International Journal of Smart Home
Vol.8, No.2 (2014), pp.77-90
http://dx.doi.org/10.14257/ijsh.2014.8.2.09

Proposal for Improving Connectivity and adding Authentication and Security to KNXNet/IP Protocol

J. A. Nazabal, F. Falcone, C. Fernández-Valdivielso and I. R. Matías

Electrical and Electronic Engineering Department, Public University of Navarra, Campus de Arrosadia 31006, Pamplona, Spain
juanantonio.nazabal@unavarra.es

Abstract

KNXnet/IP protocol defines mechanisms for using KNX Home and Building Automation technology on top of the IP protocol using client/server architecture. Due the nature of KNXnet/IP protocol it presents some connectivity and security problems inherent to the protocol definition. In this work we propose some mechanisms for trying to fix these problems and finally we implement and test them using real KNX devices.

Keywords: *KNX, Building Automation, KNXNet/IP, Interface*

1. Introduction

KNX is a Home and building automation standard for a typical application area of environmental control of building lighting/shading services, as well as heating, ventilation and air conditioning (HVAC). KNX is an approved standard in Europe (CENELEC EN 50090 [1] and CEN EN 13321-1 [2]), USA (ANSI/ASHRAE 135 [3]) and China (GB/Z 20965 [4]) as well as an international standard (ISO/IEC 14543-3 [5]), confirming its relevance in the home and building automation sector. Nowadays, other building service types like safety (*e.g.*, fire alarm) and security critical (*e.g.*, access control or intrusion alarm) systems have been only implemented by dedicated standalone systems. The information flow exchanged in some of these environments is critical, as it may lead to execution of commands into actuators and/or access to private and secret information.

The need to protect critical data exchanged between some devices belonging to home and building environments has led to the need to introduce secure transmission inside communication systems. For securing a data transmission, authentication and security mechanisms are needed. Security protects all the data during transmission while authentication guarantees that the devices are permitted to access the information. KNX devices are provided by a large number of manufacturers, offering more functionality within the home automation system and the standardization guarantees that the different manufacturers' products may be interconnected together. This means there are no limitations to a single manufacturer and if a manufacturer ceases to trade or offer a particular product, the same or similar products are available from other manufacturers. The KNX Device Network results from the formal merger of the 3 leading systems for Home and Building Automation (EIB, EHS, BatiBus) into the specification of the new KNX Association and has more than 20 years of experience in the market, so it is a robust and well proven technology. It is also royalty-free and entirely independent of any specific microprocessor platform or even architecture and may be flexibly adapted to present an optimal solution for each application domain and installation. KNX is equipped with a powerful toolkit for network management

ISSN: 1975-4094 IJSH

that specifies a set of mechanisms to discover, set or retrieve configuration data actively via the network.

There are several different physical transmission mediums that have been foreseen in the KNX standard: PLC 110 powerline [6], RF Radio Frequency [7], TP 1 twisted pair [8] and IP. Usually the most commonly used media is twisted pair and the information flows along the KNX bus installed in the building. If somebody wants to access that information he/she must be physically there and will need to have access to the network wires, which in nearly all cases is impossible as the wires are inside a building or buried underground. Something similar happens with powerline and wireless mediums. They have a limited range and if a possible attacker wants to make malicious actions into the KNX installation, he/she will need to be into the building for having access to the KNX bus. This is indeed a problem but in most of the cases it has not really big importance and no extra actions are needed. However, when using IP medium the scenario changes. In this case, the information flows all over Internet and can potentially be accessed from anywhere. Having this idea in mind it is easy to see that some extra security mechanisms are needed.

KNX security problem has been already studied before and some different solutions have been proposed [9-11]. In this work we present a specific solution for KNXnet/IP security problem in particular that can be easily extended to KNX telegrams as well. We also propose a solution to KNXnet/IP connectivity problems that appear when using dynamic IP addressing.

2. KNXNet/IP Protocol

KNXnet/IP specification [12-14] defines the integration of KNX protocol implementation on top of the Internet Protocol (IP). The KNXnet/IP protocol is used to tunnel or route KNX data over the widely spread Internet, enabling remote access and maintenance across long distances, as well as usage as high speed backbone for KNX networks. It is possible to use raw IP, TCP and UDP packets for transporting KNXNet/IP frames but from an application point of view it is much easier to simply exclude raw IP and restrict the specification to TCP and UDP.

KNXNet/IP has client/server architecture and Figure 1 shows the procedure for establishing a link. All KNXnet/IP frames shall have a common header, consisting of the protocol version, payload length information, and the KNXnet/IP service type identifier. Tunnelling consists in sending a single KNX frame into an IP frame and waiting until the response arrives or a timeout is reached and the implementation of Tunnelling on KNX Data Link Layer is mandatory. KNX frames shall always be sent within a TUNNELLING_REQUEST frame and the KNX frame shall be in cEMI format. The cEMI format shall be supported by all KNXnet/IP devices.

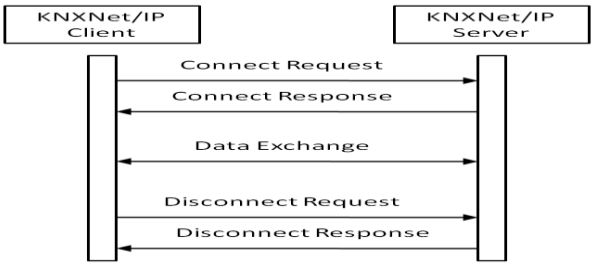

Figure 1. KNXNet/IP Client/Server Link Establishment

3. Connectivity

Let's explain the connection procedure in KNXNet/IP protocol. First KNXNet/IP client sends a CONNECT_REQUEST frame to server's well known IP address. This frame contains client's IP address that will be used by the server for communicating with it the rest of the connection time. Next server responses sending to the client a CONNECT_RESPONSE frame that contains the status of the request and the Communication Channel identifier associated to that connection and used for identification as long as the link is up. Both client and server must store and use this Communication Channel because it will be used the rest of the connection, including it into exchanged frames for authentication purpose.

Analyzing this connection procedure it can be appreciated two important facts that can lead to connection problems. The first one is that server's IP address must be well known by the client and it also must remain without any change all the connection time. The second one is that server acquires client's IP address only once, after receiving the corresponding CONNECT_REQUEST frame, and uses it the rest of the connection time. If client's IP address changes during the connection, server will send the rest of the frames to the wrong old IP address, leading to a serious connection problem. Nowadays there are several different technologies for accessing Internet. In many of them, like DSL, GPRS, *etc.*, the Internet Service Provider (ISP) assigns dynamic IP addresses to connected devices. These dynamic IP addresses change with the pass of the time and the amount of time that they remain without any change will depend on the technology and the ISP used.

As we said before, the client must know all the time server's IP address and if it has a dynamic IP address this represent a connection problem. One possible solution is for the server to use a Dynamic Domain Name System (DDNS) service. Dynamic DNS is simply a DNS server that allows you to very rapidly change the mapping between a network name and an IP address. This is used to provide a persistent network name for a resource that may change location on the network. Normally this is useful for devices using an ISP which only provides dynamic IPs.

Dynamic DNS providers offer a software client program that is executed into the client device and periodically checks client's public IP address. When this program detects an IP address change, it warns DDNS provider's system and it updates new IP address in the associated network name. Depending on the provider, hostname is registered within a domain owned by this provider, or within the customer's own domain name. Often they use an HTTP service request since even restrictive environments usually allow HTTP service. The provider might use RFC 2136 [15] to update the DNS servers. If we are using dynamic IP addresses and client and / or server's IP address changes without the other's knowledge, the rest of the data packets will be sent to the old wrong IP address representing an important connection problem. We must provide a mechanism for determining when the IP addresses has change and communicating it to the other part. For that purpose, both client and server must check their IP address periodically asking for their current addresses to an external server that provides this service. There are many of well know web pages that support this functionality.

When the client or the server detects that the IP address has changed, we propose the use of the frame shown in Figure 2 for informing the other part the new IP address to use in the future. For future references, we will call this frame ADDRESS_CHANGE_NOTIFICATION and the associated service type identifier 020bh. The first 6 bytes represent the typical KNXNe/IP header and next byte represents the communication channel identifier.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 020Bh
4	0Bh	
5	00h	Payload Length 7 Octets
6	07h	
7	01h	Communication Channel Identifier, e. g. 01h

Figure 2. ADDRESS_CHANGE_NOTIFICATION Frame

When an ADDRESS_CHANGE_NOTIFICATION frame is received, the receiving part must update the others new IP address and send back an acknowledge frame. For future references, we will call this frame ADDRESS_CHANGE_NOTIFICATION_ACK and the associated service type identifier 020Ch. The proposed frame structure is shown in Figure 3 and the last byte represents the status code of the IP address updating procedure. If no error has been encountered, the value of this code is 00h. When an ADDRESS_CHANGE_NOTIFICATION frame is received and depending if TCP or UDP protocol is used different actions must be taken.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 020Ch
4	0Ch	
5	00h	Payload Length 8 Octets
6	08h	
7	01h	Communication Channel Identifier, e. g. 01h
8	00h	Status Code (NO_ERROR)

Figure 3. ADDRESS_CHANGE_NOTIFICATION_ACK Frame

TCP protocol is a reliable, complex and heavy protocol that presents a connection establishment procedure. To establish a connection, each device must exchange some signaling packets and if IP address changes, TCP socket changes as well and TCP session connection breaks. In this case, the device whose IP address has changed must send a DISCONNET_REQUEST and close the link. UDP protocol is a non reliable, simple and lightweight protocol that has no connection establishment procedure. In the case of using this protocol, the old UDP socket can be replaced with the newer one with the updated IP address, without any further problem.

Also and due the nature of the data sending procedure in UDP protocol, we propose that when KNXNet/IP server receives a data request frame from the client, it should retrieve the IP address from the request UDP packet that contains the request frame and send the response

to the same IP address. Doing so, the connection problems when the client has a dynamic IP address and requests data will be fixed.

4. Security

For protecting the information exchanged between KNXNet/IP client and server the data must be encrypted. Doing so, a possible attacker maybe have access to encrypted data but cannot retrieve the original information from it or inject malicious code into the IP packets for unwanted activation of KNX actuators [9-11].

In this work we propose to use a Symmetric-key algorithm for encrypting everything but the KNXNet/IP header. Reading the header we can access protocol version field and determine if the frame is using the old protocol or the new proposed one and needs decryption. Symmetric-key algorithms use the same cryptographic key for both encryption and decryption. The key represent a shared secret between two or more parties that can be used to protect private information in an unsecure medium. Symmetric-key encryption can use either block ciphers or stream ciphers. Block ciphers take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size and stream ciphers encrypt the bits one at a time. There are many different Symmetric-key block ciphers but one of the most widely used is AES.

In 1997, National Institute of Standards and Technology (NIST) initiated a process to select a symmetric-key encryption algorithm to be used to protect sensitive information and develop an Advanced Encryption Standard (AES). The winner algorithm selected was Rijndael (now known as just AES) among four other finalists: MARS, RC6, Serpent and Twofish [16]. There is a lot of literature about these algorithms, with different benchmarks and analysis [16, 17] and it is complicated resuming all in only a few lines but basically the most secure but also slowest is Serpent, Rijndael is the fastest and Twofish represents a middle road.

There are also several different Symmetric-key stream ciphers and the most widely used are represented in the eSTREAM contest for a new stream cipher standard [18]. All of the candidates are divided in two different categories. One category, called Profile 1, contains four of the proposals algorithms suited to achieve fast encryption in software: HC-128, Rabbit, Salsa20/12 and SOSEMANUK. The other one, called Profile 2, contains the algorithms that offer an efficient hardware implementation: Grain v1, MICKEY 2.0 and Trivium.

There are many research documents analyzing the performance of these algorithms [18, 20] and the expectations from an efficient cryptographic algorithm will differ depending on the specific application. It is very difficult to expect that a single implementation will satisfy all requirements. In this work we have chose Salsa20 because it has, in general, better performance that AES. It also has very good software performance [21], medium/high speed and medium area in hardware Field Programmable Gate Arrays (FPGA) implementations [22] and is a promising alternative for the AES cipher in 8-bit AVR Microcontrollers [23].

Encrypting KNXNet/IP payload may avoid an unauthorized access to the information that transports but cannot prevent for unknown clients from establishing connections with the server and ask for information, even if they cannot decrypt it. The solution to that problem is to add an authentication mechanism.

5. Authentication

For providing KNXNet/IP protocol an authentication procedure we are going to employ Public-key cryptography. These kinds of algorithms require two different keys,

one of which is secret and one of which is public, and the key pair is mathematically linked. The public key is used for encrypting the plaintext and the associated private key decrypts the ciphertext. The public key may be published using an unsecure medium without compromising security, while the private key must remain secret and not be revealed to anyone not authorized to access the information. Public-key cryptography uses asymmetric key algorithms based on mathematical relationships problems in which it is computationally easy to generate the public and private keys, to decrypt the message using the private key and to encrypt the message using the public key, but it is extremely difficult to derive the private key or the encrypted message based only on their knowledge of the public key. There are many different algorithms available and Table 1 shows the most commonly used ones.

Table 1. Public-key Encryption Algorithms

Algorithm Names	Theory
RSA	RSA problem
ECC	Elliptic Curves
Diffie–Hellman	Discrete logarithm
NTRU	Shortest vector

There are many research documents that analyze the performance [24, 25] and shows practical comparisons [26] between these algorithms. In this work we have chose NTRU because it has better performance that the rest of competitors [27, 28] and even other symmetric key encryption algorithms [29]. Now that the encryption algorithm has been selected, it is time for explaining the proposed authentication connection procedure.

First client must create a NTRU key pair and then send to the server a modified proposed CONNECT_REQUEST frame, shown in Figure 4. The first byte after the header represents the host protocol used, the next one indicates the connection type and the last one represents the KNX layer. Finally, the rest of the bytes represent the client's NTRU public key.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 0205h
4	05h	
5	...	Payload Length
6	…	
7	01h	Host Protocol Code, e.g. 01h, for UDP over IPv4
8	04h	Connection Type Code, e.g. 04h, TUNNEL_CONNECTION
9	02h	KNX Layer, e.g. TUNNEL_LINKLAYER
10	…	Client's NTRU Public Key
	…	
N	…	

Figure 4. CONNECT_REQUEST Frame

After the server receives a CONNECT_REQUEST frame, it must retrieve client's NTRU Public Key, generate a binary random sequence and encrypt it with that key. Then it must send to the client an AUTHETICATION_REQUEST frame with the structure shown in Figure 5. The first byte after the header represents the temporal Communication Channel assigned to the connection and the next one indicates if any kind of error has occurred. Finally, the rest of the bytes represent the NTRU encrypted binary random sequence.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 020Dh
4	0Dh	
5	…	Payload Length
6	…	
7	01h	Communication Channel Identifier, e. g. 01h
8	00h	Status Code (NO_ERROR)
9		Random Sequence encrypted with Client's Public Key
	…	
N	…	

Figure 5. AUTHETICATION_REQUEST Frame

When the client receives an AUTHETICATION_REQUEST frame, it must retrieve the NTRU encrypted random sequence, decrypt it with the client's secret private key and encrypt it with the same Salsa20 key that both client and server pre-share and

already know. Then it must send to the client an AUTHETICATION_RESPONSE frame with the structure shown in Figure 6.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 020Eh
4	0Eh	
5	…	Payload Length
6	…	
7	01h	Communication Channel Identifier, e. g. 01h
8	00h	Status Code (NO_ERROR)
9	…	Random Sequence encrypted with Salsa20 Key
	…	
N	…	

Figure 6. AUTHETICATION_RESPONSE Frame

The first byte after the header represents the temporal Communication Channel, the next one indicates the presence of errors and the rest of the bytes represent the Salsa20 encrypted random sequence. When server receives an AUTHETICATION_RESPONSE Frame, it must retrieve the Salsa20 encrypted random sequence, decrypt it and compare it with the original random sequence generated. If the sequences are similar that indicates that both client and server know and has the same Salsa20 key and the authentication is complete. After the server receives an AUTHETICATION_RESPONSE Frame, it sends a CONNECT_RESPONSE frame to the client indicating if the connection is authorized or not. The proposed frame structure is shown in Figure 7. The first byte after the header represents the Communication Channel, the next one indicates if the connection is authorized or not and the last byte represent the connection type.

1	06h	Header Size
2	11h	Protocol Version
3	02h	Service Type Identifier 0206h
4	06h	
5	00h	Payload Length 10 Octets
6	0Ah	
7	01h	Communication Channel Identifier, e. g. 01h
8	00h	Status Code (AUTHORIZED)
9	01h	Host Protocol Code, e.g. 01h, for UDP over IPv4
10	0x04	Connection Type Code, e.g. 04h, TUNNEL_CONNECTION

Figure 7. CONNECT_RESPONSE Frame

6. Implementation

For analyzing the feasibility of the proposed protocol a real implementation has been developed, shown in Figure 8. First a minimum KNX installation has been used that consists on a 4 analog input channels device, model 2214 REG A, by Jung and an 8 channel switch actuator device, model SA/S 8.10.1, by Siemens. For accessing the KNX bus the KNX installation has also a USB / KNX interface, model 2130 USB REG by Jung. This interface is connected via an USB cable to an old Pentium 4 Personal computer (PC) with 1 Gbyte of RAM and windows XP SP3 installed. A Java application that implements the server part of the proposed new KNXNet/IP version protocol has also been developed.

That application runs into the PC and access KNX data via the USB / KNX interface and implements KNX standard Application Note 037/02 [30]. As the USB / KNX interface appears in Windows XP as a Human Interface Device (HID), the application uses Javahidapi [31] library for interacting with it. It also uses Bouncy Castle Crypto [32] Java library for implementing Salsa20 encryption algorithm and NTRU Java library provided by "The NTRU Project" webpage [33].

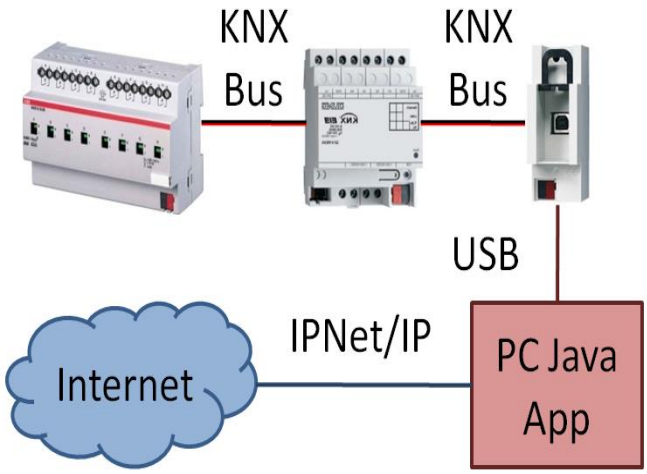

Figure 8. Developed Installation for Protocol Testing

For giving the server a static and well known network name "no-ip" site [34] that offers free dynamic DNS service has been used. First the service and the network name must be configured and then a small Dynamic Update Client (DUC) program must be correctly installed. Both the client and the server must have a mechanism for alerting the other part when their IP address has changed. As it was said before, the need to check their address periodically in an external web server dedicated to that purpose is imperative.

There are many of them available and the mechanism for obtaining the IP address is basically the same: first make a GET HTTP request to the service and then receive the IP address as a character string. In this work "dyndns" site that provides automation service [35] has been used with good results. It is also possible to easily implement one of these services programming a PHP script and putting it into a web server. This last option has been tried also and custom PHP script put running into "FreeHostingEU" web hosting site [36] with also good results. Because it was developed for testing purposes only, the free plan was selected that allows 4000 MB of monthly data transfer

and the service has limited resources. For testing this installation a java application that implements the client part of the proposed new KNXNet/IP version protocol has also been developed.

7. Results and Conclusions

For testing the correct structure and the content of the KNXNet/IP frames exchanged between the client and the server, the Wireshark [36] network protocol analyzer tool has been used. Maybe the most complex and interesting data exchange is given at the link establishment procedure and figures from Figures 9 to 12 shows a Wireshark capture of the frames involved in it. Due the big data amount content in the captured CONNECT_REQUEST and AUTHENTICATION_REQUEST frames, the associated figures shows only a portion of the original data but enough for giving a hint of their structure.

As can be seen it Figure 9, CONNECT_REQUEST frame data is totally unencrypted but does not contain any critical information, just only the client's public key stored as a byte array of the encoded "EncryptionPublicKey" [33] Java class associated to it. This byte array has a 608 byte length but is not a problem because a UDP packet can carry as much data as 65536 bytes. Figure 10 shows AUTHENTICATION_REQUEST frame structure and how all the data is encrypted but the header using the previously received client's public key. Doing so, nobody but the client can have access to the temporal communication channel identifier associated to that connection.

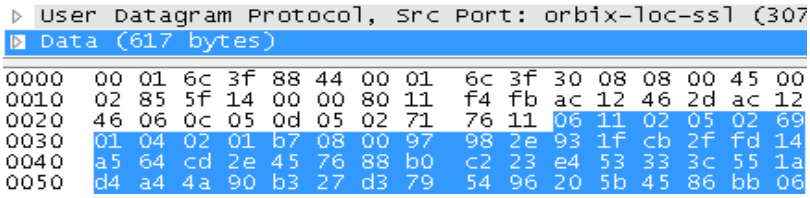

Figure 9. Captured CONNECT_REQUEST frame

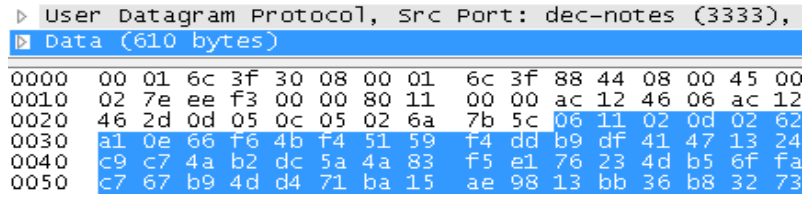

Figure 10. Captured AUTHENTICATION_REQUEST Frame

In Figure 11 it can be appreciated how every data but the header is also encrypted but this time using the Salsa20 Key. This key has a length of 32 bytes only and the size of the encrypted data is small if comparing with using NTRU. Finally Figure 12 shows DISCONNECT_REQUEST frame structure, this time encrypted with the Salsa20 Key that will be used for the rest of the connection.

As can be appreciated, the payload information is protected. If a possible attacker knows the frame structures and tries to access, for example, the Communication Channel identifier, the value obtained will be different every time and erroneous. As conclusion it can be said that the new proposed KNXNet/IP protocol works well, at

least in the testing installation that has been prepared in this work for that purpose. When establishing a link between the server and a KNXNet/IP client implemented in an old Pentium 4 PC, sharing the same LAN, the time lapse since the first CONNECT_REQUEST frame is sent until the last CONNECT_RESPONSE frame is received is only of 115 milliseconds.

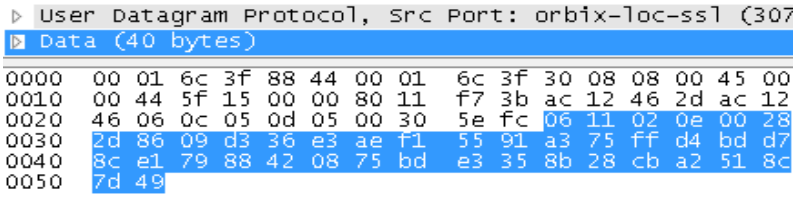

Figure 11. Captured AUTHETICATION_RESPONSE frame

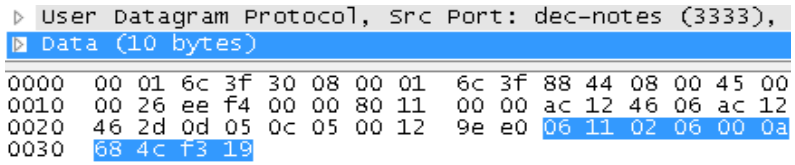

Figure 12. Captured CONNECT_RESPONSE Frame

Nowadays NTRU and Salsa20 encryption algorithms are considerate secure and there is no effective attack known that can break them so it is considered that the encryption used in this proposal is secure enough. Maybe the main problem, derived from the protocol implementation, is that both the client and the server must share the same Salsa20 key and for that reason they need to previously exchange it via a secure channel. Avoiding this in authentication procedures is complicated because both the client and the server parts must share some common secret knowledge for determining the authentication.

If more security is needed, the random sequence exchanged in the authentication procedure can be used as a Communication Channel Key for encrypting the data the rest of the connection time and use the other Salsa key as an authentication key. Because a different random Communication Channel Key is used for each different client and connection time, no one else can access the encrypted information even if knowing the authentication key.

The weakest point in the protocol authentication is the link establishment procedure because is when the Authentication Key is used and can be retrieved in the encryption is broken. The rest of the connection time, if an attacker breaks the encryption it will merely have access to a Key valid only for that client and that link. The price to pay for having more security is that the authentication mechanism is more complex and more resources are needed for using and storing different Salsa20 keys and the associated encryption structures per active link. It is complicated integrating this authentication mechanism in a compatible way into KNX telegrams but it can be feasible using the same encryption system for protecting KNX telegrams payload.

Depending on the modulation technique, access type or collision control of the medium used, the KNX telegram structure can have some extra preamble or envelope sequence but the standard structure is shown in Figure 13.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>…</td><td>N-1</td><td>N ≤ 22</td></tr>
<tr><td>Control Field</td><td colspan="2">Source Address</td><td colspan="2">Destination Address</td><td>Address Type, NPCI, Length</td><td>TPCI</td><td>APCI</td><td>Data / APCI</td><td colspan="2">Data</td><td>Frame Check</td></tr>
</table>

Figure 13. KNX LPDU standard frame structure [38]

A simple but effective security mechanism could consist in encrypting everything but the control field using a common Salsa20 key shared by all the KNX installation devices. For compatibility, a bit within the control field could be used for indicating that the telegram is encrypted. For example, the bit number 4 is always set to "1" and could be set to "0" for indicating that the telegram is encrypted.

For future works it would be a good idea testing the proposed KNXNet/IP protocol using different hardware implementations. A good candidate for accessing the KNX bus could be the bus coupler unit TP-UART 2 [39] and for implementing the protocol, maybe an 8 bit or an ARM microprocessor.

Acknowledgements

This work was supported by the Spanish Economy and Competitivity Ministry AIB2010NZ-00328.include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence).

References

[1] Cenelec EN 50090, Home and Building Electronic System (HBES), **(2005)**.
[2] Cenelec EN 13321-1, Open Data Communication in Building Automation, Controls and Building Management, HBES, **(2006)**.
[3] ANSI/ASHRAE 135, BACnet-A Data Communication Protocol for Building Automation and Control Networks, **(2008)**.
[4] GB/Z 20965, Control network HBES technology specification—Home and building control system, **(2007)**.
[5] ISO/IEC 1454-3, Home Electronic System (HSE) Architecture, **(2006)**.
[6] KNX Association, KNX Standard, System specifications: Communication media Powerline, Chapter 3/2/3, **(2008)**.
[7] KNX Association, KNX Standard, System specifications: Communication media Radio Frequency, Chapter 3/2/5, **(2008)**.
[8] KNX Association, KNX Standard, System specifications: Communication media Twisted pair 1, Chapter 3/2/2, **(2008)**.
[9] S. Cavalieri and G. Cutuli, "Implementing encryption and authentication in KNX using Diffie-Hellman and AES algorithms, 2009. IECON '09. 35th Annual Conference of IEEE, Porto, Portugal, **(2009)**.
[10] D. Lechner, W. Granzer and W. Kastner, "Security for KNXnet/IP, KNX Scientific Conference, Sint-Katelijne-Waver, Belgium, **(2008)**.
[11] S. Cavalieri and G. Cutuli, "Introducing Security and Authentication in KNX", KNX Scientific Conference, Sint-Katelijne-Waver, Belgium, **(2008)**.
[12] KNX Association, KNX Standard, KNXNet/IP: Overview, Chapter 3/8/1, **(2009)**.
[13] KNX Association, KNX Standard, KNXNet/IP: Core, Chapter 3/8/2, **(2009)**.
[14] KNX Association, KNX Standard, KNXNet/IP: Tunneling, Chapter 3/8/4, **(2009)**.
[15] RFC 2136, Dynamic Updates in the Domain Name System (DNS UPDATE), **(1997)**.

[16] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti and E. Roback, "Report on the development of the Advanced Encryption Standard (AES)", National Institute of Standards and Technology, vol. 106, no. 3, pp. 511.
[17] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, T. Kohno and M. Stay, "The Twofish Team's Final Comments on AES Selection", available at http://csrc.nist.gov/archive/aes/round2/comments/20000515-bschneier.pdf, last accessed: 25 January 2013, **(2000)** May 15.
[18] The eSTREAM Portfolio in 2012, available at http://www.ecrypt.eu.org/documents/D.SYM.10-v1.pdf, last accessed: 25 January 2013, **(2012)** January.
[19] P. Mukherjee, "An Overview of eSTREAM Ciphers", available at http://cs.au.dk/~pratyay/eSTREAM.pdf, last accessed: **(2013)** January 25.
[20] T. Good and M. Benaissa, "Hardware Results for Selected Stream Cipher Candidates", The SASC 2007 Workshop Record, Ruhr University Bochum, Germany, **(2007)**.
[21] D. J. Bernstein, Salsa20/8 and Salsa20/12, Available at http://cr.yp.to/snuffle/812.pdf, last accessed: **(2013)** January 25.
[22] M. Rogawski, Hardware evaluation of eSTREAM Candidates: Grain, Lex, Mickey128, Salsa20 and Trivium, The SASC 2007 Workshop Record, Ruhr University Bochum, Germany, **(2007)**.
[23] G. Meiser, T. Eisenbarth, K. Lemke-Rust, C. Paar and H. Görtz, "Efficient implementation of eSTREAM ciphers on 8-bit AVR microcontrollers, IEEE Third International Symposium on Industrial Embedded Systems - SIES 2008, Montpellier / La Grande Motte, France, **(2008)**.
[24] C. Narasimham and J. Pradhan, "Performance Analysis of Public key Cryptographic Systems RSA and NTRU, IJCSNS International Journal of Computer Science and Network Security, vol. 7, no. 8, **(2007)**.
[25] C. Narasimham and J. Pradhan, "Evaluation of Performance Characteristics of Cryptosystem Using Text Files", Journal of Theoretical and Applied Information Technology, vol. 4, no. 1, **(2008)**.
[26] C P. Karu and J. Loikkanen, Practical comparison of fast public-key cryptosystems, Seminar on Network Security, Telecommunications Software and Multimedia Laboratory, Kelsinki University of Technology, Kelsinki, Finland, **(2000)**.
[27] J. Hermans, F. Vercauteren and B. Preneel, "Speed records for NTRU", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5985 LNCS, pp. 73-88.
[28] R. D'Souza, "The NTRU Cryptosystem: Implementation and Comparative Analysis, Available at http://teal.gmu.edu/courses/ECE543/project/reports_2001/dsouza.pdf , last accessed: **(2013)** January 25.
[29] Y. Mote, P. Nehete and S. Gaikwad, "Superior Security Data Encryption Algorithm (NTRU)", International Journal of Engineering Sciences, vol. 6, **(2012)**.
[30] "KNX on USB Protocol Specification & KNX USB Interface Device Requirements", Application Note 037/02 Rev., vol. 4, **(2003)**.
[31] Javahidapi Library, http://code.google.com/p/javahidapi/, last accessed: **(2013)** January 25.
[32] Bouncy Castle Crypto Library, http://www.bouncycastle.org/, last accessed: **(2013)** January 25.
[33] The NTRU Library, http://tbuktu.github.com/ntru/, last accessed: **(2013)** January 25.
[34] "no-ip" free Dynamic DNS, http://www.noip.com/, last accessed: **(2013)** January 25.
[35] "dyndns" free Dynamic DNS automation service, http://checkip.dyndns.org/, last accessed: **(2013)** January 25.
[36] "FreeHostingEU" Free Web Hosting, http://www.freehostingeu.com/, last accessed: **(2013)** January 25.
[37] Wireshark, http://www.wireshark.org/, last accessed: **(2013)** January 25.
[38] KNX Association, KNX Standard, Architecture, Chapter 3/1/1, **(2009)**.
[39] KNX EIB TP-UART 2-IC technical data, www.hqs.sbt.siemens.com/Lowvoltage/gamma_product_data/gamma-b2b/TPUART2_technical-data.pdf, last accessed: **(2013)** January 25.

