STFTP: Secure TFTP Protocol for Embedded Multi-Agent Systems Communication

Goran HORVAT, Drago ŽAGAR and Goran MARTINOVIĆ
Faculty of Electrical Engineering, J. J. Strossmayer University of Osijek, Croatia
goran.horvat@etfos.hr

***Abstract*—Today's embedded systems have evolved into multipurpose devices moving towards an embedded multi-agent system (MAS) infrastructure. With the involvement of MAS in embedded systems, one remaining issues is establishing communication between agents in low computational power and low memory embedded systems without present Embedded Operating System (EOS). One solution is the extension of an outdated Trivial File Transfer Protocol (TFTP). The main advantage of using TFTP in embedded systems is the easy implementation. However, the problem at hand is the overall lack of security mechanisms in TFTP. This paper proposes an extension to the existing TFTP in a form of added security mechanisms: STFTP. The authentication is proposed using *Digest Access Authentication* process whereas the data encryption can be performed by various cryptographic algorithms. The proposal is experimentally tested using two embedded systems based on micro-controller architecture. Communication is analyzed for authentication, data rate and transfer time versus various data encryption ciphers and files sizes. STFTP results in an expected drop in performance, which is in the range of similar encryption algorithms. The system could be improved by using embedded systems of higher computational power or by the use of hardware encryption modules.**

***Index Terms*—software agents, embedded system, telecommunication, multiagent system, security, Secure TFTP, TFTP**

I. INTRODUCTION

With the overall rising trend in use and production of embedded systems, the impact on establishing effective and low cost communication over the existing infrastructure is more and more emphasized. According to [1], semiconductor and embedded industry is projected to bloom from $3.25 billion in 2005 to $43.7 billion by 2015. Consequently, the field of embedded systems now influences many industrial sectors including automotive, consumer electronics, communications, medical and other, representing an interesting area for research. Due to the fact that this growth is propelled by the penetration of stand-alone low cost chips such as microprocessors and microcontrollers [1], the question that arises is how to establish effective and secure way of communication in these configurations.

As the complexity of an embedded system rises the need to replace human interaction in embedded systems becomes increasingly important. The introduction of agent technologies into embedded systems (bringing intelligence and flexibility) presents a problem, as these devices are of low computational power and low memory [2]. This problem can be avoided by designing an eMAS (embedded Multi-Agent System) within boundaries of an embedded system. The question that remains is how to establish effective communication between agents (embedded systems) in MAS [3]? In certain applications mobile agents can be located within Wireless Sensor Network (WNS) nodes where the communication between WSN nodes is separated from the "real world"; presenting no need for additional communication protocols. In this scenario communication is carried out through WSN protocol stack [4]. On the other hand, large number of embedded systems imposes a need for embedded agent integration, where the communication component presents a setback for implementation. The problem here lies in finding an effective and secure communication protocol for agent exchanging messages (agent interactions) and exchanging data whilst stepping within computational memory boundaries of an embedded system.

Modern embedded systems are composed of embedded hardware (microcontroller) and embedded software (often an embedded agent) that incorporates a data storage device. In order for agents to effectively transfer data files and messages from and to the destination, the embedded system must incorporate a file transfer protocol. Because the FTP protocol induces fairly large overhead on the existing software, a viable alternative to this problem is the use of *Trivial File Transfer Protocol* (TFTP). Unlike FTP, the TFTP protocol is based on UDP transport protocol and effectively simplifies the implementation.

A major drawback of the TFTP is not providing any form of security mechanism (primarily authentication and data encryption). Therefore, this paper proposes an extension to the existing protocol by adding security mechanisms in a form of *Digest Access Authentication* accompanied by *Secure Hash Algorithm* 1 (SHA-1) to establish a secure way of agent authentication. To assure the data confidentiality, two ciphers are proposed: *Advanced Encryption Standard* (AES) and *eXtended Tiny Encryption Algorithm* (XTEA).

In order to verify the proposal, the test setup is implemented on two different embedded systems based on different architectures. Further on, the database is stored on a *micro Secure Digital* data card (*microSD*) and an *Ethernet* interface is used. The tested architectures differ in microcontroller type, of which the former is an *Atmel XMEGA* based microcontroller and the latter is *Microchip PIC32* based microcontroller. Both test setups consists of *Stand Alone Ethernet Controller ENC28J60* functioning as a link to the *Ethernet* LAN, incorporating only Physical and Medium Access Control layer. Higher layer support is added

This work was sponsored by the Ministry of Science, Education and Sports of the Republic of Croatia under project 165-0362027-1479 and 165-0362980-2002.

Digital Object Identifier 10.4316/AECE.2013.02004

through embedded system's firmware.

The testing of the proposed extension was carried out using Network Protocol Analysis. The data were analysed regarding data rate, transmission time and authorization flow versus different data encryption ciphers. Data rate and transfer time were analyzed versus different file sizes and different data encryption ciphers.

In the following Section related work on TFTP and security aspects in embedded systems use are described. Section III describes the implementation of authentication mechanism in TFTP. Section IV proposes the data encryption accompanied by various ciphers. In Section V experimental setup is described and measurement results are analyzed, respectively. Section VI gives the conclusion with added future work.

II. EMBEDDED MAS ARCHITECTURE

An agent is interactive software designed to avoid the need for human interaction to a certain degree. By definition, a software system of autonomy, social ability, reactivity and pro-activeness can be called as agent [5]. The agents that can be moved within a network are called mobile agents and together they form a multi-agent system or MAS. If examined from another point of a view, agent is a programming thought rather than a particular technique. It changes the notion of software and presents new ways to bring more intelligence to the internet [6][12]. Example of MAS is shown on Figure 1.

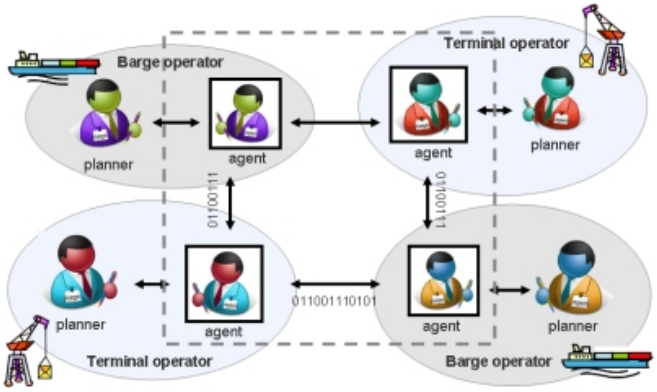


Figure 1. An example of MAS [7]

On the other hand, the difference between a standard agent and an embedded agent is the point of the implementation. According to [3] agents of eMAS have the following constrains:

- Low memory resources (*16K-32K)*
- Low core frequency (*1MHz-4MHz)*
- Autonomous energy management

Consequently, upon eMAS implementation these constraints must be taken into consideration. A major goal of embedded software design is to construct the environment where the agents can live. This requires the existence of an EOS (Embedded Operating System) layer that is often not present in low power configurations. The basic link protocols, such as TCP/IP, and so far as web service, are located in this layer, so the absence of EOS layer results in the absence of higher layer communication protocols. This presents a problem for embedded agents as their behaviour requires social interaction with other agents [6].

This problem can be solved throughout various hardware dependant solutions. For instance, an embedded multi-agent system implemented in Wireless Sensor Network (WSN) experiences all the benefits that EOS provides in the area of communication in the manner that every WSN node already incorporated a communication stack (such as ZigBee, IEEE802.15.4) [8]. With the included communication stack, sending messages to other WSN nodes and embedded agents is substantially simplified. The agents can easily exchange messages and need not to worry about security or protocol issues. However, even in these systems there is a wide area for application layer protocol development.

On the other hand, establishing communication in embedded system without an existing backbone network can present a demanding task. For instance, a microcontroller embedded system with an implemented agent has to have the ability to contact other agents in vicinity, but it lacks the memory for the implementation of a standard TCP/IP stack. One of the existing solutions is presented in [9] where a hardware module is used to establish communication with an *Ethernet* network using an existing single chip solution. The advantage here is the lower layer support implemented within the communication chip (PHY and MAC layers), discussed in detail in Section III. This simplifies the implementation but still requires higher layer communication protocol support. To avoid the additional induced overhead that the higher layer protocol induces, this paper proposes the use of a very simple yet outdated communication protocol named Trivial File Transfer Protocol (TFTP). The main advantage of TFTP is the low induced overhead for implementation. However, the existing problem with TFTP is the overall lack of security mechanisms that renders the protocol useless for real versatile *Ethernet* environments.

Throughout this paper a solution for communication protocol between agents in eMAS is presented in a form of a STFTP protocol that enables an efficient implementation while retaining communication security. Also, the choice of security is presented in a form of various encryption ciphers that can vary depending on the application at hand.

III. TFTP AND SECURITY ASPECTS IN EMBEDDED SYSTEMS

A. TFTP and Embedded Systems

TFTP is a simple protocol designed for simple data file transfer. It is built on top of the Internet *User Datagram Protocol* (UDP) using port number 69. It is designed to be easy implemented, representing an ideal candidate for use in low cost embedded systems [10]. TFTP defines three modes of transfer: *netascii*, *octet*, and *mail*. Octet mode allows the transfer of arbitrary bytes, as opposed to the *netascii* (uses only ASCII characters). Transfer mode named *mail* is used to relay *e-mail* messages and can be effectively used to relay messages between agents in eMAS, if agent names are substituted for *mail* addresses [11].

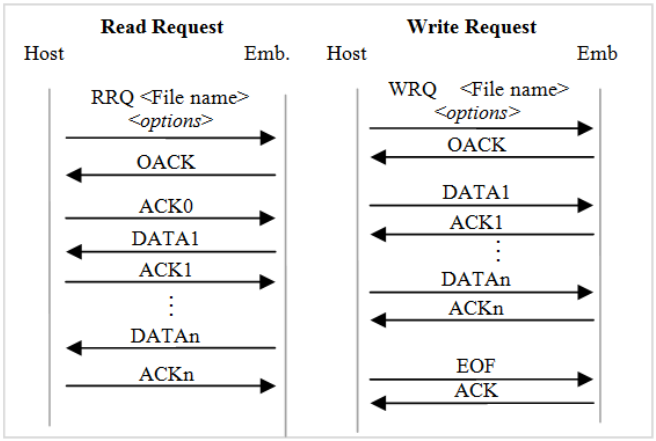


Figure 2. TFTP data flow

As shown on Figure 2, file transfer begins with a request to read or write a file, also serving as a request for connection. If the server grants the request, the server responds with *Option Acknowledgement* (OACK). The connection is opened and the file is sent in fixed length blocks (512 bytes or larger, depending on block size (*blksize)* parameter) [11].

The TFTP protocol uses a lock-step algorithm. A data packet of size less than *blksize* parameter indicates the termination of a transfer. After sending a block of data, sender should wait for an acknowledgement. If a packet is lost in the network, the receiver will timeout and may retransmit the last received packet, indicating the sender to retransmit the last packet. The sender has to keep just one packet for retransmission, since the lock step acknowledgment guarantees that all previous packets have been received. The transfer rate is therefore limited to the round-trip time (RTT). If the round-trip time is e.g. 20 ms, then the transmitter can send up to 50 blocks per second. With a default block size of 512 bytes, the transfer rate is bounded to 25 kB/s [1][10][11].

In order to incorporate the UDP communication in embedded system, a *Network Interface Controller* (NIC) must be present to establish the communication between the micro controller (embedded system) and a *Local Area Network*. Most of the existing microcontrollers are not designed to incorporate the desired functionality in their design, making the implementation more complex [1].

On the other hand, presently only a small number of microcontrollers incorporate the necessary hardware and software requirements to enable *Ethernet* communication (e.g. Microchip PIC18F97J60)[13]. To overcome the lack of functionality in the existing systems it is possible to use existing *Stand Alone Ethernet Controller* that is interfaced in familiar industry standard. One of the solutions is to use Microchip's *Stand-Alone Ethernet Controller ENC28J60*, which uses SPI interface to relay data to and from embedded system. ENC28J60 incorporates integrated 10BASE-T MAC and PHY support, with the support for UDP communication [13]. This choice presents a cost effective and easy implementation solution for embedded systems without integrated *Ethernet* interface.

The major problem of TFTP use in *Ethernet Local Area Network* is the lack of authentication and data encryption security mechanisms. This presents a serious problem in versatile Ethernet environment, so an extension in a form of additional TFTP security mechanisms should be implemented.

B. Communication security in embedded systems

Related work on this subject is versatile and covers a wide area. Primarily, the security issues in embedded systems are largely discussed in [14], where the impacts of current technologies are shown. Foremost, the use of communication protocols and standards (such as SSL and WEP) for secure communication presents a problem for embedded system implementation. Many embedded systems are constrained by the environments they operate in, and by available resources [14]. The implementation of these mechanisms presents a problem for the low cost, low power embedded systems, as the size of the protocol stack is too large. Another aspect of embedded system communication is the overall data rate. According to [15], the implementation of the protocols such as SSL diminishes the communication performance in embedded environments. The work on improving the efficiency of the SSL encryption was presented by various researchers, not resulting in significant improvement [15].

The emerging new technologies and next generation high speed protocols are swiftly taking their place in the everyday communication. The example is the UDT protocol that relies on UDP socket presenting protocol applicable for embedded systems [16]. On the other hand, the implementation of the UDT in embedded systems presents a problem since UDT requires the use of the UDT socket represented on a higher protocol layer.

The problem of authentication and security in embedded systems is specially addressed by stressing the low computational power of embedded systems for effective implementation of cryptographic algorithms [17]. Further on, some solutions incorporate a very complicated authentication system that uses a time synchronization server in the process of authentication, acting as a trusted third party [17]. This presents a problem for small networks, since the presence of the time synchronization server is required. Furthermore, the proposed solution does not incorporate any form of data encryption thus not retaining the confidentiality of the transmitted data [17]. In addition, the cryptographic primitive MD5 (used in *Digest Access Authentication*) is not secure from cryptographically aspect as it can be easily broken in a few seconds [19]. Therefore, an alternative authentication mechanism should be proposed.

From the point of implementation in embedded systems, TFTP presents an ideal candidate. On the other hand, the extension of the TFTP protocol towards a secure communication is not adequately researched and documented. In order to improve the security of the TFTP protocol, the authentication procedure is required for controlled access to embedded system. By implementing data encryption it is possible to ensure data confidentiality.

IV. AUTHENTICATION IMPLEMENTATION IN TFTP

The authentication procedure applied to low computational embedded systems can be of various complexity levels. Taken into consideration the overall complexity of the authentication algorithm, the

cryptographic authentication presented in [17] is fairly complex as it requires the use of a time synchronization server. To lower the complexity of the authentication as well as to retain the level of security, this paper proposes a modification of the existing *Digest Access Authentication (DAA)* for use in embedded systems. The alternative to using DAA is the Datagram Transport Layer Security (DTLS), however the implementation of DTLS results in higher system complexity and larger overhead.

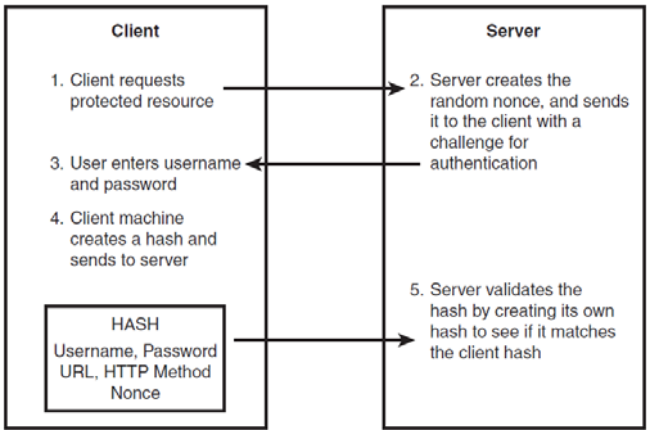


Figure 3. HTTP Digest Access Authentication

Digest Access Authentication (also known as Digest - MD5) was originally proposed to provide peer authentication in HTTP protocols (Figure 3.) [18]. Most prevalent browsers have substantially implemented the specification of Digest-MD5, including Mozilla Firefox, Netscape, Konqueror, Internet Explorer and Google Chrome [19]. Digest-MD5, a typical hash-based challenge and response protocol, intends to provide protection against replay attacks, on-line dictionary attacks and choosing plain text attacks, et al. Digest-MD5 is one of the most widely used cryptographic hash functions, to hash some shared secret information needed for authentication. The authentication server first sends a random string (hashed using client IP address, time stamp and private key) called a *nonce* as a challenge to the peer, and the peer generates the corresponding response.

However, according to [19], the overall security of MD5 based *Digest Access Authentication* is compromised, and all applications based on that must be re-evaluated seriously. According to [23], a 2009 attack by Tao Xie and Dengguo Feng breaks MD5 collision resistance using just $2^{20.96}$ time complexities. This attack runs in a few seconds on a regular computer. In order to preserve the implementation simplicity of the *Digest Access Authentication* as well, to improve the overall security of authentication, this paper proposes the use of Secure Hash Algorithm 1 (SHA-1) instead of MD5. Some preliminary results indicate that SHA is 62% as fast as MD5 presenting a perspective alternative in the processing aspect [20]. According to [21], a 2008 attack by Stéphane Manuel can break SHA-1 hash function by producing hash collisions with complexity of 2^{51} operations. This presents a significant improvement in comparison to MD5 hash function as well as a generally safer authentication process. To additionally secure authentication process it is possible to use SHA-256 hash function. However, this can additionally burden effective embedded system implementation.

To enable any form of authentication, an important segment is the generation of random *nonce* values used in authentication procedure.

A. Generating nonce values in eMAS

In cryptography, *nonce* is an arbitrary number used only once in a cryptographic communication. Most commonly it is a random or pseudo-random number issued in an authentication protocol to protect against replay attacks. For instance, *nonces* are used in HTTP digest access authentication to calculate an MD5 digest of the password. Since *nonce* values are said to be random, an effective way of generating these values must be proposed.

An interesting approach in generating random or pseudo-random values is the use of a universally unique identified or UUID and a hashing function. This claim is supported by the claim that the hash function statistically eliminates the possibility of generating two identical values from two different inputs [21].

Further on, by the definition UUID is designed as a number that is globally unique in space and time. Two calls to UUID() function are expected to generate two different values, even if these calls are performed on two separate devices that are not connected to each other [22]. This presents a basis for *nonce* generation, however some drawbacks exist. For instance, to form UUID (Version 1.) basic parameters are the MAC address and a date-time (100ns based). These parameters do not impose a problem on normal personal computers, but on embedded systems due to the limited resources, these parameters could be difficult to acquire. However, according to [22] there is a solution for the generation of unique identifiers specifically designed for MAS named GHUUID (Geo-Hash Universally Unique Identifier). Geohash+UUID (GHUUID) mechanism takes the location based data with UUID function. GHUUID gives the agents spatio-temporal awareness and with its uniqueness it guarantees the generation of fully random *nonce* values.

Example of GHUUID:
(u2j70vx29gfu-2d004620-3fc7-11e1-b86c-0800200c9a66)

- First part: *u2j70vx29gfu* represent GeoHashed lat/lon information that corresponds to the initial geo coordinates of *45.557, 18.675*.
- The second part: *2d004620-3fc7-11e1-b86c-0800200c9a66* is generated UUID that contains a timestamp taken at Sunday, January 15, 2012 10:20:48 PM GMT.

The main drawback of this hashing method is the need for spatial coordinates and precise time stamp. However, this data can be easily provided by the GPS (Global Positioning System) implemented on an embedded system. The choice for the use of GPS lies on the fact that one of the first MAS applications was distributed vehicle monitoring (DVMT) where a set of geographically distributed agents monitor vehicles that pass through their respective areas, attempt to come up with interpretations of what vehicles are passing

through the global area, and track vehicle movements [22]. Due to the fact that these agents are mobile, their main prerequisites for mobility is the knowledge of their spatial position provided by the GPS system, thus providing the hashing algorithm with all the data required to compose a GHUUID.

However, in applications with no GPS support the generation of universally unique identifiers can be performed using simple UUID (Version 1) that for input takes a MAC address and a timestamp. Due to the fact that the MAC address is a global identifier (organized usually around the central issuing authorities who can guarantee their uniqueness) the temporal uniqueness of the UUID is guaranteed.

After the calculated GHUUID or UUID the embedded system hashes the *nonce* in the following manner:

$$nonce = SHA1("host\_ip : (GH)UUID") \quad (1\text{-}1)$$

where the *host_ip* represents the IP address of the host device, *(GH)UUID* represents a Geo hashed Universally Unique Identifier (GHUUID) or a Universally Unique Identifier Version 1. (UUID). When *nonce* value is generated and hashed using SHA-1 hash function, it is ready for the use in the authentication procedure.

B. Digest Access Authentication procedure

The proposed *Digest Access Authentication* using SHA-1 follows the prescribed procedure: Host requests the *nonce* value from embedded system. When *nonce* value is generated and hashed using SHA-1 hash function, it is transmitted to the host device. After the host receives the *nonce*, it hashes two additional strings and joins them with the *nonce* hash string:

$$HA1 = SHA1("username : realm : password")$$
$$HA2 = SHA1("method : uri") \quad (1\text{-}2)$$

where the *username* represents the standard username, *realm* represents the realm received from embedded device and *password* represents the password for the given username. After the two hash strings are generated and combined with *nonce* value, the response is sent to the embedded system.

$$response = SHA1("HA1 : nonce : HA2") \quad (1\text{-}3)$$

Embedded system recalculates the response from previously randomly generated *nonce* value and stored hash strings *HA1* and *HA2* (stored in database on *microSD* data card). If the generated response on the embedded system matches the received hash string the communication is signed and the host is authenticated. The embedded system establishes a virtual channel to the host device using specified IP address and port number. All requests received from another IP addresses will be discarded. In order to prevent access from IP address spoofing, the sent and received requests as well as the data are encrypted.

The data encryption key is derived from *nonce* string and stored HA1 hash string. The key can be represented as follows:

$$HA3 = SHA1("HA1 : nonce")$$
$$ENCRYPTION\_KEY = HA3 \cap (2^{128} - 1) \quad (1\text{-}4)$$

The encryption key is represented by first 128 bits of the 160 bit hash string. Due to the fact that the hash string HA1 is secret and *nonce* string is dynamically generated, the selected combination ensures a dynamic change of the encryption key. This presents a novelty in key generation and ensures that the encryption key changes for every session. Another advantage of this method is eliminating the need for a central authority for the distribution of keys that would present a drawback in embedded system design.

The control flow of authentication and hand shaking of the *nonce* and *response* strings are displayed on Figure 4.

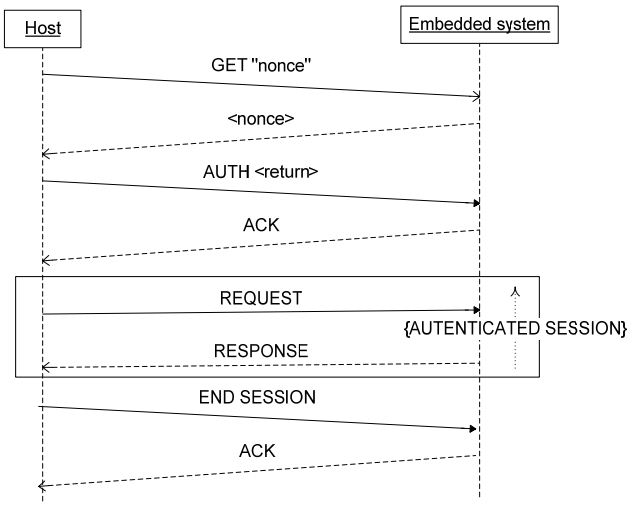


Figure 4. Digest Access Authentication flow

After transfer session is completed, the host closes the channel the data and terminates the session. In order to terminate the session the host needs to send the end session request alongside with the initially acquired *response* value. The sent *response* value imposes a form of security mechanism preventing direct *DoS* attacks by the malicious hosts. However, the imposed mechanism is vulnerable to replay attack, therefore additional security mechanisms should be implemented in future work.

C. Authentication in the TFTP frame

In order to enable the TFTP support for the authentication, message format and the frame of the TFTP must be adapted. Specifically, the support for GET, AUTH and END SESSION requests must be implemented in the existing message format of the *Trivial File Transfer Protocol*. The section that defines the type of the packet is named *opcode,* and the original *opcodes* for TFTP are shown in Figure 4 [25]. The *opcodes* omitted from Figure 5 are defined by authors in [11], as well as the optional fields added in read and write requests.

Format of the TFTP frame is defined by [9] and throughout sent and received requests the desired parameters are being exchanged. By examining TFTP *opcodes* it was concluded that *opcodes* with hexadecimal prefix 0x10 are not used in the standard TFTP, so the STFTP uses hexadecimal prefix 0x10 to define authentication headers.

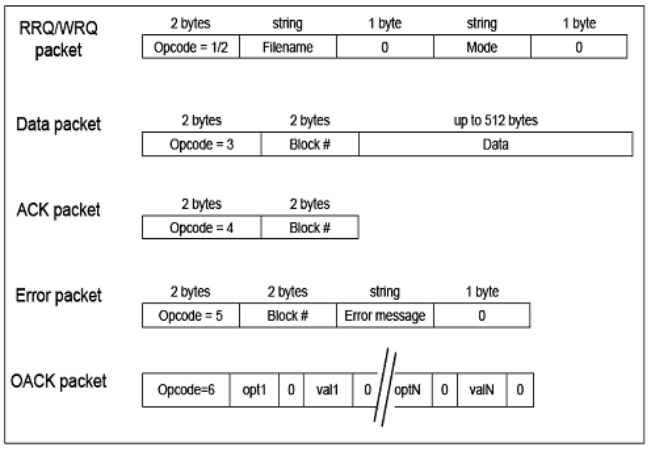


Figure 5. Format of the existing TFTP

This paper proposes the addition of five additional *opcodes* to enable the process of authentication. The added *opcodes* include GET_NONCE, RETURN <nonce>, END_SESSION, AUTH <return> and AUTH_ACK. The new proposed *opcodes* are shown in Figure 6.

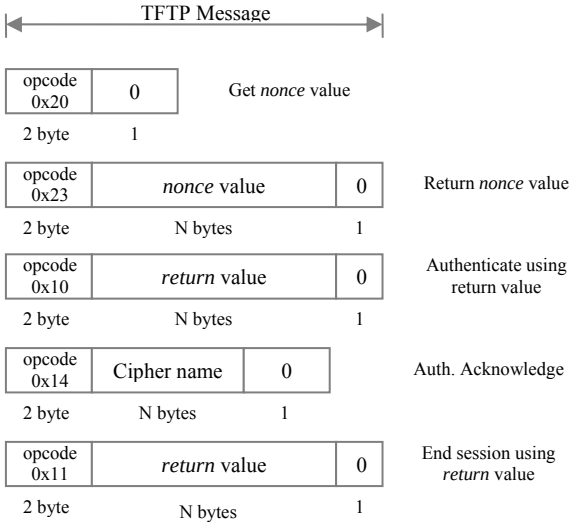


Figure 6. Proposed authentication opcodes

By adding the new *opcodes* the implementation of the authentication mechanism in TFTP is enabled. Authentication process is a critical component in designing secure communication. Another novelty is the support of different data ciphers within the protocol (defined by AUTH_ACK *opcode*). Accordingly, to ensure desired data security level, confidentiality must be addresses as well.

V. Establishing Confidentiality in TFTP

Establishing confidentiality is the process in which the body of the message is encrypted. Encrypting the entire message body can be computationally expensive, depending on the volume of data, the hardware limitation, and the encryption algorithm, respectively. Therefore, careful considerations should be made before implementing this process indiscriminately [26]. The cipher algorithm is executed upon reading and writing a data segment into/from the DATA packet (Figure 4) were the encryption key is derived from the equation (1-4).

The proper choice of data ciphers in the encryption process is very demanding, especially for embedded system engineer. Various software algorithms have been proposed for use in embedded systems (i.e. microcontrollers). For example, the implementation of the *Blowfish* algorithm in microcontroller architecture is proposed in [27]. On the other hand, some proposal uses a software implementation of a cipher known for its simplicity of implementation, resulting in only a few lines on code. This cipher in *eXtended Tiny Encryption Algorithm* (XTEA) presenting a suitable implementation for embedded systems [28].

Next on, various ciphers for embedded systems use (e.g. AVR microcontrollers) are analyzed and documented regarding different parameters in [29], [30] and [31]. These algorithms include AES, Dragon, SOSEMANUK and many others, which are analyzed regarding memory allocation and throughput on an 8MHz CPU clock. By combining the results from XTEA analysis [28], we can summarise and compare the results for the different chosen algorithms, shown in Table 1 [28][29].

TABLE I
THROUGHPUT AND MEMORY USAGE FOR VARIOUS SOFTWARE CIPHERS

Cipher	Block [byte]	Flash size [byte]	Encryption [cycles]	Encryption [cycles/by]	Throughput [by/s]
AES	16	3 410	3 766	235,4	33 985
Dragon	128	57 434	24 227	189,27	42 267
SOSEMANUK	80	44 704	14 134	176,68	45 279
XTEA	8	224	6 347	793,38	10 083
HC-128	64	23 100	10 804	161,81	47 390
LEX	40	21 312	8 061	201,53	39 696
Salsa20 imp.	64	3 842	48 942	764,72	10 461
DES	8	4 314	8 633	1 079,9	7 408
PRESENT	8	936	10 723	1 340,7	6 067
IDEA	8	596	2 700	337,5	23 703

Data shown on Table 1 can be easily presented in a bar graph form where the most important parameters are shown. These parameters are Flash size and Throughput. Ciphers are shown on Figure 7.

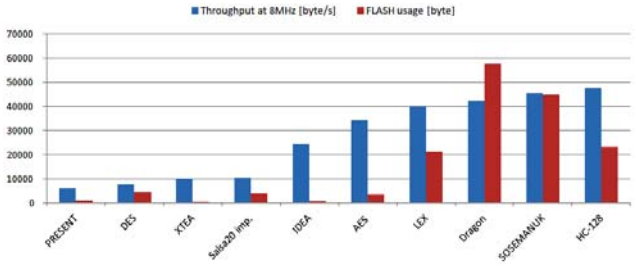


Figure 7. Overview of the contemporary software ciphers

In order to effectively choose between an effective implementation (low flash usage) and high data transfer rate (high throughput) the ratio between the parameters is proposed. The proposed ratio dictates the effectiveness of the cipher and the larger the ratio gets the cipher is said to be more effective. The ratios for the presented ciphers are shown on Figure 8.

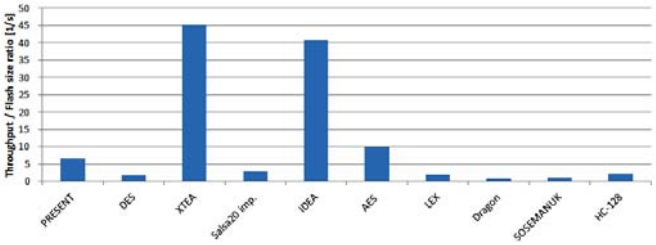


Figure 8. Throughput and Flash size ratio representing the effectiveness of software ciphers

As seen from Figure 8, the most effective cipher is the XTEA and IDEA ciphers, as they generate least Flash overhead and results in a moderate throughput. As shown in Table 1, the lowest overhead is generated by the XTEA algorithm, as it is the simplest algorithm for implementation [28]. Also, the XTEA uses fewer cycles for encryption, however due to the low block size, the overall throughput is lower. This is a trade-off for high throughput (using embedded systems of larger flash size) and low flash size for microcontrollers (lower throughput).

The next most efficient cipher is the AES algorithm as it generates high throughput and its implementation induces moderate overhead. The main advantage of using AES cipher is the high security that it offers. Also, AES is widespread algorithm also implemented as an ASIC solution, avoiding the need for software algorithms.

The algorithms such as Dragon, SOSEMANUK and others generate high throughput, however the flash size restricts them from being implemented in low memory embedded systems, as they generate a large overhead and are least effective (e.g. on a microcontroller of 128kB of memory the Dragon cipher uses 43% of the overall flash [29]). Due to the various used compilers, the induced overhead and the encryption cycles varies, therefore this should be taken into consideration [28][29][30].

By comparing results of available software ciphers for use in embedded systems the choice is reduced to AES and XTEA, presenting a compromise solution by overall throughput. The reason for not choosing other algorithms was the idea of demonstrating general implementation of a cipher. The IDEA cipher presets a very similar cipher to XTEA and TEA, but it incorporates more complex arithmetic operations (such as modulo 2^{32} multiplication) so for the purpose of demonstration this cipher was omitted from the implementation.

This paper proposes the use of both AES and XTEA ciphers, presenting the strengths and drawbacks for each of them. Alternately, the choice of ciphers could be left to the embedded software agent giving the agent more freedom in communication. The choice of ciphers is enabled from the added authentication procedure and the AUTH_ACK frame, within which it is possible to define what cipher is used in the communication.

A. *Hardware vs. Software ciphers*

The proposed ciphers are software algorithms that need to be implemented in the overall code. However, there are alternatives for using software ciphers. By using designated hardware modules integrated in certain integrated circuits, the encryption can be achieved on a hardware level. These hardware modules consist of logical gates in a standard-cell ASIC implementation.

A practical use example is the *Atmel's XMEGA* series microcontroller. The XMEGA incorporates an integrated AES and DES cipher module that uses 375 clock cycles before the encrypted/decrypted cipher text/plaintext is available for readout in the state memory [38]. This is a major advantage over the existing software ciphers (the difference is 34 times, Table 1).

The hardware implementation various cipher in embedded systems currently presents the single alternative to the highly overhead software based ciphers. According to [31] there are numerous hardware ciphers that can be implemented in ASIC solutions. These ciphers are shown on Table 2.

TABLE II
THROUGHPUT AND MEMORY USAGE FOR VARIOUS HARDWARE CIPHERS

Cipher	Block [byte]	Number of gates	Encryption [cycles]	Encryption [cycles/by]	Throughput [kby/s]
mCRYPTON	8	2500	13	0,2	4 923
CLEFIA	16	4950	36	0,28	3 556
PRESENT	8	1000	32	0,5	2 000
HIGHT	8	3048	34	0,53	1 882
AES	16	3100	156	1,22	800
XTEA	8	3490	109	1,7	571
DESXL	8	2168	64	1	444
DES	8	2300	64	1	444
AES [38]	16	3400	375	2,93	333

Graphical representation of the ciphers alongside with the number of gates and throughput is shown on Figure 9.

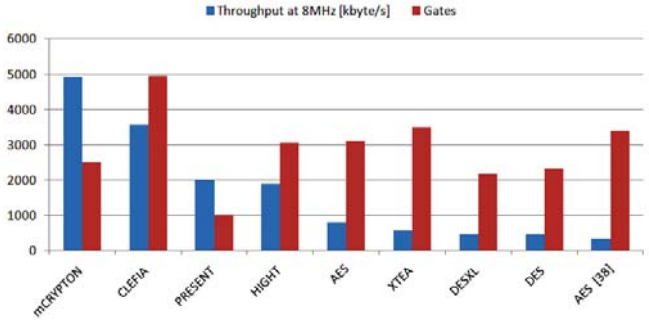


Figure 9. Overview of the contemporary hardware ciphers

As seen from Figure 9, the throughput of hardware based ciphers in almost 1000 times higher than the throughput of the software ciphers. The main reason for the vast difference in throughput is the low number of cycles for encoding resulted from parallel processing of the data. The efficiency of the ciphers can be seen from the ratio of throughput and number of gates. The results are shown on Figure 8.

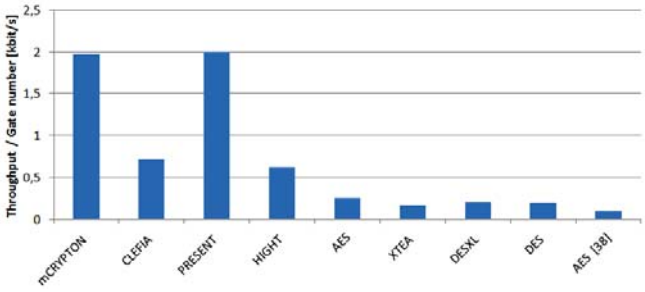


Figure 10. Throughput and Flash size ratio representing the effectiveness of hardware ciphers

Figure 10 shows that the most effective hardware ciphers are mCRYPTON and PRESENT. However, not all ciphers can be found in a hardware based solution provided by various manufacturers. One of these widespread solutions is the AES standard that can be found in various micro-controllers as an additional crypto module, as stated before [38], so it presents a logical solution for the use as a cryptographic function in low computational power embedded systems.

B. *XTEA and AES – Proposed TFTP cipher algorithms*

The block cipher TEA (Tiny Encryption Algorithm) was designed by Wheeler and Needham in 1994 as a short C language program that would run safely on most machines. It achieves high performance by performing all its

operations on 32bit words, using only exclusive-or, addition modulo 2^{32}, multiplication modulo 2^{32} and shift operators. TEA has a simple *Feistel* structure, but uses a large number (i.e. 64) of rounds to achieve the desired level of security [24]. However, taking advantage of its simple key schedule, in 1997 Kelsey, Schneier and Wagner described a related-key attack. To secure TEA against related-key attacks, Needham and Wheeler presented an extended version of TEA in 1997, known as XTEA, which retains the original objectives of simplicity and efficiency [28][34].

Advanced Encryption Standard (AES) is a specification for encryption of electronic data. It has been adopted by the U.S. government and now is used worldwide. The algorithm described by AES is a symmetric-key algorithm. The AES encryption algorithm has five main operations: *AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns*, and *KeyExpansion*. The decryption operations are basically the inversed encryption operations. Besides, the number of rounds of the looping is set to Nr-1 in which Nr is specified according to the AES specification [35].

According to [32], the best cryptanalysis for XTEA is a related-key differential attack. This attack can break 32 out of 64 rounds of XTEA, requiring $2^{20.5}$ chosen plaintexts and a time complexity of $2^{115.15}$, which demonstrates the strength of the mathematical algorithm. All known attacks against AES are computationally infeasible. Related-key attacks can break AES-192 and AES-256 with complexities 2^{176} and $2^{99.5}$, respectively [33]. However, some attack techniques target the physical implementation rather than the algorithm itself. When sampled at high rates and examined in more detail, power waveforms can reveal the key bits [36]. The impact of these attacks is emphasized by the fact that the attacker is in possession of the device. This problem can be overcome by several methods. One method is based on using specifically designed and secures integrated circuits (micro-controllers). The problem using this method is inability to implement the encryption in the existing embedded systems. This can be avoided by using a smart random code injection to mask power analysis based side channel attacks [37].

VI. Implementation and Testing of Secure TFTP

As discussed in Section III, the proposed secure version of TFTP uses a modification of *Digest Access Authentication* to authenticate the incoming requests. In order to implement the authentication in TFTP an experimental TFTP server is required. In order to design an experimental TFTP server, the hardware and software components were chosen accordingly. The measurements were conducted using two platforms: AVR XMEGA (running at 57MHz) and PIC32 (running at 80MHz), respectively [39]. In order to enable the proper functionality in both platforms, the used programming language is ANSI C compiler *MikroC* [39]. The advantage of using *MikroC* is the ability to use the identical C code in both architectures, thus simplifying the implementation of the TFTP. In addition to the used development boards, the Ethernet controller was used to establish the connection to the LAN as well as a *microSD* data card for data storage. To establish the baseline UDP communication required by the TFTP, the existing precompiled libraries within the *MikroC* compiler were used accordingly.

After establishing basic TFTP communication, the security component was implemented as well. In order to examine the complexity of the added security component the additional overhead is calculated as well and the results are shown.

Figure 11 shows the added overhead compared to basic TFTP protocol. The induced ROM memory (FLASH memory) overhead is 12%, whereas the induced overhead in the RAM section is negligible (1.7%).

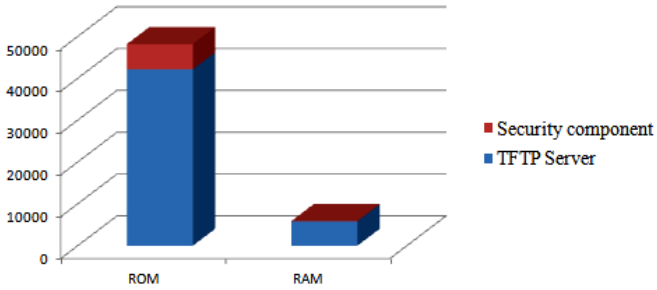


Figure 11. Additional overhead for the TFTP security component

Upon analysis of the authentication mechanism, it was inconvenient to display these data (raw data format). However, from the conducted measurement it can be concluded that the implemented protocol provides the required functionality defined in Section III and on Figure 4.

A. Secure TFTP functionality testing

To test the data encryptions proposed in Section IV the implementation of the XTEA and AES algorithms was necessary in both hardware platforms: XMEGA and PIC32. Due to the fact that XMEGA incorporates hardware AES module the software algorithms for this encryption was not developed. XTEA and AES software algorithm were developed in ANSI C and implemented in hardware platforms.

The measurements were conducted using *WireShark* network analyser [43]. Various file sizes were transferred using STFTP protocol by different encryption scenarios: unencrypted scenario, XTEA encryption and AES encryption. The request used to transfer data is a Read request, initiated from the host computer. The measurement results (displayed on Figure 13) show the throughput and transfer time, versus transferred file sizes whereas Figure 12 shows the summary of the data rates against different ciphers and architectures.

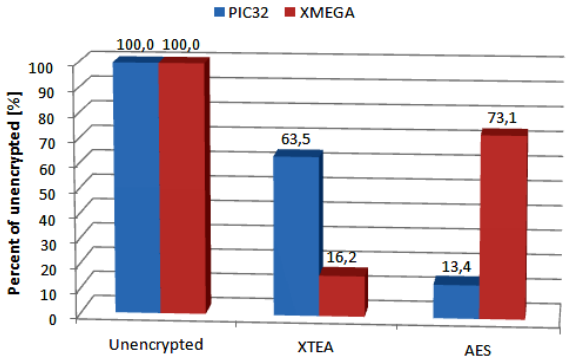
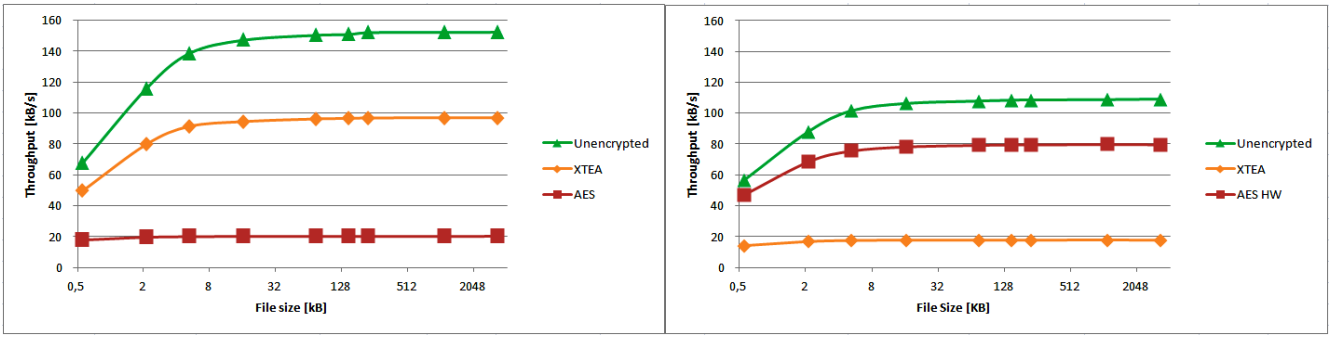


Figure 12. Data rate in comparison for proposed ciphers against unencrypted communication



a) Microchip PIC32 b) Atmel AVR XMEGA

Figure 13. Measured throughput of the Secure TFTP for various cipher

By implementing secure TFTP in PIC32 while using software encryption algorithms the drop in throughput for different encryptions is evident (Figure 12). For simple algorithms (XTEA) the overall drop in performance is 36.5% while more complex algorithms such as AES delivers a drop in performance by 86.5%. The main reason for this amount of drop in performance is the use of non optimised algorithms for the used architecture. By optimising the algorithm, the difference in number of clock cycles for encryption can vary significantly (33 times, maximum throughput 310kB/s!)[40]. This had a profound effect on the overall throughput while being dependant on the chosen architecture. The difference is also visible by comparing two tested architectures and graphing throughput versus file sizes (Figure 13a and 13b, respectively). The unencrypted throughput of these microcontrollers varies primarily due to the different architecture and different clock rates. The saturation of the data rate occurs when the file size is much larger from the overhead of the TFTP packets, making the overhead negligible. For lower file sizes the TFTP overhead results in a drop of data rate due to the small difference between whole packet and the data within.

By analysing the results of the added encryption using the XMEGA hardware platform, it is visible that the drop in performance is not as pronounced. The main reason behind this is the use of hardware based AES module that encrypts/decrypts the data in only 375 cycles [38]. The drop in performance in the case of using a hardware based AES is 26.5%. These results could be compared to the results achieved by the use of WPA2-AES encryption mode. The claim behind this comparison is that the WPA2-AES encryption mode uses AES cipher to establish data confidentiality and it is often implemented in low computational power embedded systems, such as wireless access points. Comparison is seen on Figure 14.

According to [42], in certain applications the drop in performance when using WPA2-AES encryption compared to unencrypted network can be 33.2%. When comparing this to the induced drop in performance for secure TFTP using hardware AES and XMEGA hardware platform, the reduction in throughput of the proposed STFTP is less than the induced drop measured for WPA2-AES [42]. As the maximum throughput of the hardware AES module is calculated to be 2.3 MB/s, the optimization exists [38].

Finally, the functionality of software based XTEA is measured on a lower computational power XMEGA hardware platform, operating at a clock of 57MHz.

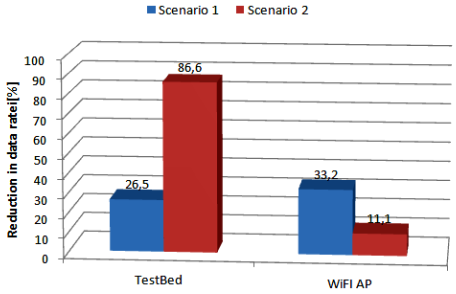


Figure 14. Reduction in data rate for the AES cipher in testbed compared to the Wi-Fi Access Point

The throughput of the software based XTEA encryption is reduced by 87.5% (Figure 13 b). The main reason behind this is the use of optimized software algorithm limited by low computational power of a microcontroller.

The conducted set of measurement demonstrates the functionality of the proposed STFTP protocol, while the actual performance depends on the hardware architecture and processing capabilities. From the measurements it is obvious that the STFTP can easily be implemented even to performance constrained hardware architectures (such as low computational power micro controllers). This claim is supported by the measurements conducted on the low power XMEGA microcontroller. Also, the minimum hardware requirements for the implementation of STFTP are solely related to the application of the STFTP, as certain eMAS require real time support and the transfer of large quantities of data while others are oriented towards passing messages small in size and not constrained by real time requirements.

VII. Conclusion and Future Work

The main advantage of using TFTP in the embedded multi-agent systems (eMAS) is the simplicity of implementation. At the same time, the main disadvantage of the TFTP is the lack of the security. Therefore, this paper proposes an extension to the existing TFTP in a form of added authentication and established confidentiality for use in embedded systems of low computational power – STFTP.

The authentication process proposed in STFTP is a method derived from *Digest Access Authentication* accompanied by SHA-1 hash function, as opposed to the original MD5 algorithm that was proven cryptographically unsecure. By implementing SHA-1 the credibility of this authentication scheme was enhanced. To establish data confidentiality this paper proposes two data ciphers, XTEA and AES; secure for all known attacks. By deriving the encryption key from the authentication process the shuffling of the encryption key is ensured. Another advantage of the

AES cipher is the hardware module availability that simplifies the software implementation.

The experimental results of the implemented STFTP show that by adding data encryption, the overall throughput is reduced. According to measurements done in similar environments (WPA2-AES), the reduction in throughput of the proposed STFTP is less than the induced drop measured for WPA2-AES in certain scenarios.

By implementing secure mechanisms within the original protocol this outdated protocol can be reused in modern embedded systems of multi-agent architecture. The advantage of using this protocol in eMAS is the ability for the software agents to exchange messages using *mail* transfer mode, where the e-mail address is substituted for the agent identification (a unique agent's name). Also, because the STFTP implements the ability to use various data ciphers the choice of data ciphers is left to the software agent by means of choosing most appropriate protocol for the situation at hand. This presents flexibility for the eMAS in form of effectively managing the security and the data rate of the communication. By allowing embedded agents to perform the authentication procedure the need for human intervention is eliminated and the system has the ability of being fully autonomous.

REFERENCES

[1] J. Parab, S.A. Shinde, V.G. Shelake, R.K. Kamat, G.M. Naik, "Practical Aspects of Embedded System Design using Microcontrollers", Springer 2008, XXII, 150 p.

[2] J.P. Jamont, M. Occello. "Design of embedded multiagent systems: discussion about some specificities", Proc.of VII Agent-Oriented Software Engineering Technical Forum, Paris, France, 15. Dec. 2010.

[3] J.P. Jamont, M. Occello "Presentation on Design of embedded multiagent systems", VII Agent-Oriented Software Engineering Technical Forum, Paris, France, 15. Dec. 2010

[4] N. Tziritas, T. Loukopoulos, S. Lalis, P. Lampsas, "Agent placement in wireless embedded systems: Memory space and energy optimizations," Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on , vol., no., pp.1-7, 19-23 April 2010

[5] L. Yun, W. Xun, F. Yulian, „Intelligence technology and information processing", Science Press, Beijing,2003.

[6] M. Xinglu, Q. Yingjie, "Research on Embedded Agent System Architecture," International Conference on Embedded Software and Systems Symposia, 2008. ICESS Symposia '08., pp.142-145, 29-31 July 2008

[7] Planning apart together – multi-agent system, [Online]. Available: http://www.planningaparttogether.nl/?page_id=150, July 2012.

[8] R. Tynan, G.M.P. O'Hare, M.J. O'Grady, C. Muldoon, "Virtual Sensor Networks: An Embedded Agent Approach," Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on, pp.926-932, 10-12 Dec. 2008

[9] K. Sollins, "The TFTP Protocol (Revision 2)", RFC 1350, July 1992.

[10] G. Horvat, D. Šoštarić, Z. Balkić "Cost-effective Ethernet Communication for Low Cost Microcontroller Architecture", International Journal of Electrical and Computer Engineering Systems, Vol 3. No 1, pp.1-8, 2012.

[11] G. Malkin, A. Harkin, "TFTP Option Extension", May 1998.

[12] Shamshirband, S. S., Shirgahi, H., Setayeshi, S., "Designing of Rescue Multi Agent System Based on Soft Computing Techniques," Advances in Electrical and Computer Engineering, vol. 10, no. 1, pp. 79-83, 2010, doi:10.4316/AECE.2010.01014

[13] Microchip Technology Inc., [Online]. Available: www.microchip.com

[14] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, S. Ravi,, "Security as a new dimension in embedded system design," Proc. of. 41st Design Automation Conference, pp.753-760, 7-11 July 2004

[15] Z, Hengwei; W, Wei; G, Qiang; , "Research and Design of Secure Transmission Protocol Applied to Embedded System," Proc. of (ICICTA), 2012, pp.276-279, 12-14 Jan. 2012

[16] D.V. Bernardo, D. Hoang, "Protecting Next Generation High Speed Network Protocol - UDT through Generic Security Service Application Program Interface - GSS-API," Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on , vol., no., pp.266-272, 18-25 July 2010

[17] B. Groza, P.-S. Murvay, I. Silea, T. Ionica,"Cryptographic Authentication on the Communication from an 8051 Based Development Board over UDP," Internet Monitoring and Protection, 2008. ICIMP '08. The Third International Conference on, pp.92-97, June 29 2008-July 5 2008

[18] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[19] Fanbao Liu; , "On the Security of Digest Access Authentication," Proc. of CSE 2011, pp.427-434, 24-26 Aug. 2011

[20] Metzger, P.; Simpson, W., "IP Authentication using Keyed SHA"," RFC 1852, October 1995

[21] S. Manuel, "Classification and generation of disturbance vectors for collision attacks against SHA-1", In Proc. of Des. Codes Cryptography, 2011, pp.247-263.

[22] Z. Balkic; D. Sostaric and G. Horvat, "GeoHash and UUID Identifier for Multi-Agent Systems", Agent and Multi-Agent Systems. Technologies and Applications, Lecture Notes in Computer Science, , Volume 7327/2012, pp.290-298. Springer, June 2012.

[23] Xie, T., Feng, D. "How to find weak input differences for MD5 collision attacks", IACR Cryptology ePrint Archive, Report 2009

[24] David J. Wheeler, Roger M. Needham, "TEA, a Tiny Encryption Algorithm" The Computer Laboratory, Cambridge University, 1994

[25] "Trivial File Transfer Protocol", http://www.pcvr.nl/tcpip/tftp_tri.htm

[26] C. M. Chu-Jenq, "Implementing a Secure Communication Protocol for Embedded Systems", www.embeddedonline.com

[27] Ali E. Taki El_Deen, Noha A. Hikal, "Microcontroller Application in Cryptography Techniques", Canadian Journal on Electrical and Electronics Engineering Vol. 1, No. 4, June 2010

[28] M. Pavlin, "Encription using low cost microcontrollers", Proc. of MIDEM - Society for Microelectronics, Electronic Components and Materials, cop. 2006, pp. 189-194

[29] G. Meiser, T. Eisenbarth, K. Lemke-Rust and C. Paar, "Software implementation of eSTREAM profile I ciphers on embedded 8-bit AVR microcontrollers", Workshop Record State of the Art of Stream Ciphers (SASC 07), 2007.

[30] S. Rinne, T. Eisenbarth, C. Paar, "Performance Analysis of Contemporary Light-Weight Block Ciphers on 8-bit Microcontrollers", Horst Gortz Institute for IT Security Ruhr University Bochum Germany, 2007.

[31] Eisenbarth, T.; Kumar, S.; , "A Survey of Lightweight-Cryptography Implementations," Design & Test of Computers, IEEE , vol.24, no.6, pp.522-533, Nov.-Dec. 2007

[32] Lu Jiqiang, "Related-key rectangle attack on 36 rounds of the XTEA block cipher", International Journal of Information Security, Vol: 8 (1): pp. 1–11, 2009

[33] A. Bogdanov, D. Khovratovich, C. Rechberger, "Biclique Cryptanalysis of the Full AES". IACR Cryptology ePrint Archive 2011: 449 (2011)

[34] Jiqiang Lu, „Cryptanalysis of Block Ciphers"; Technical Report RHUL–MA–2008–19, Department of Mathematics Royal Holloway, University of London, 2008

[35] C. C Lu; S. Y. Tseng;, "Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter," Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on, pp. 277- 285, 2002

[36] Knežević M., Rožić V., Verbauwhede I. , "Design Methods for Embedded Security", TELFOR Journal, Vol.1, No. 2, 2009,

[37] Ambrose, J.A.; Ragel, R.G.; Parameswaran, S.; , "A smart random code injection to mask power analysis based side channel attacks" Proc. of International Conference on Hardware/Software Codesign and System Synthesis 2007 5th, pp.51-56

[38] Atmel AVR XMEGA A Manual, [Online]: www.atmel.com

[39] El. Equipment Manufacturer: MikroElektronika, www.mikroe.com

[40] Schramm, K.; Paar, C.; , "IT security project: implementation of the Advanced Encryption Standard (AES) on a smart card," Proceedings International Conference on Information Technology: Coding and Computing, 2004., vol.1, pp. 176- 180 Vol.1, 5-7 April 2004

[41] Narayan, S.; Kolahi, S.S.; Sunarto, Y.; Nguyen, D.D.T.; Mani, P.; , "The Influence of Wireless 802.11g LAN Encryption Methods on Throughput and Round Trip Time for Various Windows Operating Systems," Proc. of CNSR 2008. 6th Annual, pp.171-175, 5-8 May 2008

[42] A. Murabito, "A comparison of efficiency, throughput, and energy requirements of wireless access points", University of New Hampshire, InterOperability Laboratory, March 2009.

[43] WireShark - Network Protocol Analyzer: [Online]. Available: www.wireshark.com