A Lightweight Authentication Protocol for Secure Communications between Resource-Limited Devices and Wireless Sensor Networks

Piotr Książak, Department of Computing, Letterkenny Institute of Technology, Letterkenny, Ireland

William Farrelly, Department of Computing, Letterkenny Institute of Technology, Letterkenny, Ireland

Kevin Curran, Faculty of Computing & Engineering, Ulster University, Londonderry, UK

ABSTRACT

The number of Resource-Limited Wireless Devices utilized in many areas of IT is growing rapidly. Some of the applications of these devices pose real security threats that can be addressed using authentication and cryptography. Many of the available authentication and encryption software solutions are predicated on the availability of ample processing power and memory. These demands cannot be met by the majority of ubiquitous computing devices, thus there is a need to apply lightweight cryptography primitives and lightweight authentication protocols that meet these demands in any application of security to devices with limited resources. A security framework is presented here that combines aspects of the Gossamer protocol and the Scalable Encryption Algorithm (SEA) to provide an implementation of inter-device security. The Gossamer Protocol is additionally used as a means of exchanging session keys for use with the SEA encryption protocol. Our system performed well with the code space requirements smaller than 600 bytes (excluding shared libraries) and a performance of 27 milliseconds per one 96-bit block of data.

Keywords: *Gossamer Protocol, Lightweight Authentication Protocol, Lightweight Cryptography, Resource-Limited Devices, Scalable Encryption Algorithm, Secure Communications, Wireless Sensor Networks*

DOI: 10.4018/IJISP.2014100104

1. INTRODUCTION

The number of Resource-Limited Wireless Devices utilized in many areas of the IT industry is growing rapidly. This growth rate is expected to rise even higher when RFID transponders begin to replace Barcodes on a larger scale. Some of the applications of these devices pose a security threat which can be addressed using cryptographic techniques. Most of the currently used cryptographic solutions are predicated on the existence of ample processing power and memory. These demands cannot be met by the majority of ubiquitous computing devices, thus there is a need to apply lightweight cryptography primitives that meet security demands when considering devices with low resources.

A Risk Analysis of threats associated with the usage of Wireless Sensor Networks or RFID systems for the item-level stock control and temperature monitoring include the following:

- **Tag/Sensor cloning:** A serious threat related to the counterfeiting of medicines with a high likely-hood of occurrence (Juels, 2005). Can be addressed with a strong encryption and authentication system;
- **Tag/Sensor tracing:** A threat related to unauthorised Track & Trace of a Sensor/Tag movement throughout a given area, which has negative privacy implications. It can be addressed with a proper Authentication system that does not allow the disclosure of a Tag's/Sensor's unique ID;
- **Data Eavesdropping:** Unauthorized retrieval of sensor/tag data. A strong encryption algorithm provides a counter-measure to this threat;
- **Denial of Service attack:** Affects the operation of the entire network or a group of Tags/Sensors. The likely-hood of occurrence can be regarded as medium. Such an attack would require appropriate hardware and in-depth knowledge of the radio protocol used. A proper Authentication system provides counter-measures to this threat;
- **Rogue-Data Injection:** An adversary can inject malicious data into the network causing improper configuration of the sensors for example. The probability of occurrence can be low as this kind of attack is not valuable to an adversary in most cases. A Mutual-Authentication system prevents accepting rogue data from unknown sources;
- **Cryptanalysis Attack:** Secret key discovery through a cryptanalysis attack on the authentication and/or encryption system's secret data. Such an attack compromises the whole security and leads to a full disclosure of all data. The likelihood of such an event is very low if the encryption key-space is large enough to prevent brute-force attacks (assumes unbreakable algorithm).

Typically, the application of security to wireless networks, such as the Wi-Fi Protected Access specification (Wi-Fi Alliance, 2003), requires complex mathematical computation and significant protocol data overhead. Since these requirements cannot be fulfilled by the types of Resource-Limited Devices used in Wireless Sensor Networks (WSN) and Radio Frequency-Identification (RFID) systems due to the constraints imposed by limited computational power, limited memory size and the requirement for low power consumption (Akyildiz et al. 2002), there is a need to provide a lightweight security mechanism that can be implemented within device specifications. The primary aspects of the security of data exchange are mutual authentication, confidentiality, integrity and availability (Menezes et al., 1997). Another important aspect of security especially in the context of Wireless Sensor Networks, is Data Freshness which ensures that the data received is fresh and the adversary cannot replay old messages. Weak data fresh-

ness ensures the order of messages, and strong data freshness allows additionally for the delay of the message estimation.

Our work examines the nature of inter-device security in the context Wireless Resource-Limited Devices by decomposition; splitting it into the sub-problems of authentication and encryption. These sub-problems address the key security issues identified in the literature (Schneier, 1996; Menezes et al., 1997; Mollin, 2007; Ranasinghe & Cole, 2008, Karlof et al., 2004).

2. AUTHENTICATION IN WIRELESS RESOURCE-LIMITED DEVICES

Mutual Authentication is a process of ensuring that all parties taking part in the communication can validate each other's identity. An intruder should not be able to masquerade as someone else (Schneier, 1996). The physical properties of the radio frequency communication channel (the ease of eavesdropping), computational efficiency and power consumption constraints (Akyildiz et al., 2002) impose limitations on the range of authentication protocols which can be taken under consideration. The problem of authentication in the context of networking resource-limited devices is explained in the following sections.

2.1. Authentication with Resource-Limited Devices

The issue of Authentication in the networking of wireless resource-limited devices was given little attention until RFID systems became popular. As RFID systems are expected to be widely used for item-level tagging of consumer products, the Electronic Privacy Information Center (EPIC) take a keen interest in privacy and security (Juels, 2006). There is a need for the application of lightweight cryptographic primitives and protocols in the development of solutions for RFID (Sarma et al., 2003). Major threats to consumers are tracking (traceability) and inventory privacy (Juels, 2006). Under normal operating conditions, a tag reader will interrogate and read all tags in its proximity. Thus an unsecured RFID tag reveals its unique identifier in the absence of authentication between tag and reader. Any reader compliant with a given RFID specification is able to interrogate and identify the tag. In consequence, a person carrying a given tag, e.g. in a shopping bag, can be tracked around an area by a series of purposely located interrogators without the person's consent. If an unsecured tag conforms to the Electronic Product Code (EPC) specification, it also carries a unique identification of the item to which it is attached. This poses a threat in respect of itemising the contents of say, a shopping trolley, and identifying an individual's purchasing patterns (Leong et al., 2006).

Privacy, although drawing most of the attention, is not the only set of issues associated with the absence of an authentication mechanism. RFID systems and Wireless Sensor Networks are facing the threat of data forging and manipulation. Using commonly available equipment, an adversary can easily inject messages (Perrig et al. 2002), leading to false sensor readings. The majority of commonly used authentication mechanisms rely heavily on computationally intensive mathematical techniques requiring the manipulation of, for example, long keys. Resource Limited Devices share a number of constraints which in the case of RFID systems make the implementation of computationally intensive mathematical routines impossible due mainly to significant reduction in processor power and the absence of sufficient memory to store lengthy keys. A secondary argument is that an increase in the number of logic gates implementated on an Integrated Circuit dramatically increases the overall price per tag (Sarma 2001). Although Wireless Sensor Networks (WSNs) use more capable hardware they are also tightly constrained by power limitations. WSN sensor battery life requirements force limited usage of the CPU and the radio bandwidth. Additionally, a node in a WSN is imbued with many tasks such as the Analogue

to Digital Converter (ADC) readings interpretation, radio protocol handling, reprogramming behaviours etc., thus the code space left for security mechanism implementation is very limited. In recent years the field of lightweight security has emerged rapidly and is offering solutions mostly for RFID but also covering the area of WSNs. A number Ultra-Lightweight Authentication protocols have been developed which mainly target RFID but additionally, promise ways of providing a resource-saving authentication mechanism for Infrastructure Wireless Sensor Networks due to their computational simplicity and small data overhead (Juels, 2005; Chien, 2007; Peris-Lopez et al., 2009; Lee et al., 2009).

There are a number of attacks such as passive attacks, where the adversary eavesdrops on transmitted messages. In this case we assume that the adversary is not able to alter the messages or inject new ones and active attacks, where the adversary is able not only to eavesdrop the communication but also inject new messages or alter and replay the previous ones. Physical invasive attacks are where the adversary has a physical access and toolset required to access the device's circuitry and for example read the EEPROM memory contents. While the physical access attack threat cannot be fully negated by a protocol, it has to be noted that the results of such an attack have to be minimised: a compromise of one tag/node should not compromise the security of other nodes/tags. It should not be possible to crack a node's previously recorded and stored communications with a recently discovered key. This requirement is known as data freshness. It is a requirement of RFID systems and Wireless Sensor Networks that it should not be possible to track nodes without express authority to do so. This is known as a Traceability (ID disclosure) Attack (Juels 2006). The attack is performed to obtain a device's unique ID number which can be further used to track the device's movements using an appropriate RF transceiver. The ID disclosure attack may be performed using passive or active methods and typically targets the authentication protocol as the ID has to be transferred in one of the protocol's messages.

The success of full disclosure attack means that the entire security of the protocol has been compromised and all secret information used during the protocol flow is disclosed. This allows the adversary to fully impersonate (spoof) one of the devices taking part in the communication and effectively 'Clone' one of the nodes/tags. Typically a full disclosure attack requires active methods, but weak authentication protocols can be fully compromised using passive eavesdropping of consecutive rounds only (Bárász et al., 2007a). A de-synchronization attack is one of the most serious threats for an authentication protocol that is used in wireless networks. Synchronization means that both parties are aware of the status of the protocol and are able to continue executing the protocol with a normal flow. A de-synchronization attack breaks the protocol by altering the state of one (or both) of the parties authenticating each other in a way which renders further phases of the protocol not executable (Li & Wang, 2007). This kind of attack may effectively cause a denial-of-service of one or more nodes in the network.

2.2. Infrastructure Wireless Sensor Network (IWSN) Protocols

Ultralightweight protocols, which were designed for low-cost RFID systems, rely on minimalistic cryptography techniques and provide a viable alternative for securing a heavily constrained Infrastructure Wireless Sensor Network (IWSN) with minor modifications. Other more computationally intensive schemes designed specifically for Wireless Sensor Networks (although filtered by the specific requirements of IWSN) or advanced RFID systems are also discussed.

2.2.1. M²AP: Minimalist Mutual-Authentication Protocol

M²AP (Minimalist Mutual-Authentication Protocol) is an Ultralightweight Mutual Authentication Protocol (UMAP) (Peris-Lopez et al., 2006c). M²AP uses an index-pseudonym (IDS) to avoid

disclosing device's ID which prevents the privacy issues (Traceability and Inventory) associated with both RFID and some applications of WSN, for example Wireless Body Sensor Networks (WBSNs). The IDS (96-bit long) is effectively an index to a record in a database storing tag-specific information. Each tag stores a key consisting of four concatenated 96-bit long parts (K = K1 II K2 II K3 II K4). It is assumed that the communication link between a reader and the back-end database is secure. The protocol's author provided a security analysis of the proposal in terms of resistance to ID disclosure, Man-in-the-middle, replay attacks and Data Integrity assurance. The anonymity of the tag (ID hiding) is ensured by the usage of an index-pseudonym (IDS). The Data Integrity is guaranteed by the IDS and four sub-keys - the attacker would have to be able to modify these values on both the database and the tag, otherwise even a single bit manipulation would stop the protocol execution. The mutual authentication mechanism based on two random numbers refreshed with every iteration of the protocol renders the Man-in-the-middle attack impossible. The IDS and sub-keys updating mechanism aims to prevent Replay Attacks.

A passive attack (eavesdropping only) against the M²AP which is able to retrieve the IDS and all sub-keys by eavesdropping over a few consecutive runs of the protocol is possible (Bárász et al., 2007b). Weaknesses in M²AP include the usage of the bit-wise operations and the modulo 2^{96} addition which only implies that every bit affects only bits which are to the left of it and the least significant bit is independent of any other bits. Such operations are called triangular functions or T-functions and per Klimov and Shamir "A *T-function* is a mapping in which the *i*-th bit of the output can depend only on bits 0,1,..., *i* of the input"(Klimov & Shamir, 2004). Another weak aspect is the OR and AND operations used in messages B and D which can help to derive n1 and n2 values with the help of set and reset bits of IDS. The attacker can learn the ID, K1, K3, n1 and n2 after eavesdropping only two consecutive rounds of the M²AP which already allows for Traceability of the tag. K2 and K4 sub-key discovery requires eavesdropping more rounds but provides the attacker with the ability to impersonate the Tag or the Reader.

2.2.2. EMAP: An Efficient Mutual-Authentication Protocol

The EMAP Protocol (Peris-Lopez et al., 2006a) was developed as a result of weaknesses discovered in the M²AP Protocol (Bárász et al., 2007b). EMAP is similar to M²AP: It has the same four stages and uses IDS and four sub-keys K1-K4. The only changes which were applied were the mathematical operations used to construct sub-messages A, B, C, D, E and the formulas for updating the IDS and four sub-keys. The IDS updating formula was supposed to have better statistical properties than the M²AP as the entire number use bit-wise XORed with a random number n2. The key updating formulas now contain a parity function $\left(F_{p(X)}\right)$ which divides the 96-bit number into 24 4-bit blocks, calculates and outputs a parity bit for each block.

A de-synchronization attack and full disclosure attack are possible on LMAP (Li & Deng, 2007). As both protocols rely on a synchronization of IDS and keys stored on a tag and the back-end database therefore a full round of the protocol has to take place in order to keep synchronization on both sides. A man-in-the-middle de-synchronisation attack can be performed by changing the message C – by intercepting message (A II B II C) and XORing sub-message C with a series of zeros excluding the least significant bit set to 1 and forwarding the set of messages to the tag. The tag can still authenticate the reader as A and B remain unchanged, but it will get the wrong n2 number. Despite this the protocol will continue and the tag will reply with incorrect D and E messages; however, the reader will not be able to discover changes in D and will accept in all cases. It was shown that there is a 75% chance on average that the reader will accept an incorrect value E and update its database using original n2. The tag will do the same

using incorrect n2 and both devices will lose synchronization. The full disclosure attack is based on a stateless nature of the tags - there is no way to save the state of the protocol execution on a tag. The attack consists of four stages, the first three of which are performed on a single protocol run and disclose all secret values apart from K2, K4 and the tag ID. The fourth stage requires approximately $(\log_2 m - 1)$ runs to fully disclose tag's ID (m-bits long).

2.2.3. LMAP: A Real Lightweight Mutual Authentication Protocol

LMAP addresses weaknesses discovered in M^{2}AP and EMAP (Peris-Lopez et al., 2006b). LMAP and EMAP share some similarities including the same size of the IDS and the same size and number of sub-keys. However, the Tag to Reader message (previously consisting of sub-messages D and E) was reduced only to a single message D. Weaknesses were discovered in both the LMAP and the M^{2}AP protocols (Li & Wang, 2007). The vulnerabilities and possible attacks are similar to the EMAP security flaws (Li & Deng, 2007). Again, the main issue is related to the fact that the tag is not able to verify if the reader successfully received and verified message D, which may lead to a protocol de-synchronization. The de-synchronization attacks are practically identical to the one proposed earlier: message C alteration and messages A&B alteration attacks performed by XOring the message with zeros and one as the least significant bit. The probability of the success of the first attack remained at 50%. The full disclosure attack is slightly more difficult than in the case of the M^{2}AP protocol. The attacker has to obtain the current IDS of the tag and then try all possible (A II B II C) messages by sending them to the tag and changing the j-th bit in A and B at each try. This reveals the n1 random number value and allows the calculation of K1 and K2. The rest of the secret values can be discovered by interacting with the reader and the tag one more time and then derived from the known sub-message creation equations and a simple algorithm described in (Li & Wang 2007). Several countermeasures were proposed, the most interesting one proposes a tag status storage mechanism preventing de-synchronization attacks: an additional status bit on the tag indicating whether a protocol has been successfully completed and two additional 96-bit memory spaces for storing n1 and n2 values used in the last protocol run. A Similar mechanism was included in (Peris-Lopez et al. 2006b) as a LMAP+ extension. Dispite these countermeasures, LMAP and M^{2}AP are still susceptible to de-synchronization and full disclosure attacks (Chien & Huang 2007). With LMAP and M^{2}AP, the attacker can flip some bits without being noticed by the reader or the tag so the protocol round would complete and both sides would update the IDS and keys with different n1 and n2 random numbers (Chien & Huang, 2007). A fully passive full disclosure attack is also possible against LMAP, which requires only eavesdropping a few (about 10) consecutive rounds of the protocol (Bárász et al. 2007a)[1]. Another weakness of the protocol is related to triangular functions properties (weak propagation of bits from left to right) (Bárász et al. 2007b).

2.2.4. SASI: Strong Authentication and Strong Integrity

The family of UMAP protocols (Peris-Lopez et al., 2006b) influenced the SASI (Strong Authentication and Strong Integrity) protocol (Chien, 2007). The tag has a unique 96-bit ID and pre-shares an index-pseudonym (IDS) and two keys K1 and K2 with a back-end database accessible by the reader (secure link assumed). In order to resist de-synchronization a state-verification has been employed: the tag stores two sets of (IDS, K1, K2) – the old values and the potential new values. In each protocol instance the reader may probe the tag twice: the first time the tag replies with its potential new IDS and if it was not found it may probe the tag again and this time the tag will use the old IDS value. The protocol flow is also similar to UMAP. It is secure

against de-synchronization attacks, ID disclosure attacks and it should provide privacy, anonymity, mutual authentication and forward secrecy (keeping the past communication secure even if a tag is compromised later) while retaining the ultra-lightweight properties (Chien, 2007). It requires a message length of 4L[1] and the total memory size on a tag of 7L as opposed to 6L in UMAP family protocols. There have been no published successful passive attacks against the SASI protocol using Hamming rotation function. However, several active attack possibilities were discovered. De-synchronization attacks on the SASI protocol include targeting the anti-de-synchronization mechanism of the SASI protocol: the possibility of re-trying the communication with the old IDS in case the next-possible IDS was not found in the database (Sun et al., 2008). A denial-of-service and ID disclosure attack is also possible (Cao et al., 2009). A de-synchronization, ID disclosure and a full disclosure attack against the SASI protocol is also achievable (D'Arco & De Santis, 2008).

2.2.5. Gossamer Protocol

The Gossamer Protocol (Peris-Lopez et al. 2009) is a recent entrant in the field of lightweight cryptography. It was built on the premise that many of the weaknesses in other approaches which use simple bitwise operations like AND, OR, XOR and modulo 2^{96} addition are T-functions (Klimov & Shamir 2004), and thus suffer from weak propagation of bits from left to right. Another weakness is the bias in the probability (75%) of obtaining a bit '1' when using bitwise AND operation. The Gossamer Protocol that is largely similar to the SASI protocol in general concept: each tag has a static identifier (ID), an index-pseudonym (IDS) and two keys K1 and K2 in memory. Additionally each tag is required to store two sets of the tuple (IDS, K1, K2): old value and the potential next value. It is assumed that the only mathematical operations that will be used are bitwise XOR, addition modulo 2^m and left rotation function $Rot(x,y)$. The rotation function performs a circular shift on the value of x by $(y \ mod \ N)$, positions to the left for a given N (96 in case of the EPC RFID). The most computationally expensive operation of generating two random numbers required in each protocol run is designed to be done on the reader side. An additional security layer is added with a lightweight function called *MixBits* (Hernandez-Castro et al. 2006) and uses only bitwise right shift. The protocol executes in three stages: tag identification, mutual identification and updating phase. The protocol requires exchanging four messages between the reader and the tag. Hello message length is not specified, the IDS and D messages are 96-bits long and the concatenated A II B II C message consist of three 96-bit long sub-messages. A total of 384 bits (excluding Hello message) needs to be transmitted during one protocol run. The Storage Requirements on the tag side are limited to 7 times the key-length (96-bits in the original specification) to hold two IDS, K1, K2 tuples and the static identifier ID. Each database record is required to story only one IDS, K1, K2 tuple and the static ID. The Gossamer Protocol prevents attacks as follows:

- **ID Disclosure Attack:** The notion of an index-pseudonym (IDS) and private keys K1 and K2 changed for every authentication session prevents disclosure of the unique identifier (ID) of the tag;
- **Full Disclosure Attack:** The secret data (ID, K1, K2) is always scrambled using two random numbers and sum, Mixbits and Rot functions before being transmitted over the wireless link;
- **De-Synchronization Attack:** Each tag stores (IDS, K1, K2) tuples used in a previous protocol run. In case of an unsuccessful update on the reader side in the last stage of the

protocol (message D) the tag can be still identified using old values. The result is that both the tag and the reader can recover their synchronized state.

The requirement of *Data Freshness* is fulfilled by updating secret values K1, K2, n1 and n2 at each protocol run.

There are a number of attacks against the protocol (Ahmed et al., 2010). One is feasible if both random numbers n1 and n2 were equal to zero allowing the discovery of all secret values after eavesdropping two consecutive runs of the protocol. Another attack concerns a case where both K1 and K2 values are equal to zero, which leads to disclosure of all secret values during a single authentication round. A modification was proposed (Ahmed et al., 2010) however it has a flaw in that it renders the extraction of n1 and n2 impossible.

An attack on the original Gossamer protocol is feasible if both random numbers n1 and n2 were equal to zero permitting the discovery of K1 and K2 after eavesdropping two consecutive runs of the protocol. The values n1 and n2 are known to the reader. In A, $ROT\left(IDS + K1 + \pi + n1, K2\right) + K1$ is rotated by n2 by the reader and likewise in B, $ROT\left(IDS + K2 + \pi + n2, K1\right) + K2$ is rotated by n1. Messages A & B are exchanged with the tag. The tag's job is to extract the values n2 and n1 from messages A and B and to perform the appropriate inverse rotation to verify the remainder of the contents of messages A and B. However, in this modification, the tag is not aware of the value n1 or n2 and therefore cannot perform the inverse rotation to retrieve $ROT\left(IDS + K1 + \pi + n1, K2\right) + K1$. This is a flaw that will not permit the completion of authentication. Another modification (Ahmed et al., 2010) concerning the MixBits function also has a weak effect on overcoming the problem of both random numbers n1 and n2 equalling zero. In the original Gossamer protocol, the mix-bits function exists during the creation of the new IDS and Key values.

Where X and Y are the input 96-bit numbers and Z is the final result of the MixBits function. The weakness identified (Ahmed et al., 2010) is that if both of the MixBits input values (n1 and n2 in the first run) are equal to 0 then the result of the function is also equal to 0. As a result all transformations are dependent on the Key values, the IDS and Pi. This weakens the effective security of the Gossamer Protocol. (Ahmed et al., 2010) proposed a modification however in a case where both n1 and n2 numbers are equal to zero, then the result of the MixBits function will be always the sum of numbers 1 to 32 which is 528. The proposed attack on the MixBits functions where n1 and n2 are 0 has been rectified but now the first attack proposed can be still performed but using the value of 528 instead of 0 at the first call of the MixBits function within the Protocol (n3 calculation) and the result can be applied to the subsequent formulae to generate keys and messages. Temporarily as a solution to the first attack it is recommended not to allow both random generated numbers to hold a value of zero at the same time. This verification should be performed by the PRNG function before the values are forwarded to the reader. An altered Gossamer Protocol is suitable as a mechanism for authenticating Resource Limited Devices. The reader-tag relation is close to the master-slave one in the Infrastructure WSN scenario. The main difference is the fact that a RFID tag is triggered by the reader, where in the IWSN all slaves will periodically initiate the communication. This difference is not significant in terms of the Gossamer specification as the 'Hello' message send by the reader to initiate the communication does not carry any protocol-specific data, thus can be discarded without any effect.

2.2.6. Ultralightweight RFID Protocol with Mutual Authentication (UMA-RFID)

UMA-RFID is similar to the Gossamer specification but simplified to use only bitwise operations (XOR, OR, AND) and a left bitwise rotation function ROT (Lee et al. 2009). Each tag contains a static identifier ID, pseudonym called the dynamic temporary identifier (IDT) and a secret key (K). All variables are 128-bits long and shared between the tag and the back-end database accessible by the reader (secure channel assumed). The reader is assumed to be capable of generating random number (N). The protocol consists of the Authentication Phase and the Update Phase:

- **Authentication Phase:** The reader sends a request message and the tag replies with a temporary identifier (IDT). The reader searches the database to find a secret key K_i corresponding to the IDT received, generates random number N_i and calculates messages A_i and B_i as follows:

$$A_i = K_i \oplus N_i$$
$$B_i = ROT(K_i, K_i) \oplus ROT\left(N_i, N_i\right)$$

The messages are concatenated and sent to the tag. Upon receiving these messages the tag obtains N_i from message A_i and calculates message B_i' in the same way as the reader previously. Then message B_i' is compared with B_i. If they are the same then the reader is authenticated and the tag generates reply message C_i as follows:

$$C_i = \left(K_i \vee ROT\left(N_i, N_i\right)\right) \oplus (ROT(K_i, K_i) \wedge N_i)$$

The message is sent to the reader and the reader calculates a local copy and verifies the correctness. After successful verification the tag is authenticated:

- **Updating phase:** The tag performs this phase after authenticating the reader. The updating on the reader side is done upon successful authentication of the tag. Both sides use the following equations to update IDT_{i+1} and K_i:

$$IDT_{i+1} = K_i \oplus ROT\left(N_i, N_i\right)$$
$$K_i = ROT(K_i, K_i) \oplus N_i$$

There are serious weaknesses in the scheme which can lead to ID Disclosure, Full Disclosure and De-Synchronization attacks (Peris-Lopez et al., 2008). The most significant full disclosure attack allows cloning of the tag to be performed after eavesdropping of only two consecutive runs of the protocol and requires only computing XOR among some of the messages transmitted over the radio channel.

2.2.7. SQUASH

SQUASH (short for SQUare-hASH) is an authentication mechanism based on a challenge-response scheme and Message Authentication Code (MAC) specifically for Resource Limited Devices (Shamir, 2008) such as RFID tags. The challenge-response scheme allows tag-to-reader authentication and does not address the ID disclosure issue. A strong one-way hashing function (H) is performed by the tag upon receiving a random challenge message (R). The reader shares the secret key S and performs the same calculation upon receiving the MAC to validate if a tag is legitimate. Most of the standard one-way hash functions such as SHA-1 (Eastlake & Jones, 2001) are primarily designed to be collision resistant as their main area of usage concerns digital signatures. The requirement for collision resistance typically adds complexity to the algorithm. Since a collision is not a security threat in a challenge-response scheme, the author proposed an algorithm based on the Rabin encryption scheme (Rabin 1979). In the Rabin scheme the ciphertext (c) is computed as $c = m^2 \left(mod\ n\right)$, where (m) is a message and (n) is a product of at least two unknown prime factors. Shamir has shown how the calculation can be simplified using a step-by-step process that has no adverse effects on the strength of the security and has proposed a hardware implementation using mixing function (M) applied to the secret and challenge (S, R) and then the SQUASH function SQUASH(M(S,R)). The proposed SQUASH-128 hash function uses a modulus $2^{1277} - 1$, a 64-bit key S and a 64-bit challenge R to produce a 32-bit response. There is a vulnerability in this scheme with the key recovery attack known as "known random coins attack" against the Rabin scheme using 1024 chosen challenges (Ouafi & Vaudenay 2009). The "known random coins attack" allows an adversary to request many encryptions of the same plaintext and in consequence get the random coins. The attack is only effective if a linear mixing function is used, thus the security of SQUASH is still regarded as strong, assuming that a non-linear mixing function is used.

2.2.8. SPINS: Security Protocols for Sensor Networks

Perrig et al. propose a security mechanism consisting of two blocks: Secure Network Encryption Protocol (SNEP) and μTESLA (Perrig et al. 2002). SNEP's security goals are data confidentiality, mutual authentication and the evidence of data freshness[2]. μTESLA provides a mechanism for an authenticated broadcast. SNEP possesses a low communication overhead of only 8 bytes per message. SNEP ensures semantic security[3] using two counters C_A and C_B shared by the communicating nodes. These counters are further used by the block cipher in counter mode. Counters do not have to be attached to messages but there is a mechanism of counter synchronization. The mutual authentication is achieved through the usage of a MAC function. Both communicating nodes A (sender) and B (receiver) share a master secret key X_{AB} used to derive keys through a pseudorandom function. It has to be noted that the authors advised deriving different key sets for MAC and encryption. Each key set consists of two keys - one for each direction of the communication. Upon receiving the message and MAC verification the node A is sure that the node B generated the message using the cryptographic nonce supplied in a request message. SNEP messages require synchronized counters on both sides of the communication. If the synchronization is lost for example due to lost messages, counter values can be re-synchronized through specific messages.

Another security protocole is μTESLA which was based on a TESLA protocol providing a mechanism of an authenticated broadcast (Perrig et al. 2001). This scheme achieves asymmetry through a delayed disclosure of symmetric keys rather than using computationally expensive

Public Key Cryptography. The TESLA proposal is not suitable for implementation within a constrained devices environment. In order to adapt it for the Wireless Sensor Networks the following issues were addressed:

- TESLA authenticates the initial packet with a digital signature. The computation of a digital signature is too expensive on sensor nodes so μTESLA uses only a symmetric mechanism;
- Standard TESLA discloses the key for the previous intervals with every packet. Since this generates too much overhead μTESLA discloses the key once for each pre-defined epoch;
- Sensor nodes are not able to store an entire one-way key chain in the memory. This is addressed in μTESLA by limiting the number of authenticated senders.

The μTESLA requires that all receiving nodes are loosely time synchronized with the base station. In order to send an authenticated broadcast, the base station computes a MAC using the packet and a key which is secret at that point in time. The receiving node stores the packet in the buffer in order to validate its authenticity later when the base station broadcasts the verification key. Each MAC is a key of a key chain, generated by applying a one-way hash function. A successive key is generated by applying the hash function on the previous key. The time synchronization can be achieved by the means of the SNEP protocol. The encryption algorithm is not specified in SNEP, or which one-way hash function should be used by μTESLA, or indeed which Random-number generation should be used in both protocol blocks but example functions for the experimental implementation are provided. In order to tackle the issue of limited code space and RAM size all cryptographic primitives are based on a modified subset of the RC5 encryption algorithm (Rivest 1995). The μTESLA's disadvantage is the need for an initial unicast-based parameter distribution however is has been addressed in the Multi-Level μTESLA specification (Liu & Ning, 2004). The scheme provides a way to predetermine and broadcast the initial parameters. Additionally, Multi-Level μTESLA introduces a mulit-level key chain scheme which removes the need for very long key chain. The key chain commitment distribution mechanism improves the survivability of the scheme against message loss and Denial Of Service (DOS) attacks. There are efficiency and key management issues in SNIPS (Liu & Ning, 2004; Yu-Long et al., 2007; Hegazy et al., 2007). Both of the SPINS schemes suffer from using Pseudo-Random Number Generator (PRNG) engines not only on the base station side but also on the sensors. The sensor nodes may draw random numbers from the actual sensor readings (Perrig et al. 2002). However, the Analog-to-Digital Converters (ADCs) which are sometimes only 8 or 10-bit wide may not be able to provide random values in a magnitude large enough for cryptographic usage. Thus a resource expensive PRNG functions need to be implemented within a limited sensor node code space. Due to a high computational overhead on the sensor node side, the SPINS implementation was not considered during this project.

2.3. Encryption

Cryptography is the art and science of keeping messages secure (Schneier, 1996). A message (referred to as a plaintext) undergoes a process of hiding its substance (encryption) and converting it into a non-meaningful gibberish (ciphertex) that can be sent over an insecure communication channel. The process of retrieving a plaintext from a ciphertext received is referred to as decryption. The general rule followed in modern cryptography states that the security of the system cannot rely on the secrecy of its components (security by obscurity) – the secrecy must reside entirely in the encryption key. This principle was stated by Auguste Kerchoff in the nineteenth century (Menezes et al. 1997), who assumed that a cryptanalyst has a complete knowledge of the algorithm and implementation. An algorithm that has its security based on keeping its foundations

secret is called restricted. Such a security system can be compromised through an information leak, reverse engineering, etc. Quality control and standardisation cannot be maintained. The most common type of cryptography is the Secret-Key Cryptography (symmetric cryptography), where a message 'M' gets encrypted with encryption function E, using a key 'k' to generate a ciphertext 'c'. Therefore, c = E(k,M). The decryption function D should provide a way to recover the plaintext 'p' using the shared secret 'k', such that p = D(k,c).

Resource-Limited Devices (RLDs) are highly constrained in terms of available memory and processing power. The reference platform nRF9E5 does not provide any hardware support for any encryption algorithm, thus the entire mechanism needs to be implemented in software. The following characteristics are used to evaluate possible encryption algorithms:

- The code-space required for the implementation is limited (algorithm simplicity) and will be used as one of the metrics;
- The algorithm should be optimized for 8-bit word size;
- The expected data payload size is limited, thus the resource efficiency of the algorithm will take precedence before the data throughput;
- The single data payload size is limited to 24 bytes on most occasions;
- It may be not possible to implement a random number generator on the node side due to general hardware and execution time constraints;
- Thanks to the proposed authentication algorithm, key management problem may be resolved through a re-use of the authentication key for the purpose of a session key fed into a symmetrical block or stream cipher.

The most security critical aspect of wireless sensor operation is the reconfiguration of the nodes. An attack enabling an adversary to alter the control messages may lead for example to Denial Of Service (DOS) attacks affecting the entire network of sensors. It is assumed that the authentication system will guarantee frequent session key changes for the purpose of maintaining the data freshness. In consequence, the control messages have to remain safe for a relatively short period of time, until the next session key is exchanged. In most IWSN applications the data transferred by sensors will not be valuable to an attacker and will not require infinitely long secrecy. Given the analysis above, the encryption algorithm may be based on relatively short encryption keys.

Attacks can be generally divided into two categories - passive and active attacks. Passive attacks concern monitoring the communication channel and gathering data (eavesdropping) but not altering it in any way. Wireless Sensor Networks are especially prone to passive eavesdropping. Attacks requiring alteration of the transmitted ciphertext or alteration of the computation in a device are referred to as active. These types of attacks commonly target specific protocol implementations of the security system rather than cryptographic algorithms on their own. A special case of an active attack is a physical invasive attack - the adversary has a physical access and toolset required to access the device's circuitry and for example read the EEPROM memory contents. Since a full protection against this types of attack requires advanced hardware (such as a sensor case destroying the EEPROM chip during opening), it is assumed that such attack cannot be prevented. The consequences of the physical invasive attack have to be limited in such a way that the security of the entire system is not compromised when a single node's key is revealed and the past communication remains safe. This issue introduces a requirement to maintain session keys unique to each of the sensor nodes. Cryptography for low cost embedded devices has not been given much attention until recently such as the search for solutions easily implementable in hardware (Bogdanov et al. 2007, Eisenbarth et al. 2007, Poschmann et al. 2007)

or focused on a software implementation efficient on low resource microcontrollers (Standaert et al. 2006, Wheeler & Needham 1994).

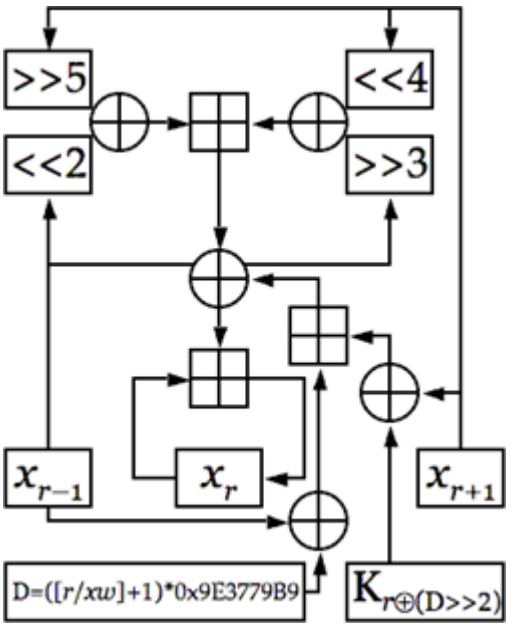
2.3.1. Tiny Encryption Algorithm (TEA) Family

The Tiny Encryption Algorithm (TEA) (Wheeler & Needham 1994) was the initial proposal of a family of algorithms (chronologically): XTEA and Block TEA (Needham & Wheeler 1997) and XXTEA (Wheeler & Needham 1998). The main principle behind the TEA algorithm design was the simplicity of the implementation and the ease of translation to many programming languages (including Assembly). The initial proposal was a block cipher operating on 64-bit blocks with 128-bit key. Each of the identical 64 rounds of the algorithm uses only logical AND, OR, as well as bit-shift operations and addition/subtraction $\text{Mod } 2^8$. The sample C-language source code consisted of less than 10 lines. The authors favoured large number of iterations over the complexity of the code. The set up time is relatively short and there is no need to store any Look-Up-Tables (LUTs) in the memory.

The first weakness discovered in TEA was the fact that each key is equivalent to three others which effectively reduces the key size to 126 bits. This vulnerability was used to construct an attack against Microsoft's Xbox game console, which uses TEA as a hash function (Russell 2004). Since the initial proposal in 1994 several attacks were published, for example a Key-schedule cryptanalysis (Kelsey et al. 1997) and Related-key cryptanalysis (Kelsey et al. 1997). Wheeler & Needham addressed the issue mentioned above when proposing Block TEA and XTEA algorithms. The key schedule was revised and other computations (bit-shifts, XORs and additions) were rearranged to introduce the key material more slowly. The XTEA algorithm and it's block version Block TEA also suffer from weaknesses discovered by shortly after publication by Saarinen (Saarinen 1998): slow diffusion in the decryption direction exploited by chosen plaintext attack. Several other cryptanalysis attempts were also published in (Andem 2003, Hong et al. 2004, Ko et al. 2004, Lu 2009, Moon et al. 2002). The slow decoding propagation pointed by Saarinen was addressed by Wheelar & Needham in their XXTEA proposal as a short amendment to the Block TEA (Wheeler & Needham 1998). XXTEA operates on a block consisting of at least two 32-bit words using a 128-bit key. A single round of the algorithm can be viewed as operations on a word and its two adjacent words (previous and the next one). Figure 1 shows one round of XXTEA cipher, where x_r represents a current block and the four-squares symbol represents addition Modulo the size of the word. The number of rounds equivalents to the number of words in the block.

(Rinne et al. 2007) analysed the performance of several ciphers, including DES (Federal Information Processing Standards 1993), AES (Daemen & Rijmen 1999), IDEA, SEA (Standaert et al. 2006), HIGHT and the TEA family. The TEA family requirements in terms of the code space required, are among the lowest (after the IDEA algorithm) throughout all ciphers. The small code space footprint was achieved thanks to the lack of substitution tables common in other block ciphers. The XXTEA optimisation and performance analysis were also provided in (Jinwala et al. 2008) proving it to be a viable encryption algorithm for WSNs. Recently, a chosen-plaintext attack against the XXTEA requiring 2^{59} queries was discovered (Yarrkov, 2010). Here, advantage was taken of the fact that the number of full cycles to perform over each block is equivalent to $6 + 52 / n$, where n represents the number of rounds. If the block consists of at least 53 words then the number of cycles per word is reduced to only 6. This characteristic was used to perform differential cryptanalysis, where the difference was considered subtraction per word. The author described two attacks proving that XXTEA does not provide the intended 128-bit security.

Figure 1. One round of XXTEA (el Ruptor, 2007)



2.3.2. Scalable Encryption Algorithm (SEA)

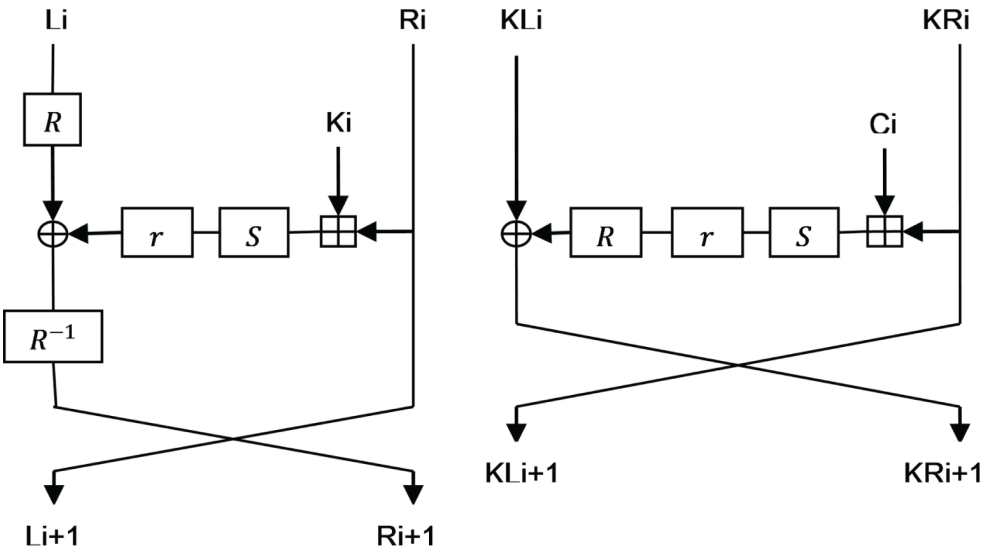
The Scalable Encryption Algorithm (SEA) (Standaert et al. 2006) provides low cost encryption implementable on resource limited processors. Similarly to the TEA family, it uses basic operations such as logical AND, OR, XOR, word/bit rotations, modular additions and a simple substitution box. Apart from the limited instruction set, the other design criteria were the low memory requirements and small code size. It is capable of "on-the-fly" key derivation. The scalability of the algorithm is achieved through the flexibility in the size of the input parameters. There is one constraint on the size of the key/plaintext that n is a multiple of $6b$ ($n = x6b$). The minimum required number of rounds to provide security against well known attacks (assuming word size equal or greater than 8 bits) is:

$$\frac{3n}{4} + 2\left(n_b + \left(\frac{b}{2}\right)\right)$$

Figure 2 shows one encryption and key round, where R denotes the word rotation, r the bit rotation and S the substitution box ($S_T := \{0,5,6,7,4,3,1,2\}$ in C-like notation). The $C(i)$ represents an n_b -word vector with all words of a value 0 except the least significant word which value is equal to i. The Li and Ri represent left and right halves of the word or the key (KLi, RKi).

The performance of the algorithm was analysed using the Atmel AVR ATiny reference 8-bit CPU platform among others. The expected code size for a 96-bit key implementation was estimated at 386 bytes and the amount of clock cycles required for encryption/decryption was estimated

Figure 2. Encrypt/decrypt and key round of SEA



at 17745. A performance analysis (Rinne et al., 2007) on AVR Atmel163 showed a code size of 2132 bytes for the 96-bit SEA (compared to 1160 bytes for XTEA) and the number of CPU cycles required to complete encryption/decryption was 9654 (compared to 6718 with XTEA). The performance and code space requirements of the XTEA algorithm look more promising than the SEA. However, due to the discovery of security weaknesses in XXTEA, the implementation of this algorithm will be abandoned in favour of the Scalable Encryption Algorithm in 96-bit version (Perrig et al. 2002).

3. RESOURCE-LIMITED DEVICES

The term Resource-Limited Device (RLD) describes a microcontroller device with significantly lower processing power and limited memory in comparison to a modern Personal Computer. This group of devices range from Radio Frequency Identification (RFID) transponders to a wide spectrum of embedded devices equipped with small (typically 8-bit) microcontrollers. Such devices are utilised in wireless sensor networking for example. This research focuses on the security of the communication over the radio channel, thus the area of research will be restricted to Wireless Sensor Networks and advanced RFID systems. The work here is predicated upon the application of the Nordic Semiconductors nRF9E5 Integrated Circuit (Nordic Semiconductors, 2009b) as the target device. This microcontroller was chosen due to its low price (approximately 2$US per unit at quantities over 1000), integrated UHF radio transceiver and excellent power saving characteristics which make it an ideal solution for the design of a low-cost wireless sensor. The nRF9E5 entire chip will be referred to as a microcontroller and the Intel 8051-compatible Central Processor Unit - a subset of this system will be referred to as CPU or microprocessor.

3.1. IWSN

A typical Wireless Sensor Network consists of a set of Wireless Sensor Nodes and one or more Upload Stations (also referred to as Gateway Sensor Nodes or sinks) (Akyildiz et al. 2002) that provide a connection to a Host Computer on an external network. The external network uses a communication media not available to the Wireless Sensor Nodes, such as Ethernet or a different RF technology. The most commonly used architecture (Ye et al. 2002) is an ad-hoc network where every sensor node either broadcasts the message to all other nodes (using an endless message repetition preventing mechanism) or uses a routing mechanism to forward the message to the upload station through a series of other sensor nodes used as 'hops'(Kamble et al. 2007). Once the upload station receives the message it is uploaded to the External Network. Such architecture is useful in applications where sensors are distributed in an unplanned manner (e.g. battlefield sensor network deployed from an aircraft) and messages can be sent unreliably with no confirmation of the delivery from the Upload Station (although the acknowledgement system can be implemented in this architecture if the routing mechanism allows that). The Infrastructure Wireless Sensor Network (IWSN) example (HINT Project 2010) describes an architecture consisting of the following:

- **Master device:** An equivalent of the Upload Station in Classic WSN, linked to the External Network using for example an Ethernet controller or 802.11 WiFi controller;
- **Sensor (Slave):** A battery operated Wireless Sensor Node in the network equipped with microcontroller, radio transceiver and ADC converter allowing readings from the attached sensors;
- **Repeater:** A bridge forwarding messages between wireless sensors and a master device. Uses similar radio hardware to sensors but is assumed to have a regulated power source;
- **Host PC:** Host computer used by an operator to control the IWSN.

This type of architecture can be found in WSNs with a planned distribution of sensors, e.g. a network of temperature monitoring sensors deployed within a large building. It is assumed that all devices operate on the same radio frequency. All battery operated sensors (Slaves) attempt to connect to the Master device at a pre-programmed interval. The Master device uses an acknowledgement mechanism to guarantee the delivery of a single packet or an entire multi-packet transmission (depending on the configuration and packet type). Each Slave repeats the transmission attempt a pre-programmed number of times if an acknowledgement was not received. Repeaters are used to extend the coverage area of the network by forwarding each received packet to the Master (or other Repeater) and forwarding acknowledgements back to Sensors.

All devices using the radio link utilise a simple collision avoidance mechanism with a back-off system similar to Pure-Aloha Protocol (Abramson, 1970). Here every node listens to determine if the radio carrier is busy prior to a transmission attempt. If the carrier is sensed as busy then a node backs off for a random period of time before another transmission attempt. The main advantage of this architecture is the simplicity of communication between devices as no routing tables need to be maintained, even though the delivery of specific (user pre-defined) data packets can be confirmed by the acknowledgement mechanism. Thanks to this simplicity the radio-handling part of the software can be implemented within the limited code space and run efficiently on many Resource Limited Devices. However, this architecture is not ideal in environments, where Repeaters and Masters cannot be provided with a fixed power source. The other disadvantage is that as more slaves are introduced to the network performance degrades. The simplicity of the collision avoidance mechanism inhibits the use of a large number of slaves because slaves share a common frequency channel for transmission.

3.2. Authentication and Encryption in IWSNs

Infrastructure WSNs experience common issues related to the use of modulated radio frequency spectrum (radio waves) as the communication medium: Eavesdropping is possible on any wireless link using virtually any radio transceiver tuned to a given frequency with the ability to demodulate the signal. The architecture assumes that the link between the Master and the Host is secure. The Repeaters are used only to receive and forward data packets, without processing them and act as radio range extenders. Repeaters will not perform any active role in the security mechanism. The only parties requiring mutual authentication and secured (encrypted) communication channel are the Master and the Slave. Since the encryption and decryption mechanism has to be implemented on the Slave device, choosing such a mechanism must involve consideration of the limitationss. Ideally, the Master device will have an always-on, secured link with a Host (server) and this Host device can perform all of the computationally heavy encryption and decryption-related calculations. In other words, the master can offload all computationally heavy tasks to a back-end server and accept returned values. This permits the processing power of the master device to be used to handle service requests from a number of slave devices, rather than becoming occupied with computations associated with authentication and encryption.

3.3. Resource Limited Devices Security Issues

The constraints imposed on possible implementations of security systems for RLDs can be categorised as Central Processor Unit (CPU) limitations, memory limitations, power consumption and cost barriers. The main CPU constraint in resource-limited devices is obviously the limited processing power of the processor. Passive RFID transponders (powered by an external interrogator) with a very limited number of logic gates on the circuit may be only capable of performing simple logical operations with one-bit values. More powerful embedded devices may be using 8-bit CPUs for example the Intel 8051 derivative nRF9E5 clocked at 12MHz, which is able to execute only 750,000 operations per second (assuming that 50% of operations require two CPU cycles and the remaining require one). The number of operations per second is not the only constraint relating to the processor. Another issue related to microcontrollers is the word size. The most commonly used cryptographic standards were designed to be implemented either in hardware (e.g. the first proposals of the Data Encryption Standard - DES (Federal Information Processing Standards 1993)) or more flexible using software. However, the majority of standards assume that 32-bit CPUs will be used, thus their mathematical basis and implementation is commonly optimized to use 32-bit (e.g. the Rijndael cipher (Daemen & Rijmen 1999)) or even 64-bit word. 8-bit microcontrollers would be forced to perform numerous instructions to handle 32-bit numbers manipulation, e.g. it takes approximately 35 CPU operations to multiply two 32-bit numbers on the 8051 8-bit CPU (Vault Information Services 2009).

Low-cost passive Electronic Product Code (EPC) RFID tags can have as little as 104 bits of non-volatile memory (EPCGlobal 2008) and may not even contain any Random Access Memory. More advanced tags however, may be equipped with 1-2KB of memory. Microcontrollers are typically equipped with no more than 64KB memory, but this amount can be subject to limitations also due to 8-bit addressing issues causing slow access to some parts of the memory. Heavyweight cryptographic techniques using large keys (even 2048-bit in some RSA implementations) cannot be implemented in resource-limited device environments not only due to the amount of memory needed but also due to slow memory access times and limited read/write lifecycle.

Resource-limited devices are heavily constrained in terms of power availability for their operation. Passive RFID tags draw the whole power from the interrogating device; active RFID

solutions and wireless sensors are often powered by small cell batteries and are expected to provide a reasonably long operation time between battery replacements. Wireless Sensors are typically designed for one-time use, thus their lifetime can be increased only by power saving. All CPU-intensive operations and memory manipulations required by most cryptographic algorithms along with the radio transceiver usage are the most power consuming activities performed by such devices so they have to limited to a minimum. Most of the resource-limited devices are designed to be manufactured in high volumes with a very low price per item. An addition of a single logic gate to the electronic circuit may seem inexpensive but if multiplied by millions of manufactured items may have a substantial influence on the profit made by the manufacturer. This constraint forces solutions requiring little or no additional hardware modifications.

3.4. Hardware Platform: nRF9E5

The nRF9E5 microcontroller is a single chip system with an integrated sub-1GHz Radio-Frequency (RF) transceiver, 8-bit 8051-compatible processor and 4-input 10-bit Analogue to Digital (AD) converter. The design of the chip was based on the Dallas DS80C320 CPU in terms of hardware specification and instruction cycle timing. It is a low cost solution with extended power saving capabilities. The minimum power consumption in power down mode (where the chip can be woken up by a timer or an external pin) is only 2.5μA. The microprocessor draws 2.2 mA of current at a clock frequency of 16MHz and the radio transceiver (nRF905) uses 10 to 30 mA in Transmit Mode (depending on output power setting). Receive Mode power usage is estimated at 12.5 mA on average. The CPU is an 8-bit Intel 8051 derivative with the addition of Special Function Registers (SFRs) used to control the nRF9E5 radio transceiver. The microcontroller is equipped with 512 bytes of ROM that contains the bootstrap loader, 256 bytes of Internal Data Memory, 128 SFRs and 4 kilobytes of external on-chip RAM. The memory is organized with the Harvard Architecture in contrast to the Von Neuman architecture commonly used in desktop PCs. The bootstrap loader loads the program from the bottom area of external EEPROM memory upon each power-up or reset cycle. The manufacturer did not provide any possibility to extend the size of the on-chip RAM, thus the binary program size is limited to 4 kilobytes.

The on-chip radio transceiver subsystem is the Nordic Semiconductors nRF905 connected to the microcontroller through the SPI (Serial Peripheral Interface) port. It utilizes the nRF ShockBurst technology allowing high speed radio signal processing without the assistance of the microprocessor, which further reduces the power consumption requirements. The transceiver is able to generate the preamble and calculate CRC for each data payload when transmitting signals. Additionally, it can validate CRC for each incoming data payload. The CRC calculations are performed by an on-board circuit without the CPU's assistance. NRF905 supports standby mode, where the current consumption is limited but the short startup times are still maintained, and 4 different Radio Frequency (RF) transmitting power modes ranging from -10dBm (at 9mA of current consumption) to 10dBm (at 30mA of current consumption). The current consumption in receiving mode is estimated at 12.5mA and can be reduced to 10.5mA when using reduced receiving power mode. The modulation used for the air interface is Gaussian Frequency Shift Keying (GFSK) with Manchester Encoding yielding an effective data transfer rate of 50kbps. The transceiver is able to operate on radio frequencies 430 to 434.7MHz or 868 to 928MHz.

4. NRF9E5 PROCESSOR AND SECURITY

The nRF9E5 single chip system uses an 8-bit microprocessor with an instruction set compatible with the industry standard 8051 processor. The instruction timing differs from the industry

standard: each instruction uses 4 to 20 clock cycles instead of 12 to 48 in the standard. The hardware specification of the chip (Nordic Semiconductors 2009b) allows utilizing a 4-20MHz crystal oscillator to generate clocking signal on the circuit (shared by microcontroller, AD converter and radio transceiver). The crystal oscillator can be started and stopped as requested by software. While it is stopped, nRF9E5 uses the internal low power 4KHz RC oscillator which runs continuously (as long as 1.8V of power is supplied) and ensures that vital functions such as the wake-up timer are functioning even in deep power saving modes.

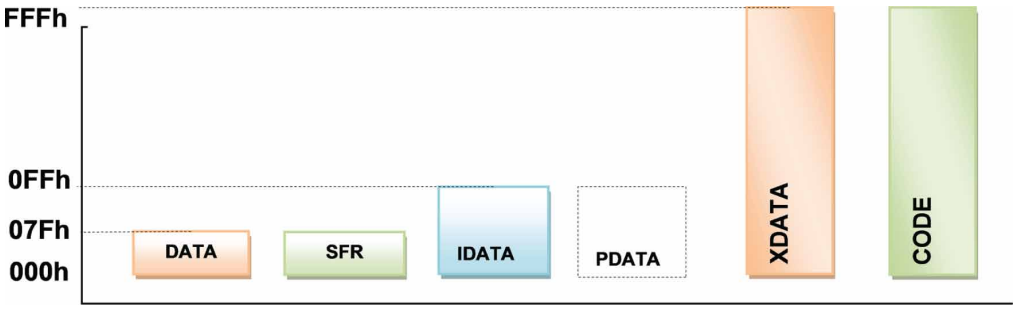
The microcontroller's architecture is 8-bit: each machine language opcode (operation code) is a single 8-bit value, which allows for 256 different instruction codes. Most of 8051's registers are 8-bit values, e.g. the Accumulator, each of the Register Banks. There are several special cases where a given register is referred to as 16-bit (such as the three Timers), but in fact these registers are addressed as two separate 8-bit registers often referred to as High and Low indicating which part of the 16-bit value they hold. The only truly 16-bit values that the 8051 handles are the Program Counter (PC) indicating the address of the next instruction to be executed and Data Pointer (DPTR) used for memory addressing. The CPU is only capable of performing basic mathematical operations on two 8-bit numbers at each cycle. There is no additional hardware support for calculation of numbers larger than 8-bit or any decryption/encryption coprocessors. In consequence manipulation of larger numbers requires numerous 8-bit calculations, for example a multiplication of two 16-bit numbers requires 9 CPU instructions. The 8-bit word size, relatively low CPU clock frequency and the lack of mathematical hardware coprocessors in the nRF9E5 narrow the area of possible security protocols and algorithms which can be successfully implemented to those that do not require exhaustive calculations (required by most of the Asymmetric Cryptography techniques) and those that are optimized for 8-bit values. In consequence, only lightweight authentication protocols and lightweight encryption algorithms are reviewed and analysed in this dissertation.

4.1. nRF9E5 Memory Structure and Security

The nRF9E5 microcontroller has 256 bytes of Internal Data Memory used as a RAM with fast access, 128 Special Function Registers (one byte each) used to set different operating modes of the CPU and the Radio Frequency Transceiver. Additionally it contains 4 kilobytes of external on-chip RAM. The memory uses Harvard Architecture and is organized into six different memory spaces (see Figure 3). It provides 128-bytes of directly addressable DATA RAM (8052 compatible) but may also be used to hold IDATA-addressed variables. The next 128 bytes are the IDATA memory area which is accessible through indirect addressing and effectively interleaves with the Special Function Register (SFR) which in turn is directly accessed. The entire 4K of memory (addresses above 0FFh) is accessible as an external XDATA memory but this area is shared with the CODE memory, so the use of XDATA variables effectively limits the available code space. The first of 256 bytes of XDATA can be addressed in paged mode and in this configuration it is referred to as PDATA. Memory addressing diagram can be seen in Figure 3.

Additionally, there is a small 512 byte ROM area located on-chip and containing bootstrap program executed automatically after power on or reset. The bootstrap loads the user program into on-chip 4K RAM from the off-chip external EEPROM memory required for operation. The manufacturer of the chip did not include any options to extend the RAM size above the 4KB - it is not possible to connect any additional external memory directly to the CPU pins. Additionally, the bootstrap program in ROM cannot be updated. The only memory size expanding option is to use an external EEPROM memory (generic 25320 with SPI) attached through one of the GPIO pins and interfacing through a common SPI bus. Accessing external memory through the SPI bus

Figure 3. 8051 memory addressing



has major consequence on the performance of the CPU as each of the SPI read/write (performed byte-by-byte) operations takes several processor cycles. The CPU is not able to perform additional tasks while in this process. One of the major limitations is the fact that external EEPROM cannot be used to expand the possible program size. In consequence, the program code size is always limited to 4KB – the bootstrap program will ignore anything above 0FFFh address in EEPROM when loading the program. We provide a solution to overcome this limitation with the support of 8051 dedicated software Assembly Language Linker. Possible security implementations have to be filtered through the following constraints imposed by nRF9E5:

- **Limited code/RAM space:** The CODE and XDATA space are shared in this CPU's architecture, so variables and constants allocated here limit the overall code space. The existing Infrastructure WSN programs already occupy a vast amount of the code/RAM space (HINT Project 2010), so the algorithms/protocols have to be implementable with a minimum machine code size and there must be a limited need for variable memory allocation. In case the solutions used to overcome the memory limitations fail, the space for the machine code may be limited to approximately 200 bytes only (assuming that for an existing sensor program already occupies 95% of the available code space);
- **Extremely slow access to the external non-volatile memory:** The reference platform utilizes 32Kb 25320 generic EEPROM. The amount of the data which needs to be accessed from the external EEPROM memory and the frequency of the access has to be limited. This imposes restrictions on possible encryption key sizes and the usage of non-volatile protocol-specific data;
- **Hacking the EEPROM:** The EEPROM memory can be read by freely available EEPROM programmers, thus in the case of a physical access attack the amount of information disclosed cannot compromise the security of the entire system. This forces solutions without global pre-shared encryption keys. An example of a physical access attack compromising the security of the entire WSN using the TinySec Protocol (Karlof et al. 2004) was described in (Hartung et al. 2005).

4.2. Radio Transceiver and Security

The nRF9E5 single chip microcontroller integrates a nRF905 (Nordic Semiconductors 2009a) compatible Radio Frequency (RF) transceiver operating on 433/868/915MHz bands (sub-1GHz). The transceiver consists of a fully integrated frequency synthesizer, a power amplifier, a modulator and receiver chain with demodulator.

The modulation type used in nRF905 is Gaussian Frequency Shift Keying (GFSK) with a data rate of 100kbps. The data bits are encoded and decoded using Manchester Encoding/Decoding and the effective symbol rate is limited to 50kbps (one symbol per two clock signals); however, no scrambling on the microcontroller is needed.

The transceiver uses SPI bus for reprogramming and data input/output. It is equipped with a circuit able to calculate the Cyclic Redundancy Check (CRC) checksum of the incoming or outgoing data packets. Transmitting (TX) and Receiving (RX) addresses can be 1 to 4-byte long and the data payload length may vary from 1 to 32 bytes.

Each data packet contains the following (see Figure 4):

- **Preamble:** Predefined 10-bit sequence used to adjust the receiver for optimal performance;
- **TX Address:** Programmable recipient's address with a length of 1 to 4 bytes;
- **Payload:** User data, length of the field configurable within 1 to 32 bytes range;
- **CRC:** 8 or 16-bit CRC checksum.

During the TX mode the packet is assembled automatically by the transceiver once the Payload and TX address is supplied – the CRC is calculated and added with a Preamble. After a transmission the RF transceiver sets the Data Ready (DR) pin high, so the microprocessor can be notified of a finished transmission. In RX mode the radio is used to listen for incoming transmissions and if one occurs the Carrier Detect (CD) pin is set high. After this action the nRF905 analyses the Address field and discards the packet if it is destined for a different address or accepts it if the address matches, sets the Address Match (AM) pin high, reads in the payload to the buffer and verifies the CRC checksum. The way the RF transceiver handles incoming packet addressing (automatic packet discarding when the address does not match) imposes constraints on the possibilities of protection against traceability (ID disclosure related) attacks (Juels 2006). In consequence in a situation where the communication is initiated by the Master device the packet will need to hold a broadcast address and all Slaves should be able to temporarily reconfigure themselves to accept such packets. A frequent usage of broadcast addressing may negatively impact the performance of the entire network (Ni et al. 1999). Another solution would require Slaves to ignore address mismatch and examine each packet which again reduces the performance of the network. The maximum Payload size of 32 bytes seems large but the bandwidth of only 50kbps has to be taken into consideration too. In the presence of multiple devices operating on the same frequency the transmission time has to be limited to avoid network congestion. It has to be noted also that the entire NRF9E5 consumes the highest possible amount of power during radio transceiver operations (up to 30mA at 10dBm output power comparing to 2.2mA when only the 8051 CPU is active), thus large data transfer, although possible, can severely degrade the sensor's lifetime. In consequence of the above limitations, the security system has to impose low radio bandwidth requirements.

Figure 4. NRF9E5 packet structure

4.3. Code Banking on the nRF9E5

The major limitations of the reference platform nRF9E5 chip are the code space size and the lack of any coprocessors enhancing mathematical calculations. While the latter can be overcome by using less CPU intensive security protocols and algorithms, the program size and RAM limitations are hard to overcome without changing the entire microcontroller platform. A software solution to this issue using the concept of Code Banking with the native support of the 8051 Assembler Linker (similar solutions are available from KEIL (ARM Ltd. 2009a) and Raisonance (Raisonance SAS 2010) Integrated Development Environments) is proposed below.

The origin of the Code Banking concept (ARM Ltd. 2009b) comes from the 16-bit memory addressing limitation of the 8051 CPU. Due to the addressing bit width the maximum memory which can be allocated is limited to 64Kb. The Code Banking mechanism permits and increases in the Code memory size up to 1MB (KEIL linker) or 4MB (Raisonance linker) by splitting the program into a Common Area section and a number of memory banks (see Figure 5). The Common Area (of a user-defined size s) and one of the Code Banks is loaded at a given time, so the microcontroller can effectively "see" and address 64KB of the Code memory. If a function makes a call to another function the linker generates a code performing that switch, called a Bridge. All bridges are located in the Common Area which remains the same regardless of which Code Bank is currently used. The full description of the assembly language routines performing bank switches and limitations such as interrupt vector handling are outside the scope of the document and can be found in Raisonance and KEIL linkers' documentation (ARM Ltd. 2009b).

A typical hardware design scenario permits connecting the memory directly to the CPUs I/O ports. In this case a bank switch process would only require changing the input/output port number to access different blocks of memory, where additional code banks are located. The Common Area has to be duplicated across all memory blocks so it would still be accessible in the same form after changing the I/O ports.

In case of the reference platform with the nRF9E5 microcontroller, where it is not possible to connect any additional memory directly to the CPU, the Code Banking mechanism can be

Figure 5. Code banking layout

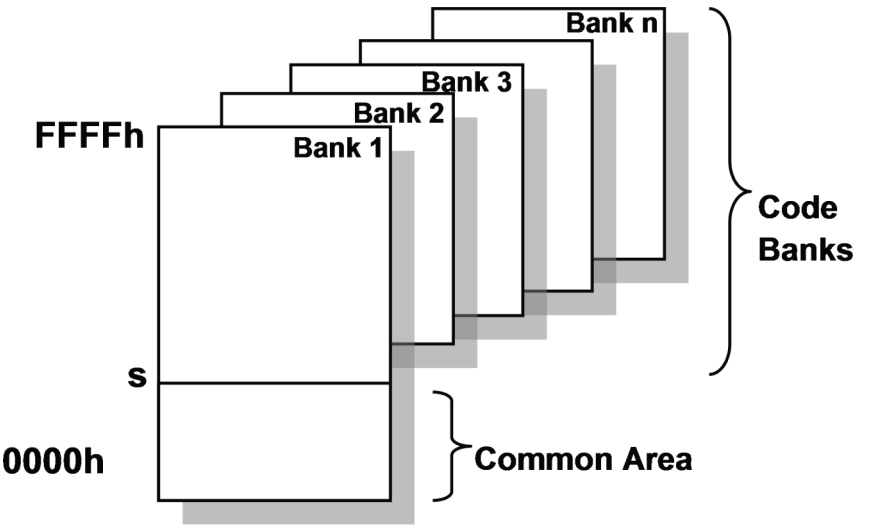
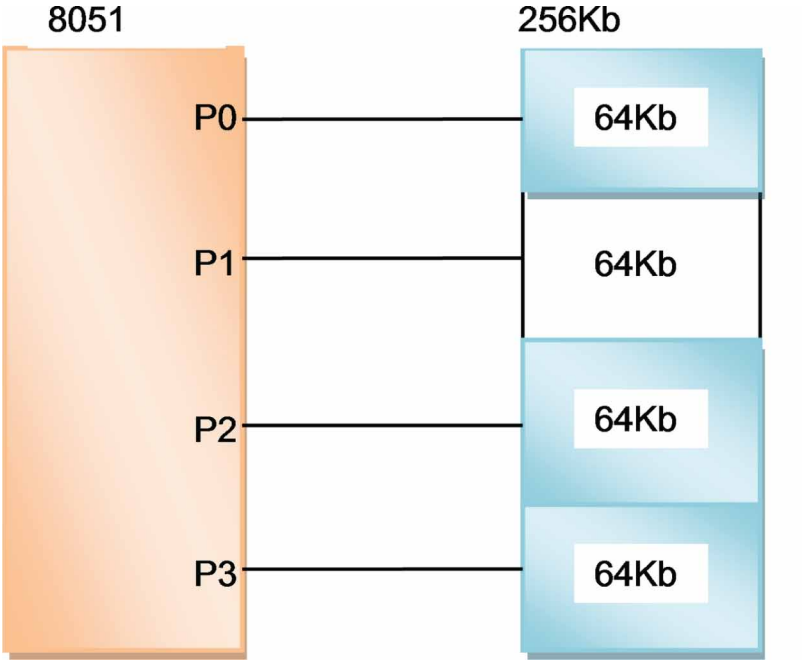


Figure 6. 8051 with 156Kb EEPROM attached to ports P0-P3

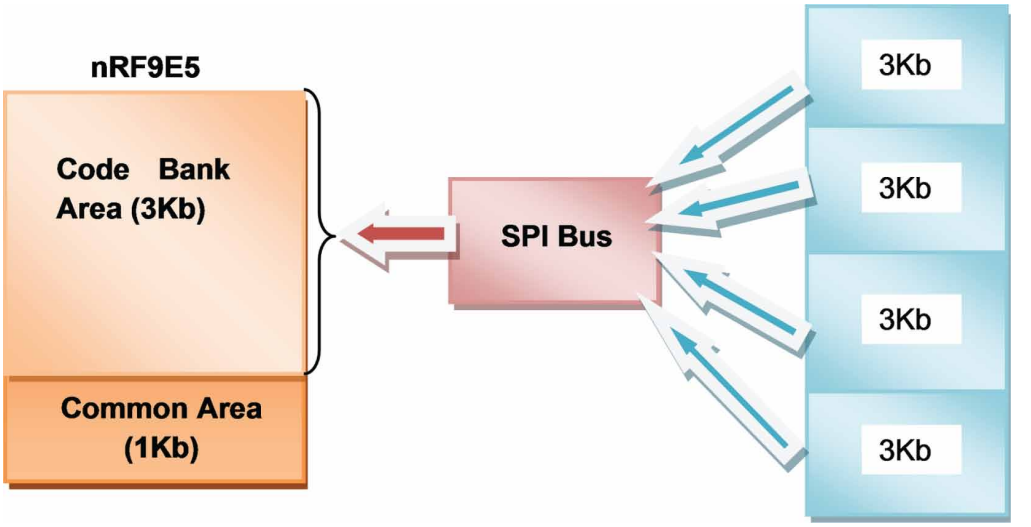


utilized to overcome the Code space limitation but in a manner different to the original design. Instead of using the directly attached memory chip an external EEPROM connected to the SPI bus can be utilized to hold additional code banks (see Figure 6). However, the I/O pin switching routine has to be replaced with a function that overwrites the code bank space in the on-chip RAM with the content of this bank located on the external EEPROM. Every bank switch will be a very slow process since the entire code bank binary file (2-3Kb) has to be read through the SPI bus from the external EEPROM (see Figure 7). Experiments have shown that it takes 65 milliseconds to load a Code Bank of 2Kb in size (HINT Project, 2010). Despite the negative effect on the microcontroller's performance this mechanism permits the effective expansion of the available code space above 4Kb without any hardware modifications in the existing reference platform. This would allow providing a relatively large code space for the implementation of the security mechanism. The failure of this concept would result in significant code space limitations for the security algorithms and force the usage of slow –access EEPROM-located variables.

5. LIGHTWEIGHT AUTHENTICATION PROTOTYPE

The main development platform used is the Nordic Semiconductors nRF9E5 microcontroller that is used for both master and slave devices. The nRF9E5 has limited resources and this has implications for the implementation of authentication and encryption on these devices. This limitation is somewhat eased by (a) offloading computationally heavy tasks from the master to a back-end server, allowing the master to more effectively handle service requests from slave devices and (b) by improving code memory space using code banking. Neither of these enhancements has been used in this project. Since a vastly scaled down communication and radio protocol is used,

Figure 7. nRF9E5 code banking with an external SPI-accessed EEPROM



the inherent memory of the nRF9E5 is sufficient to effectively run the security mechanisms. Variables that would otherwise be serviced from a back-end server have been hard-coded into master and slave, negating the use of the back-end server in the developed prototype. However, in a field implementation of secure sensor networking, where many slaves communicate with a master, it would be necessary to use a back-end server and overcome the code space limitations through code banking. Considering the limitations of the devices, a C language implementation was chosen instead of 8051 native Assembly code to allow faster porting to other platforms.

The main limitation of the nRF9E5 microcontroller in terms of the implementation was the maximum code size of only 4 kilobytes. The prototype was implemented to fit under this barrier. However, some protocol simplifications were needed to achieve small code space. The amount of RAM (256 bytes for both Data and Idata) was sufficient but almost entirely used by both master and slave prototypes. The radio transceiver embedded on nRF9E5 requires pre-configuration and manual handling of the OSI Model Data Link and upper layers. This generates another code space requirement, thus a simplified radio protocol is used in the prototype. The hardware design of the radio transceiver offers two useful tools that simplify the radio protocol implementation: Address Match and Carrier Detect bits. These tools were used to implement a simple Listen-Before-Talk collision avoidance scheme.

There are two well known Integrated Development Environments offering packaged Assembler and ANSI-C compilers for the 8051-compatible microcontrollers: KEIL (ARM Ltd., 2009a) and Raisonance RC51 IDEs (Raisonance SAS, 2010). Raisonance RKit Eval51 was utilised as it offers an 8051 compiler fully functional with the exception of a code size limited to 4 kilobytes. The code size limitation perfectly matches the hardware limitation of the nRF9E5 microcontroller. The hardware used in the implementation stage were two Nordic Semiconductor Evaluation Boards nRF9E5-EVBOARD with EEPROM emulator/programmer USB dongles nRF24E1. The programming dongles were controlled by the nRFPROG software supplied by Nordic Semiconductors.

5.1. Design: Algorithms for Both Authentication and Encryption

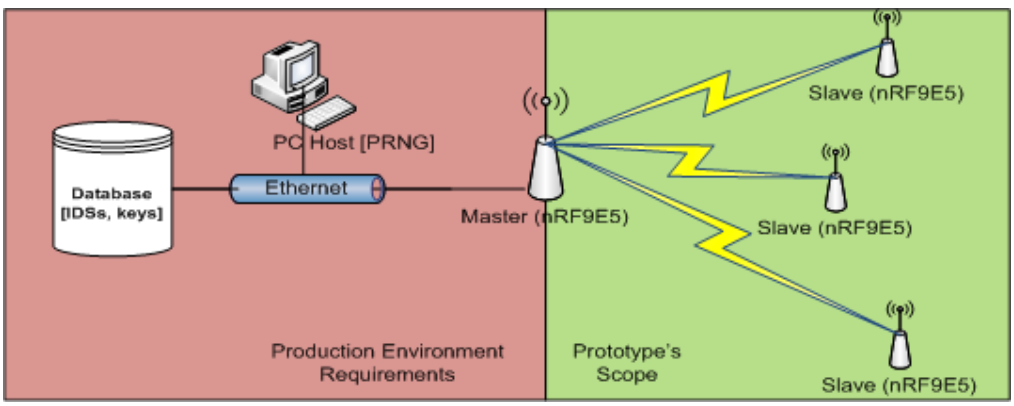
The prototype is designed to fit within 4 kilobytes of total code space available for programs on the nRF9E5 reference platform. The usage of code banking or other techniques overcoming the 4KB limitation is not considered in the prototype implementation. Instead some minor simplifications in the protocol (explained below) are used. The scope of the prototype is explained in Figure 8. The back-end database and PC Host software are outside the scope of the implementation – it will focus only on the 8-bit microcontroller code written in the C language with nRF9E5-specific radio transceiver handling functions.

The prototype uses a simplified radio protocol allowing communication between the Master and the Slave. Both devices utilise Address Match (AM) and Carrier Detect (CD) bits. The Carrier Detect bit will be used to implement a simple Listen-Before-Talk collision avoidance mechanism. Both devices test the CD bit before switching the radio into transmitting (TX) mode. In this simplified model a transmitting device will loop forever waiting for the CD bit to be clear and attempt the transmission straight away after this bit is cleared. Due to code space constraints random TX back-off period (Pure-Aloha Protocol) or wait-until-CD timeout is not implemented in the prototype.

The Gossamer lightweight authentication protocol was chosen to fulfil the requirement for mutual authentication between the Master and the Slave devices in Infrastructure WSN. It has proven security, low memory, computation requirements, and the expected simplicity of the implementation of all necessary mathematical operations on 8051-compatible CPU. The original design of the Protocol is simplified for the prototype implementation purposes in the following areas:

- **The keys and IDS:** Are not stored persistently on the Slave device due to code space overhead imposed by the EEPROM read/write routines. Upon each power loss these values will be reset to the initial ones;
- **Master side:** Random numbers n1 and n2 will be replaced by hard-coded values for experimentation purposes. The IDS of a sample Slave will also be hard-coded, so the back-end database will not be needed in the simplified model;
- **Slave side:** In the original Gossamer Protocol the Slave device sends the value D but there is no acknowledgement that D has been received and verified by the Master. The Slave

Figure 8. The scope of the implementation part



then updates its keys and IDS and saves the previous IDS and key values. In a subsequent round, if the slave cannot verify value C, in which case authentication of the master will not have been a success, the slave can roll back to the previous keys and IDS values. This de-synchronization attack prevention mechanism has not been implemented in the simplified protocol.

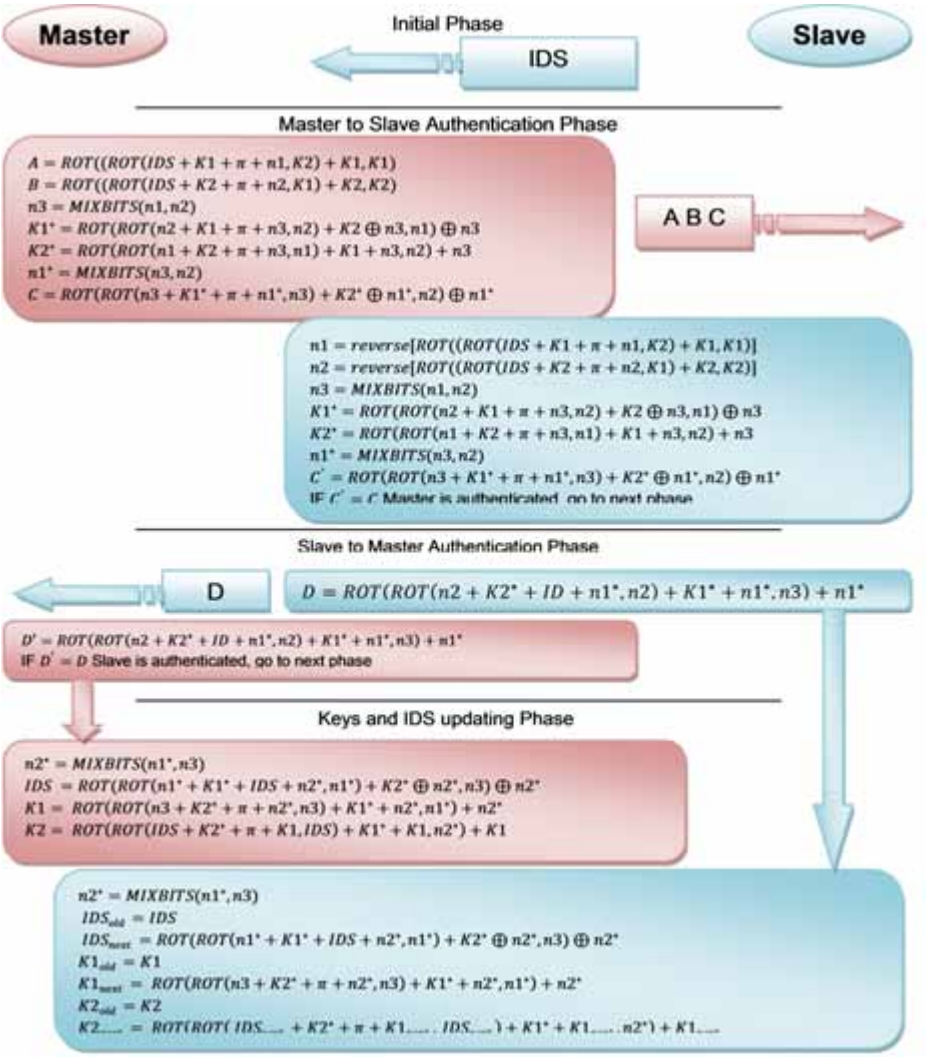
Figure 9 shows the full round of the Gossamer authentication protocol adapted to the needs of the Infrastructure WSN. The main difference was the removal of the 'Hello' message as in IWSN the Slave device (Tag equivalent in standard Gossamer specification) initiates the communication. The Scalable Encryption Algorithm (SEA) was chosen as the encryption mechanism. SEA (96, 8) mode was used, meaning that the block and the key size of 96-bits and 8-bit word matching the word size on the nRF9E5. The choice of the algorithm can be justified by the lack of proven weaknesses in the algorithm and the fact that the algorithm can be implemented with a very limited code space by sacrificing the throughput of the encryption (number of words that can be encrypted over a given period of time). The reduced throughput of the algorithm is not a significant issue in the context of IWSN, where the amount of data transferred is very small in most cases.

Another advantage of this algorithm is its scalability which permits increasing key and word sizes to 192 bits without major modifications of the code. This can be applied in cases where the 96-bit security is not regarded as strong enough. Since the Gossamer authentication protocol exchanges two new 96-bits keys at each round, one of these keys can be used as an encryption key for the SEA (96,8) algorithm during one communication session between the Master and the Slave. Both the Master and the Slave programs were written in two separate modules: Master.c and Slave.c. Each of the modules contains Initialization (UART timers, radio), Utilities Block (UART handling, SPI handling), Radio Handling Block (TX and RX), Gossamer functions and SEA functions (see Figure 10).

The Gossamer Protocol implementation uses the main gossamerMaster and gossamerSlave loops. All 96-bit values are implemented as an array of 12 unsigned characters (one byte each) in Big-endian (Most Significant Bit first) notation. The IDS, ID, K1, K2 and Pi values are initialized on the startup of the main loop, thus on every power-loss they are reset to the hard-coded values. All Gossamer-Related mathematical operations are implemented in separate functions explained next:

- Addition Modulo96 is executed on two arrays passed as parameters and saves the result into the second argument's memory location. The function simply adds each element in the array one-by-one starting with the last element (12) and carries a bit over to the lower element if the result is larger than 255. If the lower elements are 255 already then the bit is carried over to lower elements until the head of the array if needed;
- Subtraction Modulo96 is executed on two arrays passed as parameters and saves the result into the second argument's memory location. Similarly to the Addition function it simply subtracts each element one-by-one starting with the last element. If the minuend is smaller than the subtrahend, then a bit is borrowed from the lower element. If the lower elements are zeros then the borrow bit is taken from lower elements until the array head is met. This function is only required on the Slave side as it is only needed when extracting random numbers n1 and n2 form message A and B respectively.

Figure 9. Gossamer protocol adapted to the infrastructure WSN

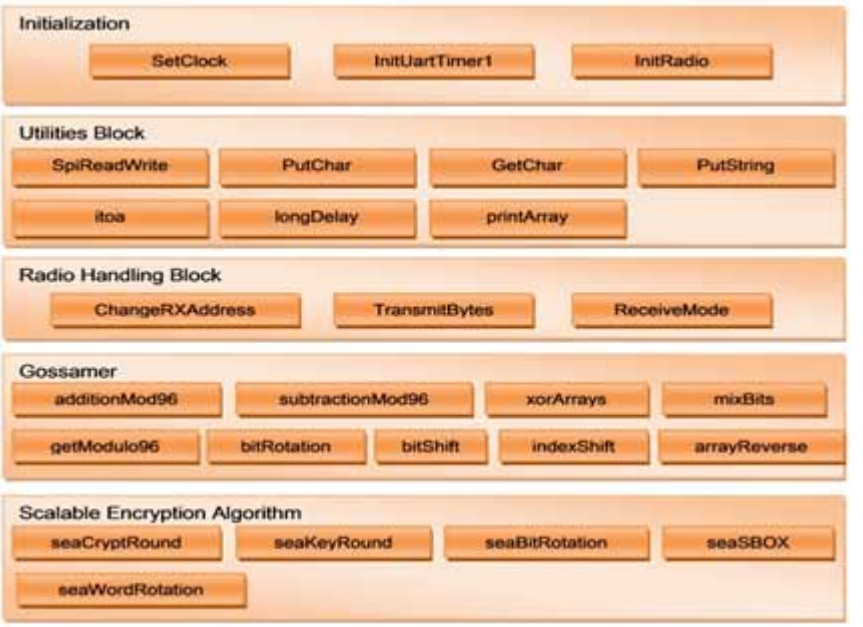


The XOR two 96-bit numbers function loops through all elements in the array and performs a bitwise exclusive OR operation on them one-by-one.

The bitRotation function performs circular bit rotation of a 96-bit number by a Modulo96 of a second number passed as a second parameter. This function uses four sub-functions to perform the bit rotation:

- **getModulo96:** Returns Modulo96 of a 96-bit number passed in the array of 12 one-byte elements. The function uses a command and conquer approach. Starting from the lowest element a Modulo96 of each element (split into two 4-bit numbers and multiplied by 256) is calculated one-by-one and added to the overall result. At each iteration the overall result is reduced Modulo96;

Figure 10. Main program components



- **bitShift:** Performs circular bit-shift (up to 7 places) of each element in the array in both directions. Depending on the direction the remainder of the shift is appended to the lower or upper element;
- **indexShift:** Rotates the elements of the array by up to 11 positions left or right. It takes advantage of the arrayReverse function and a formula assuming that the array is split into two sub-arrays A and B (A.B), where the size of array A is the number of places the elements are to be rotated. The formula is as follows:

B.A = reverse(reverse(A).reverse(B))

- **arrayReverse:** Reverses the elements in the array (array[beginning] becomes array[end] and so on).

The bitRotation function calculates the Modulo96 of the first argument (array to be rotated by) and then analyses the result to verify if bitShift and indexShift functions need to be called and calls them accrodingly. The MixBits function implements the Gossamer author's recommendation shown in Figure 11.

The function uses two arrays passed as parameters and a temporary array returned with the result. Functions described above (additionMod96 and bitShift) are utilized. The SEA (96, 8) implementation uses a word size of 8-bits (unsigned char) with a block and key size of 96-bits. Both are passed as an argument in a form of a 12-element array of unsigned characters. The main components of the SEA implementation are the following functions: cryptographic round, the key round, the S-Box, the bit-rotation, the word-rotation and the main SEA wrap-up function. In this prototype the key used in encryption will be either k1 or k2 updated by the Gossamer

Figure 11. MixBits function pseudocode

```
Z = MixBits (X, Y)
Z = X
FOR counter = 0 to 32
Z = (Z>>1) + Z + Z + Y
ENDFOR
```

function at each authentication round. Per SEA author's suggestions the S-Box can be applied bitwise to any 3 elements of a block-half currently being processed (for blocks of 96-bits). Since there are 6 one-byte elements in each half of the block the S-Box can be applied on two different set of words. (Standaert et al. 2006) suggested a function ('i' equals 0 or 1) shown in Figure 12.

Standaert agrees that it is safe to simplify this function so the S-Box is only to the first three elements in order to reduce the code space required by this function (see Figure 13).

The Bit Rotation function in SEA(96,8) implementation performs circular bit-rotation one place to the right on words numbered 0 and 3, and one place to the left on words numbered 2 and 5. The seaBitRotation function implemented uses Raisonance RC51 compiler's intrinsic functions '_cror_' and '_crol_' to save the code space (see Figure 14).

The seaWordRotation function performs circular right- or left-rotation of the block-half array elements by one place. The Gossamer indexShift function can be re-used to save the code space but this function was also implemented to make the SEA module independent and re-usable without the Gossamer functions overhead. The seaCryptRound function performs one round encryption or decryption round using left and right half of the block and one half of the key - left or right depending on the round. This function implements the following SEA equations: F_E encryption function and F_D decryption function (below):

Fe(Li, Ri, KeyHalf) = RightWordRot(Li) XOR bitRotation(sbox(Ri+ KeyHalf))
Fd(Li, Ri, KeyHalf) = LeftWordRot(Li XOR bitRotation(sbox(Ri+ KeyHalf)))

Figure 12. Code: SEA S-Box

```
void seaSBOX (unsigned char data *block, unsigned char i)
{
block[3*i] = (block[3*i+2] && block[3*i+1]) ^ block[3*i];
block[3*i+1] = (block[3*i+2] && block[3*i]) ^ block[3*i+1];
block[3*i+2] = (block[3*i] || block[3*i+1]) ^ block[3*i+2];
}
```

Figure 13. Code: SEA S-box modified

```
void seaSBOX (unsigned char idata *block, unsigned char i)
{
block[0] = (block[2] && block[1]) ^ block[0];
block[1] = (block[2] && block[0]) ^ block[1];
block[2] = (block[0] || block[1]) ^ block[2];
}
```

Figure 14. Code: SEA bit-rotation

```
void seaBitRotation (unsigned char idata *block)
{
    block[0] = _cror_(block[0], 1);
    block[2] = _crol_(block[2], 1);
    block[3] = _cror_(block[3], 1);
    block[5] = _crol_(block[5], 1);
}
```

The function takes advantage of previously described word rotation, bit rotation and substitution box functions.

The seaKeyRound function performs one round of the key scheduling. These rounds are interleaved with encryption/decryption rounds. Each key round performs the following key scheduling function:

Fk(KLi-1,KRi-1,Ci) <=> KRi = KLi-1 XOR RightWordRot(bitRot(sbox((KRi-1)+Ci)))

The function takes advantage of previously described word rotation, bit rotation and substitution box functions. The main SEA (96, 8) function takes two 12-byte parameters: block and key. (Standaert et al. 2006) advised that the minimum safe number of encryption/decryption rounds can be calculated using the following formula:

$$\frac{3n}{4}+1+2\left(\frac{n}{2b}+\left(\frac{b}{2}\right)\right), where\ \ n=plaintext\ \ size\left(96\ \ bits\right); b=word\ \ size\left(8\ \ bits\right)$$

The odd result in case of SEA(96, 8) is 93. The main function runs interleaved encryption (or decryption) and key scheduling round 46 times. After the initial 46 rounds the key halves are swapped and another further 46 rounds are executed. After the 92nd round another one encryption/decryption round runs - the key is in its final state already. It has to be noted that this final state of the key is identical to its initial state, thus no additional memory locations are needed to store a temporary key at each round. After the last round the block halves need to be swapped and the execution of the algorithm stops.

The experimental implementation takes Gossamer K1 key as an encryption key for the SEA algorithm. After a successful authentication round the Master encrypts a message using K1 and sends it to the Slave. The Slave decrypts the message using K1 and outputs it to the UART.

6. EVALUATION

The system was tested using the same hardware and software as in the implementation stage. During the testing stage two nRF9E5-EVBOARD development boards with nRF24E1 EEPROM programmers were used. The EEPROM programmers were connected over the USB link and the UART input/outputs from the development boards were connected through serial cables to the RS-232 ports on the development PC running Microsoft Windows XP Operating System. Since the RC51 compiler used does not offer nRF9E5-compatible debugger, the debugging was performed on-device using manually written debug messages sent to the UART I/O.

6.1. One Round Step-by-Step Test

The goal here is to verify the proper functioning of all core functions used by the Gossamer Authentication Protocol and the SEA encryption/decryption algorithm. Both the Master and the Slave programs are pre-configured with a Gossamer Protocol test data and set to output the data at each of the modifications so that the result can be verified with a 'paper-test' (manual calculation). The integer-to-ascii (itoa) function will be employed to output the data to the UART in a human-readable form. The SEA algorithm was tested step-by-step due to a large number of rounds. Instead, a result of the entire encryption and decryption loop is displayed.

After the Gossamer round, the Master uses the modified key k1 to encrypt a message (temp array) and sends it to the Slave (see Figures 15 and 16). After successful transmission, the Slave uses modified k1 to decrypt the message and displays it. The test was split into several stages to allow better readability:

Stage 1: Messages A and B creation (see Figure 17):

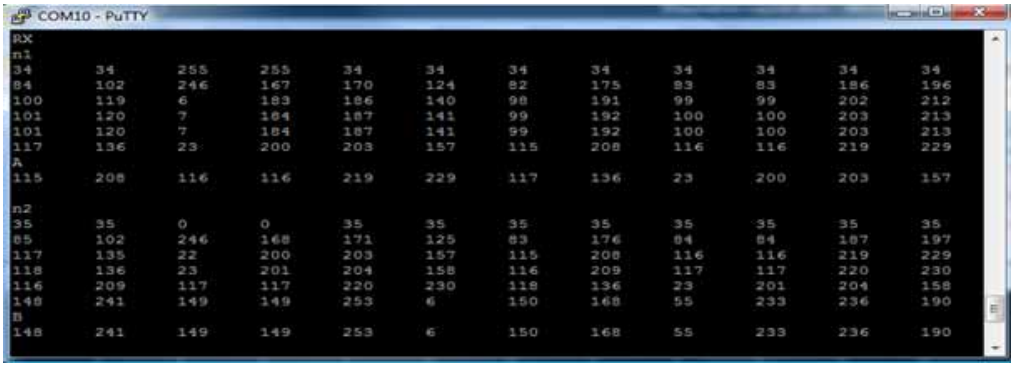
Figure 15. Code: Master side test data

```
unsigned char idata Pi[12] = { 0x32, 0x43, 0xF6, 0xA8, 0x88, 0x5A, 0x30, 0x8D, 0x31, 0x31, 0x98, 0xA2 };
unsigned char idata IDS[12] = { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 };
unsigned char idata ID[12] = { 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44 };
unsigned char idata k1[12] = { 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10 };
unsigned char idata k2[12] = { 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20 };
unsigned char idata n1[12] = { 0x22, 0x22, 0xFF, 0xFF, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22 };
unsigned char idata n2[12] = { 0x23, 0x23, 0x00, 0x00, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23 };
```

Figure 16. Code: Slave side test data

```
unsigned char idata Pi[12] = { 0x32, 0x43, 0xF6, 0xA8, 0x88, 0x5A, 0x30, 0x8D, 0x31, 0x31, 0x98, 0xA2 };
unsigned char idata IDS[12] = { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 };
unsigned char idata ID[12] = { 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44 };
unsigned char idata k1[12] = { 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10 };
unsigned char idata k2[12] = { 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20 };
```

Figure 17. Gossamer messages A and B creation (Master)



$$A = ROT((\left(ROT\left(IDS + K1 + \pi + n1, K2\right) + K1, K1\right)$$

$$B = ROT((\left(ROT\left(IDS + K2 + \pi + n2, K1\right) + K2, K2\right)$$

Stage 2: N1 and N2 extraction from messages A and B (Slave side, see Figure 18):

$$n1 = reversedA$$
$$n2 = reversedB$$

Stage 3: n3, k1next and k2next creation (Master side):

$$n3 = MIXBITS\left(n1, n2\right)$$

$$K1^{*} = ROT\left(ROT\left(n2 + K1 + \pi + n3, n2\right) + K2 \oplus n3, n1\right) \oplus n3$$

$$K2^{*} = ROT\left(ROT\left(n1 + K2 + \pi + n3, n1\right) + K1 + n3, n2\right) + n3$$

Stage 4: n3, k1next and k2next creation (Slave side):

$$n3 = MIXBITS\left(n1, n2\right)$$

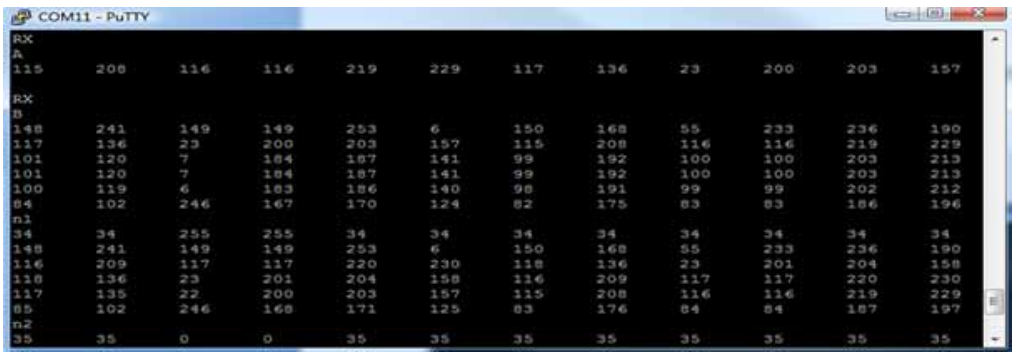
$$K1^{*} = ROT\left(ROT\left(n2 + K1 + \pi + n3, n2\right) + K2 \oplus n3, n1\right) \oplus n3$$

$$K2^{*} = ROT\left(ROT\left(n1 + K2 + \pi + n3, n1\right) + K1 + n3, n2\right) + n3$$

Stage 5: Message C creation (Master Side):

$$n1^{*} = MIXBITS\left(n3, n2\right)$$

Figure 18. Gossamer n1 and n2 random numbers extraction (Slave)



$$C = ROT\left(ROT\left(n3 + K1^* + \pi + n1^*, n3\right) + K2^* \oplus n1^*, n2\right) \oplus n1^*$$

Stage 6: Message C creation (Slave Side):

$$n1^* = MIXBITS\left(n3, n2\right)$$

$$C = ROT\left(ROT\left(n3 + K1^* + \pi + n1^*, n3\right) + K2^* \oplus n1^*, n2\right) \oplus n1^*$$

Stage 7: Message D creation (Master side):

$$D = ROT\left(ROT\left(n2 + K2^* + ID + n1^*, n2\right) + K1^* + n1^*, n3\right) + n1^*$$

Stage 8: Message D creation (Slave side):

$$D = ROT\left(ROT\left(n2 + K2^* + ID + n1^*, n2\right) + K1^* + n1^*, n3\right) + n1^*$$

Stage 9: IDS, k1 and k2 updating (Master side):

$$n2^* = MIXBITS\left(n1^*, n3\right)$$

$$IDS = ROT\left(ROT\left(n1^* + K1^* + IDS + n2^*, n1^*\right) + K2^* \oplus n2^*, n3\right) \oplus n2^*$$

$$K1 = ROT\left(ROT\left(n3 + K2^* + \pi + n2^*, n3\right) + K1^* + n2^*, n1^*\right) + n2^*$$

$$K2 = ROT\left(ROT\left(IDS + K2^* + \pi + K1, IDS\right) + K1^* + K1, n2^*\right) + K1$$

Stage 10: IDS, k1 and k2 updating (Slave side):

$$n2^* = MIXBITS\left(n1^*, n3\right)$$

$$IDS_{old} = IDS$$

$$IDS_{next} = ROT\left(ROT\left(n1^* + K1^* + IDS + n2^*, n1^*\right) + K2^* \oplus n2^*, n3\right) \oplus n2^*$$

$$K1_{old} = K1$$

$$K1_{next} = ROT\left(ROT\left(n3 + K2^* + \pi + n2^*, n3\right) + K1^* + n2^*, n1^*\right) + n2^*$$

$$K2_{old} = K2$$

$$K2_{next} = ROT\left(ROT\left(IDS_{next} + K2^{*} + \pi + K1_{next}, IDS_{next}\right) + K1^{*} + K1_{next}, n2^{*}\right) + K1_{next}$$

Stage 11: SEA Encryption using k1 (Master side);
Stage 12: SEA Decryption using k1 (Slave side) (see Figure 19).

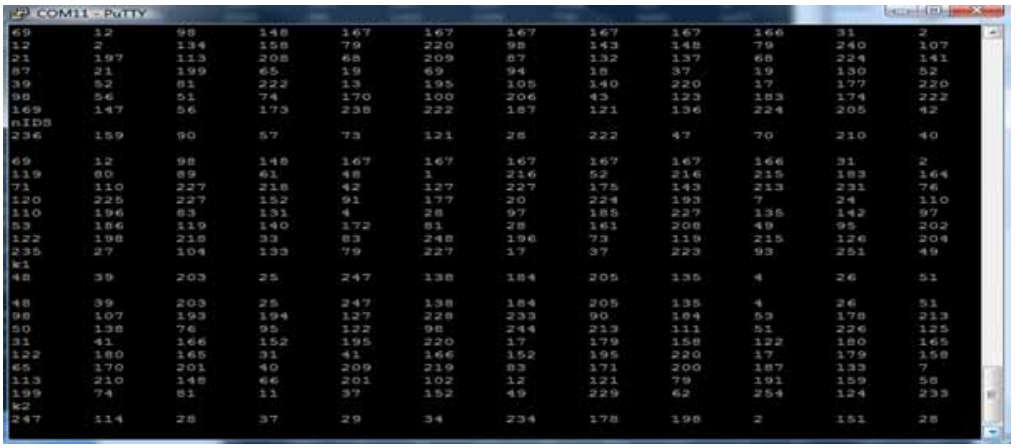
6.2. Long-Term Test

The goal here was to verify the proper functioning of the Gossamer Authentication Protocol and the SEA encryption/decryption algorithm using multiple values and multiple rounds. The test-Master and the test-Slave were pre-configured to loop indefinitely executing the following operations:

- **Both devices:** Mutual authentication between the test-Slave and the test-Master;
- **Both devices:** Updating values for the next round;
- **Master:** Encrypting a 12-byte message using the SEA encryption algorithm (using the Gossamer key k1) and transmitting the payload to the test-Slave;
- **Slave:** Receiving the payload form the test-Master and decrypting it using the SEA decryption algorithm and the Gossamer key k1.

The test-Master used a delay function before transmitting messages over the radio to allow for better readability of the UART output. Both the test-Slave and the test-Master outputed informational messages to the UART during each loop iteration. The time to complete an iteration of the main loop in both programs was estimated at approximately 1.5 seconds. The test-Master and the test-Slave programs were left running for 7 days. It was estimated that both programs would execute approximately 403200 authentication and encryption/decryption rounds (see Figure 20 and 21).

Figure 19. Gossamer keys and IDS updating phase (Master)



After the Gossamer round, the Master uses the modified key k1 to encrypt a message (temp array) and send to the Slave. After successful transmission the Slave will use modified k1 to decrypt the message and display it. Both the test-Master and the test-Slave were running continuously for 6 days and 23 hours and successfully executed approximately 400 000 mutual authentications and encryption/decryption rounds. The UART output was periodically monitored and no abnormalities were discovered.

7. RESULTS

The Master prototype required 3923 bytes + 80 bytes of xdata and the Slave prototype used in a Long-term test required 3937 bytes + 80 bytes of xdata code space. The main Gossamer Protocol function loop uses UART to intermittently output the results throughout the round. In order to verify the real size of the Gossamer loop without any UART overhead a special version of the program was compiled without any calls to the PutString or printArray functions.

The RAM storage requirements of the Master (229 bytes) and the Slave (244 bytes) seem high but it has to be noted that all arrays holding 96-bit numbers used by the Gossamer Protocol are initialized and stored in idata memory during the runtime of the program. In consequence 210 bytes of idata memory is used by the Master and the Slave. Approximately 80% of this space can be saved by moving the 96-bit values to the EEPROM trading off code space required by the external memory read/write functions. The total code space requirement by all the Gossamer-related functions is estimated at 1647bytes (66F Hex) on the Master side and 1710 bytes (6AE Hex) on the Slave side. The SEA functions code space requirements are identical on both the Master and the Slave programs and equal to 589 bytes (24D Hex). It has to be noted that the Raisonance RC51 compiler imports a LIB51 library which requires 552 bytes (228 Hex). This library is shared by many functions performing mathematical operations and is automatically imported even if only the SEA functions were to be implemented. In consequence, the code space requirements of the LIB51 have to be taken under consideration when estimating the total requirements.

The execution speed of different parts of the code was analyzed using an nRF9E5 timer interrupt set to 1 millisecond ticks and small timer handling functions. The timer was reset before

Figure 20. Code: Master initial values

```
unsigned char idata Pi[12] = { 0x32, 0x43, 0xF6, 0xA8, 0x88, 0x5A, 0x30, 0x8D, 0x31, 0x31, 0x98, 0xA2 };
unsigned char idata IDS[12] = { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 };
unsigned char idata ID[12] = { 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44 };
unsigned char idata k1[12] = { 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10 };
unsigned char idata k2[12] = { 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20 };
unsigned char idata n1[12] = { 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22, 0x22 };
unsigned char idata n2[12] = { 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23, 0x23 };
```

Figure 21. Code: Slave initial values

```
unsigned char idata Pi[12] = { 0x32, 0x43, 0xF6, 0xA8, 0x88, 0x5A, 0x30, 0x8D, 0x31, 0x31, 0x98, 0xA2 };
unsigned char idata IDS[12] = { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 };
unsigned char idata ID[12] = { 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44 };
unsigned char idata k1[12] = { 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10 };
unsigned char idata k2[12] = { 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20 };
```

entering a given block of code and the timer value was collected at the exit of the block. The full SEA (96, 8) encryption and decryption of a 12-byte block using 12-byte key and 93 rounds takes 27 milliseconds on an nRF9E5 microcontroller running at 16MHz. This gives an encryption/decryption throughput of 705 bytes per second. The full round of the Gossamer Protocol in the prototype program with no UART output (all PutString function calls removed) took 984 milliseconds on the Master side and 988 milliseconds on the Slave side. It has to be noted that the Master uses a longDelay function which loops for 280ms before each transmission (TX) attempt. At this time the Slave loops in the receiving mode (RX) waiting for messages. There are 3 TX attempts (messages A, B and C) so the total of 3*280ms can be subtracted from the total loop time on both the Master and the Slave side.

The full Gossamer loop time without the TX delay function for the master is 984ms - 3*280ms = 144ms and for the slave is 988ms - 3*280ms = 148ms. The Gossamer Protocol speed was also analyzed per major protocol stages. The message A and B creation were 2ms each (Master); message C creation was 65ms (Master and Slave); number n1 and n2 extraction: 2ms each (Slave); message D creation and verification: 3ms (Master); message D creation 2ms: (Slave) and keys and IDS update: 38ms (Master and Slave).

Additionally, the execution speed for each full round of the Gossamer Protocol (144 – 148ms) is relatively high (reflecting the limitations of nRF9E5 processing power). The simplicity of the underlying mathematical calculations would imply fast performance. In fact the actual performance varies significantly from that expected. This may have adverse consequences on the efficiency of the communications protocol. Further code optimisation and/or native assembly code would reduce code space requirement and improve performance, but not by a magnitude large enough to justify the implementation of a software implementation of the Gossamer protocol on the reference platform. However, if another microcontroller without so strict memory limitations is used and the performance is regarded as satisfactory then the mechanism proposed can be considered for implementation. The SEA (96,8) implementation results were more promising than the Gossamer ones. As expected from an algorithm designed to be adapted easily to the native word size of the CPU, the code space footprint is very small (589 Bytes). Even when the RC51 libraries overhead is taken into consideration (552 Bytes), the total size of 1141 Bytes is just below 28% of the total code space available on the nRF9E5. SEA has not been proven to be insecure to date, thus it can be recommended for microcontroller implementations with associated low data throughput requirements. The code space requirement to implement Gossamer combined with the code space required by SEA is 3341 Bytes (2200 Bytes + 1141 Bytes) or 83% of available code space. The remainder of the code space is subsumed by simple radio functionality. Given the associated memory limitations, lack of hardware support for cryptographic primitives and the difficulty of implementing code banking with any degree of performance efficiency, the nRF9E5 cannot be recommended as a suitable platform on which to implement native authentication and encryption in security demanding wireless sensor networks. Low cost microcontroller alternatives, such as the Texas Instruments CC430 family of microcontrollers with an embedded UHF radio transceiver and hardware support for 128-bit AES encryption may be viable.

8. CONCLUSION

Lightweight Authentication and Encryption protocols have emerged to fill the security void created by the transition from desktop to mobile environments. Fast processing and large memory has characterised desktop technologies. By contrast, mobile technologies are characterised by their small processing power and small memory. Authentication and encryption protocols designed for

desktop technologies cannot be easily ported to mobile Resource Limited Devices (RLDs). The central theme of this work is that lightweight authentication and encryption protocols can fulfil the requirements of secure communications between RLDs without hardware modification. An augmentation of the Gossamer authentication protocol that incorporates elements of the Scalable Encryption Algorithm (SEA) was implemented to confirm this assertion. Cora Data's wireless sensor development board, comprising the Nordic Semiconductor nRF9E5 microcontroller and auxiliary radio communications circuitry was used as the reference platform. The implementation, in software, demonstrates successful accomplishment of the key objectives of secure communications, but at a cost. Success has been achieved by greatly simplifying the radio protocol and using almost the entire code space of 4 Kilobytes allowed by the nRF9E5 microcontroller for the implementation of the security mechanisms. As a consequence, there is zero code space left for the other tasks involved in the normal operation of an Infrastructure Wireless Sensor Network, such as sampling the ADC convertor and forming a data payload with the results.

The Gossamer and SEA protocols, are the most suitable of the family of ultra-lightweight security protocols for implementation on RLDs. The algorithms are current, resistant to attacks and cryptanalysis and their design has been focused on providing solutions for resource limited devices. In addition, they can be implemented on an 8 bit platform. An augmented Gossamer protocol that incorporates elements of the SEA is presented as a possible solution to the implementation of security in networks of RLDs. A major goal of this work was to examine code space requirements of the augmented protocol's implementation (since memory is a critical resource). The target is to provide secure communications with protocols that subsume as little of the memory as possible of the RLD. Although Gossamer uses basic mathematical operations, which are easy to implement in hardware, the software implementation on an 8-bit CPU involves a great deal of code space overhead. The performance analysis shows that the total code space required by the Gossamer functions (~1700 Bytes) including the necessary RC51 libraries (552 Bytes) can be estimated at approximately 2200 Bytes, which is 55% of the code space available on the reference platform. The overhead mainly relates to operations on large numbers that have to be split into arrays with elements equal to the word size of the CPU. This leaves little room for the implementation of the radio protocol (hence the need for simplification) and zero room for ADC or other functionality.

REFERENCES

Abramson, N. (1970, November 17-19). The aloha system: Another alternative for computer communications. *Proceedings of the 1970 fall joint computer conference* (pp. 281–285).

Ahmed, E. G., Shaaban, E., & Hashem, M. (2010). *Lightweight Mutual Authentication Protocol for Low Cost RFID Tags. International Journal of Network Security & Its Application (IJNSA), Academy & Industry Research Collaboration Center*. AIRCC.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*, *38*(4), 393–422. doi:10.1016/S1389-1286(01)00302-4

Andem, V. R. (2003). *A cryptanalysis of the tiny encryption algorithm*. Citeseer.

Bárász, M., Boros, B., Ligeti, P., Lója, K., & Nagy, D. (2007a). Breaking LMAP. *Proc. of RFIDSec*. Retrieved from http://www.cs.elte.hu/~turul/pubs/lmap.pdf

Bárász, M., Boros, B., Ligeti, P., Lója, K., & Nagy, D. (2007b). Passive attack against the M2AP mutual authentication protocol for RFID tags. *Proc. of First International EURASIP Workshop on RFID Technology*. Retrieved from http://www.cs.elte.hu/~turul/pubs/mmap.pdf

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J., & Vikkelsoe, C. et al. (2007). PRESENT: An ultra-lightweight block cipher. *Lecture Notes in Computer Science*, *4727*, 450–466. doi:10.1007/978-3-540-74735-2_31

Cao, T., Bertino, E., & Lei, H. (2009). Security Analysis of the SASI Protocol. *IEEE Transactions on Dependable and Secure Computing*, 6(1), 73–77.

Chien, H. (2007). SASI: A New Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity. *IEEE Transactions on Dependable and Secure Computing*, *4*(4), 337–340. doi:10.1109/TDSC.2007.70226

Chien, H., & Huang, C. W. (2007). Security of ultra-lightweight RFID authentication protocols and its improvements. *Operating Systems Review*, *41*(4), 83. doi:10.1145/1278901.1278916

D'Arco, P., & De Santis, A. (2008). *From Weaknesses to Secret Disclosure in a Recent Ultra-Lightweight RFID Authentication Protocol*. Retrieved from http://eprint.iacr.org/2008/470

Daemen, J. & Rijmen, V. (1999). AES proposal: Rijndael.

Eastlake, D. & Jones, P. (2001, September). *US secure hash algorithm 1 (SHA1)*, RFC 3174.

Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., & Uhsadel, L. (2007). A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, *24*(6), 522–533. doi:10.1109/MDT.2007.178

el Ruptor, M. (2010, September 19). File:XXTEA.png - Wikipedia, the free encyclopedia. Retrieved from http://en.wikipedia.org/wiki/File:XXTEA.png

EPCGlobal. (2008). EPCglobal UHF Class 1 Gen 2. Retrieved from http://www.epcglobalinc.org/standards/uhfc1g2

Federal Information Processing Standards. (1993). FIPS 46-2 - (DES), Data Encryption Standard. Retrieved from http://www.itl.nist.gov/fipspubs/fip46-2.htm

Hartung, C., Balasalle, J., & Han, R. (2005). Node compromise in sensor networks: The need for secure systems. *Department of Computer Science University of Colorado at Boulder*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.8146&rep=rep1&type=pdf

Hegazy, A.E., Darwish, A.M., & El-Fouly, R. (2007). Reducing μTESLA memory requirements.

Hernandez-Castro, J. C., Estevez-Tapiador, J. M., Ribagorda-Garnacho, A., & Ramos-Alvarez, B. (2006). Wheedham: An automatically designed block cipher by means of genetic programming. *Proc. of CEC* (pp. 192–199). doi:10.1109/CEC.2006.1688308

Hernandez-Castro, J. C., Tapiador, J. M., Peris-Lopez, P., & Quisquater, J. J. (2008). Cryptanalysis of the SASI Ultralightweight RFID Authentication Protocol with Modular Rotations. *Arxiv preprint arXiv:0811.4257*.

HINT Project. (2010). *Research Project: HINT Project*. Letterkenny Institute of Technology.

Hong, S., Hong, D., Ko, Y., Chang, D., Lee, W., & Lee, S. (2004). Differential Cryptanalysis of TEA and XTEA. *Information Security and Cryptology-ICISC*, *2003*, 402–417.

Jinwala, D.C., Patel, D.R., & Dasgupta, K.S. (2008). Investigating and Analyzing the Light-weight ciphers for Wireless Sensor Networks.

Juels, A. (2005). Strengthening EPC tags against cloning. *Proceedings of the 4th ACM workshop on Wireless security* (p. 76). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.6553&rep=rep1&type=pdf

Juels, A. (2006). RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications*, *24*(2), 381–394. doi:10.1109/JSAC.2005.861395

Kamble, P., Kshirsagar, R. V., & Mankar, K. (2007). Wireless Sensor Network Architecture. Retrieved from http://www.ieee-spce.org/colloquium/proceedings/Communication_and_Networking/spit-1.pdf

Karlof, C., Sastry, N., & Wagner, D. (2004). TinySec: a link layer security architecture for wireless sensor networks. *Proceedings of the 2nd international conference on Embedded networked sensor systems* (pp. 162–175). Retreived from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.4930&rep=rep1&type=pdf

Kelsey, J., Schneier, B., & Wagner, D. (1997). Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NEWDES, RC2, and TEA. *Proceedings of the first international conference on Information and Communications Security ICICS '97* (pp. 233–246).

Klimov, A., & Shamir, A. (2004). Cryptographic Applications of T-functions. *Lecture Notes in Computer Science*, *3006*, 248–261. doi:10.1007/978-3-540-24654-1_18

Ko, Y., Hong, S., Lee, W., Lee, S., & Kang, J. S. (2004). Related key differential attacks on 27 rounds of XTEA and full-round GOST. Fast Software Encryption (pp. 299–316).

Lee, Y. C., Hsieh, Y. C., You, P. S., & Chen, T. C. (2009). A New Ultralightweight RFID Protocol with Mutual Authentication. *Proceedings of the 2009 ICIE'09 WASE International Conference on.Information Engineering* (pp. 58–61). doi:10.1109/ICIE.2009.24

Leong, K. S., Ng, M.L., & Engels, D.W., 2006. EPC Network Architecture. *Auto-ID Labs: EPC Network Architecture*. Retrieved from http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-SWNET-012.pdf

Li, T., & Deng, R. (2007). Vulnerability analysis of EMAP-an efficient RFID mutual authentication protocol. *Proc. of AReS*, 7. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.6430&rep=rep1&type=pdf

Li, T., & Wang, G. (2007). Security analysis of two ultra-lightweight RFID authentication protocols. *INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING-PUBLICATIONS-IFIP*, *232*, 109–120. doi:10.1007/978-0-387-72367-9_10

Liu, D., & Ning, P. (2004). Multilevel μTESLA: Broadcast authentication for distributed sensor networks. [TECS]. *ACM Transactions on Embedded Computing Systems*, *3*(4), 800–836. doi:10.1145/1027794.1027800

ARM Ltd. (2009a). Keil C51 Compiler Basics. Retrieved from http://www.esacademy.com/automation/docs/c51primer/c02.htm

ARM Ltd. (2009b). LX51 User's Guide: Code Banking. Retrieved from http://www.keil.com/support/man/docs/lx51/lx51_codebanking.htm

Lu, J. (2009). Related-key rectangle attack on 36 rounds of the XTEA block cipher. *International Journal of Information Security*, *8*(1), 1–11. doi:10.1007/s10207-008-0059-9

Menezes, A. J., Oorschot, P. C. V., & Vanstone, S. A. (1997). *Handbook of applied cryptography*. CRC Press.

Mollin, R. A. (2007). *An introduction to cryptography*. CRC Press.

Moon, D., Hwang, K., Lee, W., Lee, S., & Lim, J. (2002). Impossible differential cryptanalysis of reduced round XTEA and TEA. Fast Software Encryption (pp. 117–121). doi:10.1007/3-540-45661-9_4

Needham, R.M., & Wheeler, D.J. (1997). *eXtended Tiny Encryption Algorithm*, October.

Ni, S. Y., Tseng, Y. C., Chen, Y. S., & Sheu, J. P. (1999). The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. (p. 162). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.5000&rep=rep1&type=pdf

Nordic Semiconductors. (2009a). NORDIC SEMICONDUCTOR - nRF905 Multiband Transceiver. Retrieved from http://www.nordicsemi.com/index.cfm?obj=product&act=display&pro=83

Nordic Semiconductors. (2009b). NORDIC SEMICONDUCTOR - nRF9E5 Multiband Transceiver/MCU/ADC. Retrieved from http://www.nordicsemi.com/index.cfm?obj=product&act=display&pro=82

Ouafi, K., & Vaudenay, S. (2009). Smashing SQUASH-0. Proceedings of the Advances in Cryptology - EUROCRYPT 2009 (pp. 300-312). doi:10.1007/978-3-642-01001-9_17

Peris-Lopez, P., Hernandez-Castro, J., Tapiador, J., & Ribagorda, A. (2009). Advances in Ultralightweight Cryptography for Low-Cost RFID Tags: Gossamer Protocol. Information Security Applications (pp. 56–68).

Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., & Ribagorda, A. (2006a). EMAP: An efficient mutual-authentication protocol for low-cost RFID tags. *Lecture Notes in Computer Science*, *4277*, 352–361. doi:10.1007/11915034_59

Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., & Ribagorda, A. (2006b). LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags. In *Workshop on RFID Security* (pp. 12–14). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.2082&rep=rep1&type=pdf

Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., & Ribagorda, A. (2006c). M^2AP: A Minimalist Mutual-Authentication Protocol for Low-Cost RFID Tags. *Lecture Notes in Computer Science*, *4159*, 912–923. doi:10.1007/11833529_93

Peris-Lopez, P., Hernandez-Castro, J. C., Tapiador, J. M., van der Lubbe, J. C., Singh, M. K., Liang, G., . . . Kish, L. L. (2008). Security Flaws in a Recent Ultralightweight RFID Protocol. *Arxiv preprint arXiv:0910.2115*.

Perrig, A., Canetti, R., Song, D., & Tygar, J. D. (2001). Efficient and secure source authentication for multicast. Proceedings of the Network and Distributed System Security Symposium (pp. 35–46). NDSS. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.1680&rep=rep1&type=pdf

Perrig, A., Szewczyk, R., Tygar, J. D., Wen, V., & Culler, D. E. (2002). SPINS: Security protocols for sensor networks. *Wireless Networks*, *8*(5), 521–534. doi:10.1023/A:1016598314198

Poschmann, A., Leander, G., Schramm, K., & Paar, C. 2007. New light-weight crypto algorithms for RFID. *Proceedings of the IEEE International Symposium on Circuits and Systems* (pp. 1843–1846). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.1217&rep=rep1&type=pdf

Rabin, M.O. (1979). Digitalized signatures and public-key functions as intractable as factorization. *MtT/LCS/TR-212*.

Raisonance, S. A. S. 2010. Raisonance, Corporate home page. Retrieved from http://www.raisonance.com/

Ranasinghe, D. C., & Cole, P. H. (2008). *Networked RFID Systems and Lightweight Cryptography*. Springer Berlin Heidelberg. doi:10.1007/978-3-540-71641-9

Rinne, S., Eisenbarth, T., & Paar, C. (2007). *Performance analysis of contemporary light-weight block ciphers on 8-bit microcontrollers*. ECRYPT.

Rivest, R. L. (1995). The RC5 encryption algorithm. *Dr. Dobb's Journal of Software Tools for the Professional Programmer*, *20*(1), 146–149.

Russell, M.D. (2004). Tinyness: an overview of TEA and related ciphers (Draft v0.3, 3).

Saarinen, M. J. (1998). *Cryptanalysis of Block Tea* (unpublished manuscript).

Sarma, S. E. (2001). *Towards the five-cent tag* (Technical Report MIT-AUTOID-WH-006). MIT Auto ID Center. Retrieved from http://www.autoidlabs.org/uploads/media/mit-autoid-wh-006.pdf

Sarma, S. E., Weis, S. A., & Engels, D. W. (2003). RFID systems and security and privacy implications. *Lecture Notes in Computer Science*, *2523*, 454–469. doi:10.1007/3-540-36400-5_33

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (2nd ed.). Wiley.

Shamir, A. (2008). SQUASH – A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags. *Fast Software Encryption* (pp. 144–157). doi:10.1007/978-3-540-71039-4_9

Standaert, F., Piret, G., Gershenfeld, N., & Quisquater, J. (2006). SEA: A scalable encryption algorithm for small embedded applications. *Lecture Notes in Computer Science*, *3928*, 222–236. doi:10.1007/11733447_16

Sun, H. M., Ting, W. C., & Wang, K. H. (2008). *On the security of chien's ultralightweight RFID authentication protocol* (Report 2008/083). Cryptology ePrint Archive. Retrieved from http://eprint.iacr.org/2008/083.pdf

Vault Information Services. (2009). 8052.com - The Online 8051/8052 Microcontroller Resource. Retrieved from http://www.8052.com/

Wheeler, D., & Needham, R. (1994). TEA, a tiny encryption algorithm. Fast Software Encryption (pp. 363–366).

Wheeler, D., & Needham, R. (1998). *XXTEA: Correction to XTEA, Technical report*. Computer Laboratory, University of Cambridge.

Wi-Fi Alliance. (2003). Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks.

Yarrkov, E. (2010). *Cryptanalysis of XXTEA*. Retrieved from http://eprint.iacr.org/2010/254

Ye, W., Heidemann, J., & Estrin, D. (2002). An energy-efficient MAC protocol for wireless sensor networks. Proceedings of IEEE INFOCOM (pp. 1567–1576).

Yu-Long, S., Qing-Qi, P.E.I. & Jian-Feng, M.A. (2007). microTESLA: Broadcast Authentication Protocol for Multiple-Base-Station Sensor Networks.

ENDNOTES

[1] L denotes the length of one key or the IDS in bits. 96-bits in the case of the EPC RFID specifications.
[2] Data freshness ensures that the data received is fresh and the adversary cannot replay old messages.
[3] Semantic security ensures that an eavesdropper is not able to deduct any information about the plaintext even after analysis of multiple encryptions of the same plaintext.