the authentication procedure nor has its own code executed on the sensor as far as the IVS's authentication key is kept secret from the attacker.

3) Regular operation

After the initialization phase, the activated sensors can perform the data's collection, encryption and transmit ion to the base station.

All message transactions, described to the above phases, are encrypted using the XTEA cryptographic algorithm. Each message is encrypted with a key belonged to the set of random keys deployed during the Pre-Deployment Phase of the protocol.

D. *Verification Protocol*

The verification of a sensor is based on two widely used digital forensics techniques 1) Hashing (RHF) and 2) RAM dumping per WDI.

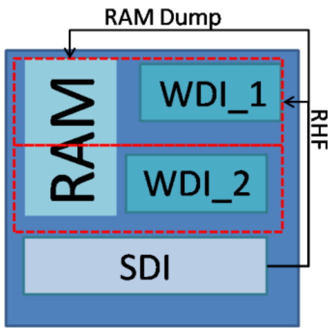

Figure 5: Fingerprinting a WDI

1) Hashing

Every Work-Dedicated Isolates has a unique Randomized Hashing Function (RHF)[16] which can be easily and with a minimum cost be calculated. Once calculated for every user it is stored on ISDB along with the SDI of every sensor creating the first fingerprint of the sensor.

Also thanks to the fact that sensors of the same network usually perform the same task can lead to a smaller number of different hash patterns stored on ISDB per WDI.

Each WDI can be classified as being 1) common to all sensors in the network, 2) common to a group of sensors with the same missions, or 3) unique to a specific sensor.

2) RAM Dump

When using this technique, our SDI reads arbitrary RAM contents from the different WDIs running on the sensor. Every process running on a system leaves specific, well distinguished footprint on the RAM. Our goal is to create hash like footprint of the memory and store it on ISDB along with the SDI of every sensor creating the second fingerprint of the sensor. In order to keep our framework in energy efficient levels specific parts of the RAM dump are checked concerning the execution of the WDIs.

These WDI –specific fingerprints are also hashed using the Randomized Hashing Function providing an extra protection parameter.

E. *Protocol Implementation*

In order to evaluate our protocol we have implemented it on Mica2 sensor nodes [23]. The MICA2 is a third generation mote module used for enabling low-power, wireless sensor networks. It consists of an ATMega128L CPU, 4kb of Ram, 128kb of program memory and 512kb of serial flash memory and a ChipCon CC1000 radio. The Crossbow MTS310 sensor board was used which provides temperature, and other sensor types.

The protocol is implemented in two parts; the first part corresponds to IVS code (Figure 6) and the second to sensor code (Figure 7).

IVS Code

```
on receive UserHashMsg:
        receive( idAddr , UserHashMsg );
        decrypt(UserHashMsg);
        if( check (UserHashMsg) == valid() ){
                VerifyMsg = Valid;
                encrypt(VerifyMsg);
                send( idAddr , VerifyMsg );
        }
        else{
                VerifyMsg = Invalid;
                encrypt(VerifyMsg);
                send( idAddr , VerifyMsg );
        }

on receive HashMsg:
        receive( idAddr , HashMsg );
        decrypt(HashMsg);
        if( IVDBcheck (HashMsg) == valid() ){
                VerifyMsg = Valid;
                encrypt(VerifyMsg);
                send( idAddr , VerifyMsg );
        }
        else{
                VerifyMsg = Invalid;
                encrypt(VerifyMsg);
                send( idAddr , VerifyMsg );
        }
```

Figure 6: IVS Code

The messages sequence diagram of the aforementioned code implementation can be seen in Figure 8.

```
Sensor Code
on boot:
          state = LOCKED;
          encrypt(UserHashMsg);
          send( broadcastAddr , UserHashMsg );

on receive VerifyMsg:
          receive( idAddr , VerifyMsg );
          decrypt(VerifyMsg);
          if( VerifyMsg==Valid && state==LOCKED){
                    state = VERYFING;
                    HashMsg = computeHashValue();
                    encrypt(HashMsg);
                    send( broadcastAddr , HashMsg  );
          }
          else if( VerifyMsg==Valid && state==VERYFING){
                    state = ACTIVATED;
                    start_data_process();
          }
          else{
                    state = LOCKED;
          }
```

Figure 7: Sensor Code

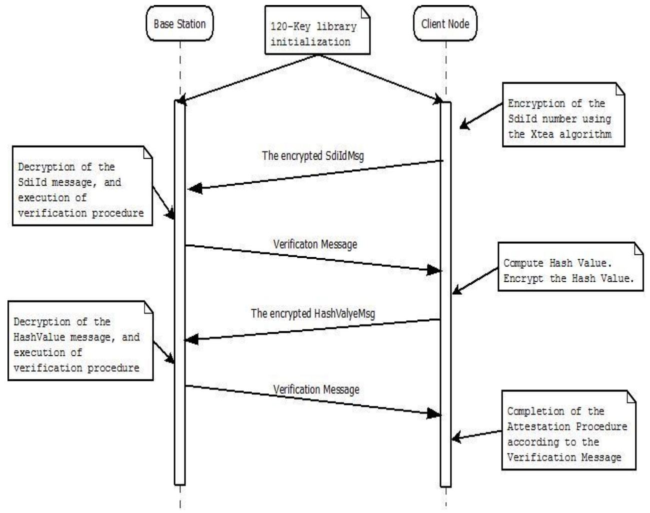
Base Station
120-Key library initialization
Client Node
Encryption of the SdiId number using the Xtea algorithm
The encrypted SdiIdMsg
Decryption of the SdiId message, and execution of verification procedure
Verificaton Message
Compute Hash Value. Encrypt the Hash Value.
The encrypted HashValyeMsg
Decryption of the HashValue message, and execution of verification procedure
Verification Message
Completion of the Attestation Procedure according to the Verification Message

Figure 8: Protocol Messages

a) During the Pre-deployment Phase, each node equipped with a set of secure keys. (120 keys.)
b) The client node encrypts the SDI_ID using the XTea algorithm.
c) The client node sends the 80-bit encrypted message to IVS
d) The IVS decrypts the message and checks the authenticity of the SDI_ID (this step was simulated with 1 sec delay during simulation).
e) IVS sends to the sensor the appropriate response (valid/invalid).
f) On valid response sensors turns from locked to verifying status, initiates the hashing and RAM-dumbing procedure (during the simulation we have used the SHA-1 algorithm (~13ms/hash) [24]. For the calculation of the hash – value the algorithm utilizes 512 bytes from the memory and produces a 160 bit hash.
g) The sensor encrypts the 160-bit hash values, producing a 208-bit message. The sensor transmits the 208bit message to the IVS for validation.
h) IVS verifies the validity of the sensor's hash value. (This step was simulated with 1 sec delay during simulation).
i) IVS validates or the sensor.
j) Sensor turns status into ACTIVATED or LOCKED in accordance with IVS message.

F. *Energy Analysis*

In order to measure protocol's energy consumption we have implemented it and simulate its performance in Avrora Simulator. Avrora [22] is a set of simulation and analysis tools for programs written for the AVR microcontroller produced by Atmel and the Mica2 sensor nodes. Avrora contains a flexible framework for simulating and analyzing assembly programs, providing a clean Java API and infrastructure for experimentation, profiling, and analysis. Avrora uses the AOEN (Accurate Prediction of Power Consumption)[21] energy consumer model. AOEN uses empirical current consumption measurements (of hardware such as the radio transceiver, microcontroller and sensors) to calculate the overall power consumption. AOEN is based on the execution of real application and OS code and measurements of node current draw, this model enables accurate prediction of the actual energy consumption of nodes. Thus, it prevents erroneous assumptions on device and network lifetime. Such a detailed prediction allows the comparison of different low power and energy aware approaches in terms of energy efficiency and the estimation of the overall lifetime of a sensor network.

1) *Energy cost of cryptography operations.*

Table 1 compares the energy consumed by the different versions of TEA cryptography algorithm. The values represent the energy consumed by a node in order to execute the following procedure:

- Encryption and sending of a 64-bit packet
- Receiving and decryption of the 64-bit packet.

The SIMPLE algorithm represents the procedure of sending and receiving the raw packet, without the execution of any cryptographic command.

We do not present the cost of key generation. We assume that the key is created during the pre-deployment phase, as described on section IV.

TABLE 1. ENERGY COST OF TEA CRYPTOGRAPHY ALGORITHMS IN ORDER TO ENCRYPT-SEND/DECRYPT-RECEIVE 64BIT DATA.(MJOULE)

Algorithm	Energy Cost	
	Encryption – Send	***Receive - Decryption***
SIMPLE	6041.81μJoule	6002.94 μJoule
TEA	6067.82 μJoule	6030.77 μJoule
XTEA	6071.98 μJoule	6033.19 μJoule
XXTEA	6087.23 μJoule	6042.30 μJoule

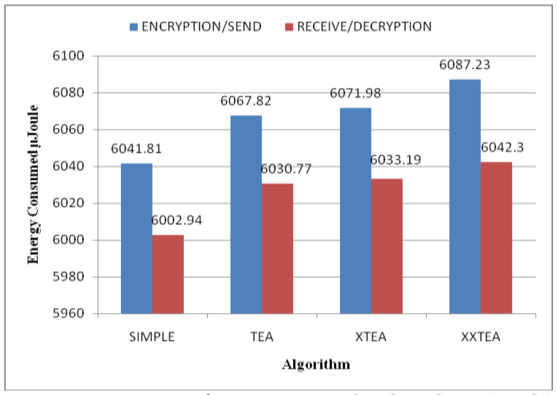

Figure 9: Energy cost of TEA cryptography algorithms. (µJoule)

2) Energy Cost of Attestation Protocol.

We analyze the energy usage of the Attestation's protocol handshake procedure. Table 2 compares the energy consumed by the 2 different version of the protocol. The main difference between these versions is the encrypted message transactions that are implemented in the second protocol. As described above, in the Encrypted Attestation Algorithm we the TEA cryptographic algorithm in order to provide an end-to-end secure data delivery protocol. For our analysis we chose to focus on XTEA version of TEA's family cryptographic algorithms.

TABLE 2. ATTESTATION'S PROTOCOL ENERGY COST (JOULE).

Protocol	**Energy Cost**		
	CPU Energy	***Radio Energy***	***Total Energy***
Attestation	0.02286 Joule	0.06034 Joule	0.08321 Joule
Encrypted Attestation	0.02311 Joule	0.06094 Joule	0.08406 Joule

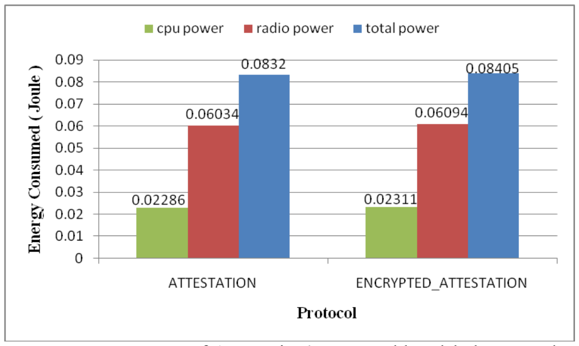

Figure 10: Energy cost of Attestation's protocol handshake procedure. (Joule)

V. SECURITY ANALYSIS

Examples of our proposed framework efficiency against different types on attacks will be displayed in this paragraph along with real life scenarios that prove its robustness.

A. *Physical Attacks*

Physical attacks that can impact the coverage of the WSN and in many cases make the WSN inoperable. Because of the widespread placement of the individual nodes in an often non-secure and unmonitored area, individual nodes are subject to capture. Physical hardening of the sensors against reverse-engineering on chips, microprobing, glitch and power analysis, and cipher instruction search attacks on the first layer of security of the proposed framework can lead to the needed results.

B. *Replay Attacks*

Replay attacks (i.e., intercepting a message and replacing it with an old message) cannot succeed as the proposed hash computation and verification are keyed operations that can be defeated as following: First, reporting a different SDI_ID will be detected by IVS when its uniqueness is checked and, moreover, the malicious sensor will not be able to pass the hash of RAM dump test unless it has the matching program which must be free of malicious codes and created an exact fingerprint. Second, modifying the Hash algorithm will cause inconsistency between two hash outputs and, hence, the verification will fail. Encryption of the communication channel makes it more difficult for an attacker to forge messages, as the messages have to be encrypted with the appropriate secret key.

C. *Forgery Attacks*

We will now show that it is impossible for the adversary to forge the hash value without the knowledge of all the specific parameters previously described, for each WDI. Consider the situation where the adversary reprograms the sensor with a malicious program and attempts to fake the verification process by nullifying the effect of the output of the Hash algorithm. This is impossible because the Hash algorithm is inherently a nonlinear function of program blocks. Thus it is impossible to create a malicious WDI that has the same RHF as the original. Encryption of the communication channel makes it impossible for an attacker to forge the communication between two nodes as unique keys are distributed to every node.

So, what can prevent an attacker from capturing and reverse-engineering a sensor, and using the same sand-boxing technique to keep a good copy of the sensor running in order to feed good answers to the challenge-response protocol initiated by the verifier?

The attacker will not be able to manipulate the sending data of the captured sensor by internal means as changes will be detected by the present SDI. Any attempt to copy only the WDIs will fail as no authenticated messages are going to be sent to the authentication authority leading to rendering the sensor useless. Any attempt to copy both images SDI and WDI in a different sensor or a more resource efficient device will lead in creating different Ram dumps, both in size and structure leading to non verification of the WDI and locking the sensor out of the network. The only way you can copy both SDI and WDI in a different device and get valid results is by copying them in the exact sensor model (hardware and software) thus leading to forensically sound proof results.

D. *Hardware Tampering Attacks*

If the malicious sensor has enough memory to maintain the original program blocks some of the previously stated attacks can succeed. However, as it has been previously defined upon initiation of our Framework a specific fingerprint of both the hash value of the WDIs and RAM dump has been stored.

Therefore, there is no room left in the sensor for the adversary to save and execute arbitrary code. The adversary may attach more memory to each sensor, but it will incur a considerable amount of hardware modification while the Ram dump check will identify the attack. Moving/Copying the isolates in different sensors, as far as hardware specs is concerned, or a personal computer will lead to rendering the sensor useless because of the RAM dump check inconsistencies.

E. *Encryption Algorithms*

TEA operates on 64-bit blocks and uses a 128-bit key. It has a Feistel structure with a suggested 64 rounds, typically implemented in pairs termed *cycles*. It has an extremely simple key schedule, mixing all of the key material in exactly the same way for each cycle.

TEA has a few weaknesses. Most notably, it suffers from equivalent keys—each key is equivalent to three others, which means that the effective key size is only 126 bits. TEA is also susceptible to a related-key attack which requires 223 chosen plaintexts under a related-key pair, with 232 time complexity.

Because of these weaknesses, we chose to focus on XTEA version of TEA's family cryptographic algorithms. XTEA is a 64-bit block Feistel network with a 128-bit key and a suggested 64 rounds. Several differences from TEA are apparent, including a somewhat more complex key-schedule and a rearrangement of the shifts, XORs, and additions.

Also, a third version **Corrected Block TEA** (often referred to as **XXTEA**) was designed, in order to correct weaknesses of the other previous two versions.

Our implementation is based on XTEA cryptographic algorithm, because it is more secure than the TEA and is less energy–harvesting than the XXTEA version, as described on table 1.

VI. Conclusion

In this paper, we have proposed a complete tamper-proofing framework based on physical security schemes, encryption, digital forensics and sand-boxing techniques which offer 1) prevention of manipulation, reverse-engineering, and reprogramming of sensors; 2) purely software based protection with/without tamper-resistant hardware; and 3) infrequent triggering of the verification.

Through securely executed isolates a verification of the Integrity of the program of each sensor device is performed successfully. For verification, it remotely calculates, 1) hash value of every WDI being executed, 2) RAM dumps and checks if the values match with those stored on IVDB depending on the SDI_ID. All communication is through encrypted channels.

Our security analysis has proven that the proposed framework effectively defeats different types of attacks while improving the state of the art in software based protection mechanisms, furthermore from the simulations conducted the protocol has proven to be low

References

[1] R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," Proc. Second USENIX Workshop Electronic Commerce, pp. 1-11, 1996.

[2] D.W. Carman, P.S. Kruus, and B.J. Matt, "Constraints and Approaches for Distributed Sensor Network Security," NAI Labs Technical Report, vol. 00, no. 010, Sept. 2000.

[3] R. Anderson, "Why Cryptosystems Fail," Comm. ACM, vol. 37, no. 11, Nov. 1994.

[4] S. Blythe, B. Fraboni, S. Lall, H. Ahmed, and U. Riu, "Layout Reconstruction of Complex Silicon Chips," IEEE J. Solid-State Circuits, vol. 28, no. 2, pp. 138-145, Feb. 1993.

[5] C. Collberg, C. Thomborson, and D. Low, "Breaking Abstractions and Unstructuring Data Structures," Proc. IEEE Int'l Conf. Computer Languages (ICCL '98), pp. 28-38, May 1998.

[6] C. Wang, J. Hill, J. Knight, and J. Davidson, "Software Tamper Resistance: Obstructing Static Analysis of Programs," technical report, Dept. of Computer Science, Univ. of Virginia, 2000.

[7] C. Wang, J. Hill, J. Knight, and J. Davidson, "Protection of Software-Based Survivability Mechanisms," Proc. Int'l Conf. Dependable Systems and Networks, pp. 193-202, July 2001.

[8] G. Wroblewski, "General Method of Program Code Obfuscation," Proc. Int'l Conf. Software Eng. Research and Practice (SERP), June 2002.

[9] M. Blum and S. Kannan, "Designing Programs that Check Their Work," J. ACM, vol. 42, no. 1, pp. 269-291, 1995.

[10] H. Wasserman and M. Blum, "Software Reliability via Run-Time Result-Checking," J. ACM, vol. 44, no. 6, pp. 826-849, 1997.

[11] F. Ergun, S. Kannan, S.R. Kumar, R. Rubinfeld, and M. Vishwanathan, "Spot-Checkers," Proc. ACM Symp. Theory of Computing (STOC '98), pp. 717-751,May 1998.

[12] D. Aucsmith, "Tamper Resistant Software: An Implementation," Information Hiding, pp. 317-333, Springer-Verlag, 1996.

[13] C.S. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation—Tools for Software Protection," IEEE, Trans. Software Eng., vol. 28, no. 8, pp. 735-746, Aug. 2002.

[14] B. Horne, L. Matheson, C. Sheehan, and R.E. Tarjan, "Dynamic Self-Checking Techniques for Improved Tamper Resistance," Proc. First ACM Workshop Digital Rights Management (DRM), pp. 141-159, 2002, 308 IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 4, NO. 3, MAY/JUNE 2005

[15] H. Chang and M.J. Atallah, “Protecting Software Code by Guards,” Proc. Second ACM Workshop Digital Rights Management (DRM), pp. 160-175, 2002.

[16] Taejoon Park, Kang G. Shin, “Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks”, IEEE Transactions on mobile computing, VOL. 4, NO. 3, pp. 297-309, May/June 2005.

[17] Squawk Project, http://labs.oracle.com/projects/squawk/ (Accessed 4 April 2011)

[18] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: Software-based ATTestation for Embedded Devices. In IEEE Symposium on Security and Privacy, pp. 272-282, May 2004.

[19] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *Proceedings* of the 2nd European Workshop on Security and Privacy in Ad Hoc and Sensor Networks, pp. 27-41, 2005.

[20] Y. Yang, X.Wang, S. Zhu, and G. Cao. Distributed softwarebased attestation for node compromise detection in sensor networks. In Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, pp. 219-230, 2007.]

[21] O. Landsiedel, K. Wehrle, and S. Gotz, “Accurate prediction of power consumption in sensor networks,” in Proc. 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II). IEEE Computer Society, 2005, pp. 37–44.

[22] B. L. Titzer, D. K. Lee, and J. Palsberg, “Avrora: scalable sensor network simulation with precise timing,” in Proc. 4th Int’l Conf. Information Processing Sensor Networks (IPSN ’05), 2005, p. 67.

[23] http://www.xbow.com (Accessed 10 April 2011).

[24] NIST, “Digital hash standard,” Federal Information Processing Standards Publication 180-1, April 1995

[25] L. Foley, S. Wilson, "Analysis of an On-line Random Number Generator", Trinity College Dublin, http://www.random.org, (Accessed 8 April 2011).

[26] A. Zaharis, A. I. Martini, L. Perlepes, G. Stamoulis, and P. Kikiras. 2010. Live forensics framework for wireless sensor nodes using sandboxing. In *Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks* (Q2SWinet '10), pp. 70-77

[27] D. Wheeler and R. Needham."TEA, a Tiny Encryption Algorithm." 1995. Springer-Verlag.

[28] S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. "Differential cryptanalysis of TEA and XTEA." In *Proceedings of ICISC 2003*, pp. 402-417, 2003b.

[29] E. Yarrkov. Cryptanalysis of xxtea. Cryptology ePrint Archive, Report 2010/254, 2010. http://eprint.iacr.org/ (Accessed 27 May 2011).

[30] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. 2002. Wireless sensor networks: a survey. *Comput. Netw.* 38, 4 (March 2002), pp. 393-422. DOI=10.1016/S1389-1286(01)00302-4

[31] Feistel H. "Cryptography and Computer Privacy", Scientific American Vol. 228 No. 5, pp. 15-23, 1973