UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

VULNERABILITY DISCOVERY IN POWER LINE COMMUNICATIONS

Fernando Baptista Leal Alves

Dissertação orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves e co-orientado pelo Prof. Doutor Alysson Neves Bessani

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

2015

Acknowledgements

In first place I thank my advisors for accepting me for this thesis. Both taught me a lot. Prof. Nuno is excellent at raising the bar, and prof. Alysson - who I work with for two and a half years - always provided excellent feedback to improve my work. Thank you for this opportunity. Next to my advisors, I thank Francisco Aragão and Manuel José Mendonça for the time they spent helping me with the thesis. Francisco provided me with an excellent jump start on hacking, and Manuel introduced me to the world of Linux device drivers. Without their help this work would have never reached this point.

I thank my parents for their support, and a very special thanks to my brother Daniel who listened to me and read a bit of this thesis without understanding a thing, besides being my daily comrade. Next, I thank all my friends for support and company, specially for César Santos, João Caetano, João Vieira, José Simões and Soraia Meneses Alarcão, for all the lunch and snack time conversations. The time spent with them allowed me to recharge batteries and keep on working. They also read my thesis, which tested their patience. For all my friends, my thanks.

Last but not (at all!) least, I thank my girlfriend and love Soraia Meneses Alarcão, not only for actually loving me (who knows how), but for being my friend, my companion, and my special little girl. You were always present and lighten my mood with your care. For making my days brighter, thank you.

Funding

This work was partially supported by the EC through project FP7 SEGRID (607109), by national funds of Fundação para a Ciência e a Tecnologia (FCT) through project UID/-CEC/00408/2013 (LaSIGE).

Para a Soraia.

Resumo

A comunicação em powerline é uma forma de transmissão de dados através da rede eléctrica. Esta é usada para a passagem de corrente e transmissão de dados, utilizando assim a mesma infra-estrutura para duas funcionalidades, ambas essenciais nos dias de hoje. Existem ligações de banda estreita e larga em comunicação por powerline, dependendo da frequência da onda eléctrica. Devido à baixa frequência e à distância entre pontos, em redes industriais existem apenas ligações de banda estreita, providenciando velocidades até 500kB/s. Em redes caseiras a frequência da onda eléctrica é alta, permitindo comunicação em powerline com velocidades de banda larga (várias centenas de MB/s).

Esta forma de comunicação tem dois principais usos: redes domésticas e redes industriais. Em redes domésticas, a comunicação em powerline é utilizada para estender uma ligação Internet já existente, através dos fios eléctricos de uma casa. O objectivo é obter conectividade em qualquer ponto de uma casa sem recorrer a repetidores, redes sem fios, ou à instalação de novos cabos. Para este efeito são utilizados adaptadores de powerline, que são ligados às tomadas eléctricas. O router que serve de ligação à internet é conectado através de um cabo Ethernet a um destes adaptadores. Note-se que este é um router comum, obtido através de uma instalação de internet típica. Ao estar ligado ao adaptador de powerline, o router transmite dados através da rede eléctrica. Outros adaptadores de powerline podem então ser ligados a outras tomadas da mesma casa, e a estes podem ser ligados computadores, impressoras, ou quaisquer outros equipamentos que se deseje que tenham uma ligação à rede, obtendo sinal tal como a partir de uma ligação directa ao router. Assim, a partir de qualquer tomada é possível obter ligação à Internet para qualquer computador ou dispositivo caseiro.

As redes industriais são compostas por vários elementos que formam a distribuição de serviços num país, como é o caso da rede eléctrica, gás e água, entre outras utilizações. Neste ambiente, a comunicação em powerline permite que a rede eléctrica já existente seja utilizada para a passagem de informação, como leituras de contadores ou o envio de alarmes. Os principais utilizadores da comunicação em powerline são as companhias eléctricas, que com esta forma de comunicação podem usar a sua infra-estrutura para fornecer electricidade e obterem leituras automáticas de contadores inteligentes (contadores com poder de processamento e ligações de rede). Com estas leituras actualizadas em tempo real, as companhias eléctricas conseguem ter um controlo elevado sobre o

equilíbrio necessário entre a produção e o consumo de electricidade. Se este equilíbrio não for mantido, podem ocorrer picos de tensão ou quebras na distribuição eléctrica, caso haja electricidade na rede a mais ou menos (respectivamente). Os picos de tensão são capazes de danificar equipamentos ao ponto de ficarem irreparáveis. As quebras na distribuição causam a paragem do funcionamento de alguns elementos ligados à rede eléctrica. Esta situação pode também ser perigosa, visto que, por exemplo, comboios eléctricos requerem um fornecimento continuado de corrente para o seu correcto funcionamento.

Na rede eléctrica a corrente é transmitida através de uma onda sinusoidal. A modelação desta onda é o que permite a comunicação em powerline. Às várias amplitudes de onda podem ser atribuídos valores lógicos - por exemplo, podemos atribuir à amplitude mínima da onda o valor lógico 0 e à amplitude máxima o valor lógico 1. Outras configurações mais complexas são possíveis. A onda eléctrica é modulada de modo a que se consigam ler os valores pretendidos na amplitude da onda, atingindo assim a passagem de informação na mesma infra-estrutura que providencia electricidade.

As companhias que produzem dispositivos para powerline juntaram-se em alianças, de modo a que todos os dispositivos produzidos pelos membros sejam padronizados e compatíveis entre si. Estes standards podem ser de acesso livre ou apenas para membros da aliança. A maioria destes protocolos inclui mecanismos de segurança. No entanto, alguns destes mecanismos já foram demonstrados como sendo inseguros, permitindo (por exemplo) que atacantes controlem a rede ou os dispositivos em si.

Este trabalho é orientado à procura de vulnerabilidades de segurança em protocolos de powerline. Apresentamos um resumo de alguns dos protocolos usados actualmente, e efectuamos uma descrição mais aprofundada do protocolo HomePlug. Este é o protocolo escolhido para análise neste trabalho, visto ser amplamente usado em ambientes caseiros e por existir um fácil acesso a adaptores HomePlug. Identificámos uma vulnerabilidade de desenho presente num dos mecanismos de troca de chaves criptográficas, que permite a um atacante que escute a rede durante a execução do protocolo obter as principais chaves de rede, conseguindo assim completo acesso à rede e à informação trocada nesta.

Para provar na práctica esta vulnerabilidade, precisamos de escutar a rede eléctrica. Dado que não sabemos construir um dispositivo que ouça a transmissão de dados na rede eléctrica, optámos por modificar um adaptador já existente que corre uma versão minimalista de Linux. Efectuámos com sucesso actualizações ao *firmware* do adaptador, de modo a conseguirmos acesso remoto com privilégios de administrador. Por acedermos ao adaptador conseguimos roubar informações e chaves criptográficas, o que só por si é uma contribuição apesar de não ser o objectivo deste trabalho. Acesso a um novo elemento da rede permite-nos fazer novos ataques, e como tal apresentamos várias possibilidades de ataque à rede e a dispositivos utilizando adaptadores de powerline.

Neste adaptador analisámos a execução do protocolo vulnerável, corremos um analisador de tráfego, colocámos binários, *device drivers*, e explicamos como modificar o

núcleo e o *bootloader*. Infelizmente, nenhum dos testes realizados serviu para provar na práctica a vulnerabilidade. Apesar de concluirmos que alguma informação do HomePlug chega a *user level*, as mensagens específicas do HomePlug continuam encobertas, fora do nosso alcance. Algumas possibilidades ainda estão em aberto para obter estas mensagens são descritas, sendo uma possível continuação deste trabalho.

Palavras-chave: Powerline, Segurança, Chaves criptográficas, Hacking.

Abstract

Powerline communication (PLC) is a form of data transfer, where the electric infrastructure is used for both power supply and network connection. PLC can be employed in industrial or home environments. In home environments, powerline is used to extend the internet connectivity through the house's electric infrastructure. Powerline adapters are connected to a house's power sockets, and these adapters provide connectivity throughout the house. A router is linked to one of the adapters to establish the connection, and other adapters are used to decode the powerline signal. These adapters provide an easy manner to extend a home network without the use of various routers, Wi-Fi, repeaters or new cables. In industrial environments, PLC is used (for example) to provide real time data about the electric consumption in the electric grid, allowing fine control of the required/used electricity. With this control, electric suppliers produce electricity more efficiently, reducing production costs and prices for the final consumers. Device manufacturers created alliances to standardize their products, developing protocols and guidelines to this effect. We present a summary of some of these standards. These protocols include security measures in their specifications (like cryptography), but some protocols have already been proven unsafe. In this work, we study the HomePlug protocol which is commonly used to extend connectivity inside homes. We describe a design vulnerability present in the HomePlug, in one of the cryptographic key exchange mechanisms. An attacker who listens to the medium can steal the critical network keys. To prove this vulnerability, we created a malicious adaptor by updating it with malicious firmware. Although we ran a large battery of tests in the adaptor, we were unable to prove the vulnerability. Nevertheless, we provide an insight on a series of attacks that can be done using a malicious adaptor as an attack point, which can be used in the future to extend this work.

Keywords: Powerline, Security, Network keys, Hacking.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Powerline communication (PLC) is a form of data transfer through the alternate electric current (AC) conductor, using it for both network and power supply [1]. Its main advantage is the use of a single infrastructure for these two purposes. The re-use of the electric infrastructure reduces costs, since there is no need for two installations - one for electricity and one for network connectivity. The electric wave has different frequencies depending on the distance between connection points, due to the nature of the electric current and distribution. Connections inside homes have high frequencies (1.8-250 MHz), allowing broadband connections (data rates up to several hundred MB/s), while high voltage cables use low frequencies (3-500 kHz), providing narrowband network (data rates up to 500 KB/s) [1].

Broadband powerline networks are available for household environments. Home PLC offers comfort to users by creating a network using the existing electric infrastructure of the house, which is employed to extend connectivity to the various rooms. The need for this kind of network comes from the typical internet installation case, where an Internet Service Provider (ISP) sets up a router in a division of the house. Devices in this room get internet access by connecting directly to the router with an Ethernet cable, but in the remaining divisions of the house there is no signal. To extend the router's signal, one can use Wi-Fi (if the router has such capability), signal repeaters (for Wi-Fi), extend new network cables, or powerline. It is also possible to install more routers, but this is typically more expensive and a cheaper solution usually suffices. Powerline adaptors' main advantages are reliability and simplicity, which is a trade off with some initial cost of buying the equipment. These provide connection speeds ranging from 200Mb/s up to 1200Mb/s depending on the adaptor, but the signal can be affected by many factors that might force a decrease on the bandwidth. Table 1.1 compares some aspects of powerline with other signal extending solutions.

A simple example scenario of extended connectivity using powerline is present in Figure 1.1. An internet connection is provided by the ISP, as we commonly have in our homes. Then, the ISP's router is connected to one adaptor through an Ethernet cable and

Solutions	**Signal range**	**Cost**	**Signal robustness**
Powerline	Up to 300m	±€45 for 2 adaptors	Influenced by medium noise
Wi-fi and signal repeaters	Varies; typically ±20m	±€30	Influenced by physical aspects (walls, etc)
Cables	High	Installation cost (±€40)	Very high
Extra Routers	High	Installation + router cost (±€70)	Equal to cables and/or wi-fi

Table 1.1: Comparison between the various technologies used to extend connectivity.

this adaptor is linked to the power strip through a power plug. Somewhere else in the house, another adaptor is linked to a power plug on the same power strip. Note that in home environments, there is typically only one power strip throughout all rooms. Finally, the computer is connected to the second adaptor, gaining access to the router without installing any extra cables through the house. Powerline in home environments can also be used to provide easy integration with the Internet of Things (IoT) [2], since devices (for example, a fridge) can be easily connected to home controllers, and also to the internet.

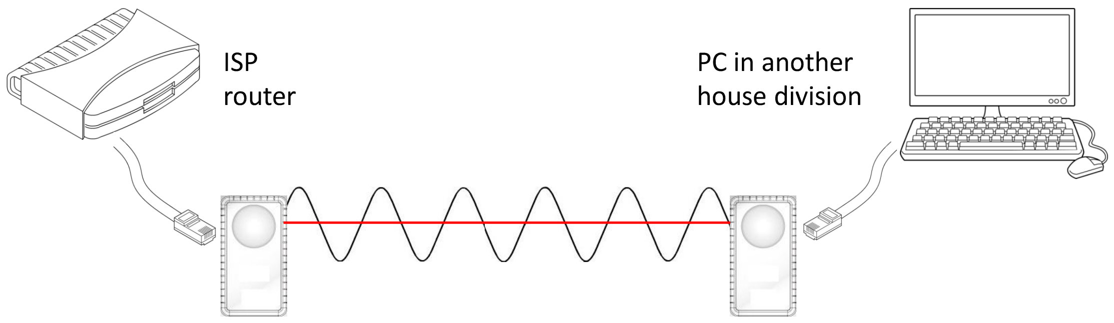


Figure 1.1: Example of a simple powerline network connecting a computer and an ISP router.

Narrowband powerline is used in industrial environments. These represent electricity distribution, Supervisory Control And Data Acquisition (SCADA) systems [3], and other infrastructure elements that provide services to homes or industries. Industrial powerline networks are used primarily for data collection, and typically cover large geographical areas. The power grid is one of the main industrial environments where powerline communication is used. In the power grid, the balance between produced and consumed energy must be kept. If too much energy is being produced, voltage peaks occur. These may damage hardware circuits, typically shutting them down and possibly impairing them beyond repair. On the other hand, if too little energy is injected on the grid, a power outage may happen, meaning that there is not enough energy for all connected elements and some of these will shut down due to lack of power. Both events are problematic and can affect society in general. For example, electric trains require a continuous electric supply to function correctly.

For a better control of the energy produced/consumed, electric utilities are replacing classic electric meters with smart meters [4]. These meters use powerline communication to provide periodic readings to the utilities, transmitting a more accurate value of how much energy is being demanded on the power grid at all times. By knowing how much energy is necessary, power generation can be adjusted to match the demands. This fine control over the electric production improves the safety of the power grid, as well as reducing electric generation costs - which should also mean decreased costs for the consumers.

Nowadays, companies have organized themselves in alliances to define PLC standards for their products, ensuring that all devices created by the members are compatible. Most of them focus either on home or industrial networks. For instance, the PRIME alliance defines a standard for industrial networks (see Chapter 2 for other examples).

The AC electric supply is defined by a voltage oscillation according to a sinusoidal function [5]. This sinusoidal defines the minimum/maximum voltage values and the frequency of the electric wave. Powerline communication is possible by using this sinusoidal, modulating the wave to control its behaviour and read logical values from the amplitude of the wave. Given maximum wave amplitudes $-\alpha$ and α, then $-\alpha$ could be interpreted as binary 0 and α as binary 1. The wave would be modulated so that the peaks $-\alpha$ and α would match the bits of the transmitted data. A simple example is illustrated in Figure 1.2. In the amplitude of the wave we can read logical values that correspond to the bits of a file to be transmitted (using $\alpha = 220$, which is the typical voltage value supplied in our homes).

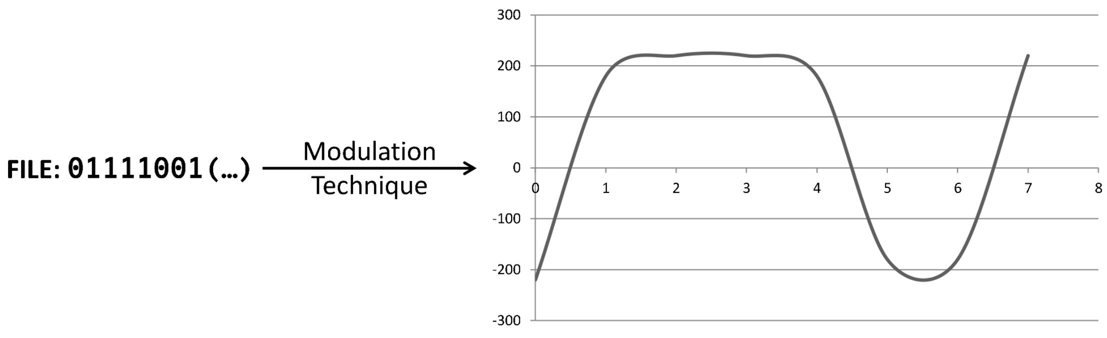


Figure 1.2: Example of a sinusoidal modulation to transmit data.

Other configurations are possible by modulating the wave and reading various logical values. Orthogonal frequency-division multiplexing (OFDM) [6, 7] is one of the principal techniques used in powerline communications to modulate the sinusoidal wave. It is a popular modulation technique that is used in several contexts like digital television and audio broadcasting, DSL Internet access, wireless networks, and 4G mobile communications. It is the modulation technique that prevents devices connected to the circuit from being affected by the changes to the sinusoidal wave. An equilibrium between the positive and negative voltages must be kept, or otherwise the modulation will cause voltage peaks that can damage circuits.

The wave frequency and modulation technique are the key factors that influence the transmission speed of the powerline. Other factors like the quality of the cables and the noise imprinted by devices connected to power supply [8] also influence the transmission speed (the higher the noise, the lower the speed).

1.1 Motivation

The powerline network can be a sensible component because it might transmit critical information. In industrial environments is sent information used to control heavy machinery and critical services over the powerline. For example, on the power grid, the network is used to transmit electric consumption readings and commands that are used to change the behaviour of field units (e.g., circuit breakers). Here, various attacks are possible, including changing the messages in transit or access the data to profile users. In the first case, an attacker tampers with the values sent by smart meters, with the objective of leading the electric utility to believe there is too little or too much power in the grid, causing a power outage or peak respectively. An attacker could also change the electric consumption values measured by the meter to make users pay for more or less energy than actually consumed. The second attack can reveal personal information about the consumers. By profiling the electric consumption of a house, an attacker may discover which devices are connected and if there are people present in the house at a given time [9, 10]. In addition, this information could be illegally used for marketing purposes. In home environments, the powerline is used to extend an internet connection. By accessing the powerline an attacker can listen or tamper with the users' internet traffic.

To protect communications, PLC standards resort to cryptographic approaches, such as encrypting the transmitted messages. To encrypt communications, every participant must possess the same encryption keys. So a reliable method is required to distribute these keys - a key distribution protocol must be robust. If there are flaws in the distribution mechanism, an attacker can compromise the key provisioning and steal the keys to listen to the network, change its messages, or impersonate participants of a network, amongst other attacks. We can classify key distribution methods into two categories: those that use the existing network and those that use an out-of-band mechanism. Methods that use the network always require some previous information exchange or a trusted third party [11]. Examples of these methods are Kerberos [12], signed Diffie-Hellman protocol [13], or Secure Sockets Layer (SSL) [14]. Out-of-band mechanisms require a secondary secure channel to transmit some data (such as identities, nonces - unpredictable random numbers - or keys) that allows the establishment of a secure channel on the insecure network. The out-of-band mechanisms are designed according to the environment in question. A few examples are the users of the devices that exchange the data directly [15], by inputting the data in the devices, or through a dedicated connection (e.g., a separate cable connecting the devices directly).

Poweline communications have already been attacked to some extent [16, 17, 18, 19, 20, 21, 22]. An example is presented next for each environment. In industrial connections, the DNP3 standard can be used for communications in the power grid. This standard has no security measures whatsoever, meaning an attacker with access to the network can freely tamper with every element present in it [16]. In home environments, the Zigbee protocol is vulnerable to denial of service attacks [21].

There are two key factors that make attacking the powerline difficult: access to the specifications, and access to the medium (to attack the protocols). Most protocols are closed, and only the device manufacturers can read them. Most of the information present in this work about these protocols was found on various sites on the internet. To access the powerline medium specific equipment is required, either by building a device or through commercial products. Building such device implies advanced knowledge on hardware circuits and electric components. The other solution is to reverse engineer [23] a compliant device. The technical specifications of these devices are mostly closed access as well. This means that attacking the devices or the powerline protocols is mainly black box testing [24] and reverse engineering. Given the underlying difficulties, we believe that any contributions made in this area are significant.

1.2 Goals

With this work we intend to assess the security of a chosen PLC protocol. If vulnerabilities exist, we explain why they are present and provide solutions. The chosen protocol was the HomePlug [25], mainly because it is used extensively in countries in Europe and also due to easy access to compliant devices. Our goals are to make a security analysis of the protocol, and if any flaws are found to prove their existence in theory and practice. For this work, we aim to answer three main research questions:

Q1: Are there any design vulnerabilities in the HomePlug protocol?

Q2: Are the key exchange mechanisms of the HomePlug robust?

Q3: Can we break the security of the protocol in a realistic scenario? i.e.,

- *Q3.1*: Can we infiltrate/modify a HomePlug adaptor?
- *Q3.2*: Can we use this adaptor to demonstrate experimentally the design security flaws that are eventually found?
- *Q3.3*: Can we demonstrate the flaws in a realistic scenario?

1.3 Contributions

We present an overview of the current powerline standards, both in industrial and home environments. Since our target is the HomePlug protocol, we provide a complete summary of it. This work identifies a flaw in one of the cryptographic key exchange mechanisms of HomePlug. An attacker who listens to this key exchange can steal the network keys, and freely infiltrate a HomePlug network. We explain the theory behind the flaw and the planning on how we intended to demonstrate it.

To access the powerline network and do the attack, we needed a HomePlug Sniffer. Since we do not possess the expertise to build a powerline Sniffer, we reverse engineered and hacked a Devolo dLAN 500 WiFi adaptor, which is HomePlug compliant. We chose this device in particular because it runs Linux, an operating system we are familiar with. We provide a complete description of our malicious adaptor's components and how it works. We introduced malicious software in the adaptor by abusing the update mechanism. We obtained root access to the adaptor, gathered its configurations and some critical keys. We also cross-compiled binaries to it [26, 27], changed/added kernel objects, included a new network driver, and explained how to change its kernel and bootloader. By hacking a powerline adaptor we get access to a new element in a network, which has the perspective of a relay between two points. Using this network position, we provide suggestions for a series of potentially dangerous network attacks using powerline adaptors as attack vectors.

We traced the execution of the flawed protocol in our adaptor. However, the adaptor limits the access to the HomePlug specific messages, which preclude the complete demonstration of the flaw. We present the adaptor's behaviour during the protocol execution, but somehow it prevents specific messages from being displayed in traffic analysers. We were unable to read the majority of HomePlug messages, including the ones required to demonstrate the identified flaw of the protocol. We also triggered the protocol manually using a program made by us, in an attempt to separate the protocol from its implementation in the adaptor. Unfortunately, this also provided no results. Nevertheless, we believe that information about the HomePlug messages (if not the packets themselves) reach the Linux user space. We enumerate possible changes that can be done in an attempt to obtain the HomePlug messages and disclose the vulnerability.

Concluding, we positively answer research questions *Q1*, *Q2*, and *Q3.1*. Although we traced the execution of the flawed protocol in our adaptor, it was not enough to give *Q3.2* an affirmative answer. Consequently, *Q3.3* remains unanswered.

1.4 Work plan

This work was developed in LaSIGE - Laboratório de Sistemas Informáticos de Grande Escala, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, under the supervision of professor Nuno Neves and professor Alysson Bessani.

The original work plan starts on 01/10/14 and finishes on 30/06/15, with the following phases:

(1) 01/10 - 28/11: Study of a set of powerline protocols. Selection of a protocol to study for the remainder of the thesis;

(2) 01/12 - 31/12: Analysis of the protocol for security flaws;

(3) 01/01 - 31/01: Design of solutions to detect vulnerabilities in the configuration or implementation of the protocol;

(4) 15/01 - 15/06: Implementation of the solutions and evaluation;

(5) 01/04 - 30/06: Description of the work in a report or thesis.

In the original work plan our study target was the PRIME protocol [28, 29], which is used by Energias de Portugal (EDP) for communication in the power grid. To efficiently run practical tests, we proposed a flexible attack tool to streamline the vulnerability testing process. We have access to the PRIME protocol specification because it is open access.

To test if the communication in the Portuguese smart grid is safe, we need access to the specific powerline devices used by EDP, which use a unique implementation of the PRIME protocol. Although smart meters are available on the market, the specific meters produced for EDP are not available for purchase. Obtaining the EDP smart grid elements is a long bureaucratic process, and we did not have access to them in time for this work.

Since we had to choose a new protocol, the HomePlug was our next target. We found the guidelines online [25], and HomePlug compliant devices are easy to obtain - they can be bought in any electronics store - and are fairly cheap (around €45 for two simple adaptors). We bought some of these adaptors, hoping we could obtain the HomePlug messages on a computer connected to them. After testing with traffic analysers running in a computer connected to one of these adaptors, we discovered that the powerline messages do not leave the powerline network. This means that in order to obtain a Sniffer, we have to either build one or make a HomePlug adaptor malicious. We opted for the second option, and successfully hacked a HomePlug adaptor. Nevertheless, hacking the adaptor was done in place of the attack tool since we did not have time to do both.

There was a small delay in the delivery of this document, because the student while working in this thesis, was also completing his contribution to the BioBankCloud project [30]. Fernando gathered data and finished writing a paper from October to the start of February, which was when the final article was submitted. Smaller contributions were also provided until the paper was finally accepted, in mid-June [31].

Summarizing, steps (1) and (2) were finished successfully, and the HomePlug protocol is our target. For step (3) we had proposed a flexible attack tool, but later we found we had to build a specific Sniffer for the HomePlug. This lead us to construct the Sniffer in place of the tool. Steps (4) and (5) were also successfully completed, although there was a small delay in the delivery of the thesis.

1.5 Structure of the document

This document is structured as follows:

- Chapter 2 - Related work: Describes some of the existing powerline protocols, their known vulnerabilities and reviews the most important features of the HomePlug protocol.
- Chapter 3 - HomePlug key exchange mechanisms and our attack plan: Presents a the key exchange mechanisms of the HomePlug, the security design flaw we discovered, and how we intend to demonstrate it.
- Chapter 4 - Implementation and evaluation: Explains the implementation of our work and the obtained results.
- Chapter 5 - Conclusions: Summarizes the contributions of the thesis and discusses future work.

Chapter 2

Related work

This chapter presents some of the protocols currently in use for PLC. These protocols are employed to guarantee the compliance of the products from various device manufacturers. The standards are divided into two categories: standards for industrial and for home environments. PLC in home environments is mainly used to extend network connectivity throughout a house. In industrial environments PLC is utilized to provide connections between various elements of large infrastructures, industry, and services. In Table 2.1 there is a summary of the most relevant PLC protocols. It also mentions the IEEE 1901 [32] and 1902.1 [33] standards, which are two of the base standards for PLC.

Some of the consortia do not provide open access to their guidelines, requiring in some cases a membership fee. Since we do not have such membership, we had no access to the guidelines/protocols documentation, and therefore only a summary is presented of the information that was available in the Internet.

Later in this chapter, we explain some security vulnerabilities found in PLC protocols. The already disclosed security issues were used as inspiration for our security analysis of the HomePlug standard, which is the protocol chosen for this work. An in depth discussion about the HomePlug protocol and its compliant devices is presented in the last section of this chapter.

2.1 Home environment standards

Home environments are characterized by a low voltage/high frequency electric infrastructure. The high frequency of the electric wave allows broadband data rates, like a connection through an Ethernet cable. Thus, in home environments the powerline can be used to extend internet connections and/or to provide broadband connections between devices, such as for media streaming. In the following we briefly describe the main protocols used for this purpose.

Name	**Consortium**	**Website (http://)**	**Standard date**	**Standard version**
		Home Environment		
HD-PLC	HD-PLC Alliance	www.hd-plc-org/	?	?
HomePlug Alliance	HomePlug Alliance	www.homeplug.org/	2013	1
IEEE 1901	IEEE	standards.ieee.org/	2010	1
		Industrial Environment		
DNP3	Distributed Network Protocol Group	www.dnp.org/	2010	3
G3	G3-PLC Alliance	www.g3-plc.com/	2011-2014	2.0 - 1.1
EN 12757-2/3 (M-BUS)	European Standard	www.m-bus.com/	1998	4.8
Meters and More	Meters and More Open Technologies	www.metersandmore.com/	?	?
OSGP	OSGP Alliance	www.osgp.org/	2012	1
PRIME	PRIME Alliance	www.prime-alliance.org/	2014	1.4
Zigbee	Zigbee Alliance	www.zigbee.org/	2010 - 2013	1.0 - 1.2
IEEE 1901.2	IEEE	standards.ieee.org/	2013	1

Table 2.1: PLC standards, publishing alliances, websites and versions/publish dates.

2.1.1 HD-PLC

The focus of HD-PLC is the creation of a simple PLC network in a home environment [34]. The HD-PLC alliance only provides some information on the physical layer of their protocol, which is similar to the physical layer of the HomePlug (e.g., modulation technique). As they maintain their protocols closed, we have no further information about their network topology.

2.1.2 HomePlug Alliance

The HomePlug Alliance provides a standard for PLC in home environments [25, 33]. The guidelines define the usage of an electric infrastructure as a network, resorting to the electrical medium as a network extension. This network can be used by various devices to communicate with each other, just like Ethernet cables or Wi-Fi. A HomePlug network is created by connecting adaptors (small electronic devices that modulate the electric wave to transfer data) to power plugs and pushing a button present in these adaptors. Then, devices (such as computers) can be linked through an Ethernet cable to these adaptors for connectivity.

We chose the HomePlug for this mainly due to ease of access to HomePlug compliant devices. These devices can be bought in electronic stores, and are fairly cheap. Also, there is little work done around the HomePlug, which means this is also a great opportunity to find something new that might be relevant to the scientific community.

2.1.3 IEEE 1901 standard

The IEEE 1901 standard is designed for high speed communication devices via powerline [33]. This standard focuses on the balanced and efficient use of the powerline communications channel assuring that the desired bandwidth and quality of service may be provided. Security issues related to the privacy of communications are also addressed. The standard describes only the physical layer and the medium access to the data link layer, as defined by Open Systems Interconnection (OSI) model.

2.2 Industrial environment standards

Industrial environments comprise large infrastructures, such as electric distribution. Over large cables, the frequency of the electric wave is low (when compared to home installations), providing only narrowband connections. However, these data rates are sufficient for their purpose, which is transmitting data readings from meters.

Figure 2.1 presents the distribution of these protocols across Europe. Spain uses both PRIME and Meters and More protocols, but the last one is less spread.

2.2.1 DNP3

DNP3 is the third version of the Distributed Network Protocol [35]. The development of DNP is centered on achieving open, standards-based interoperability between substation computers, Remote Terminal Units (RTUs), Intelligent Electronic Devices (IEDs), and master stations of the electric utility. This specification covers multiple communication layers (such as physical and link), and the protocol may interconnect with the TCP/IP protocol to use the Internet with the PLC link. On the application level, DNP uses an event oriented data reporting for improved bandwidth efficiency.

The DNP master gathers control information about its network in the form of events, which are related to noteworthy status changes and classified according to a priority policy. There are four priority Classes, labelled from 0 to 3. The 0 Class is special, since it is defined as the "static" or current status of the monitored data. This means that Class 0 does not contain events, but the status of the device.

The RTUs monitor data points (e.g., smart meters) and generate events when data should be reported. Although RTUs can also be configured to spontaneously report Class 1, 2, or 3 data, in the normal use case RTUs buffer the events. The master queries RTUs for an Integrity Poll (a combined read of Class 1, 2, 3 and 0 data), causing the RTUs to

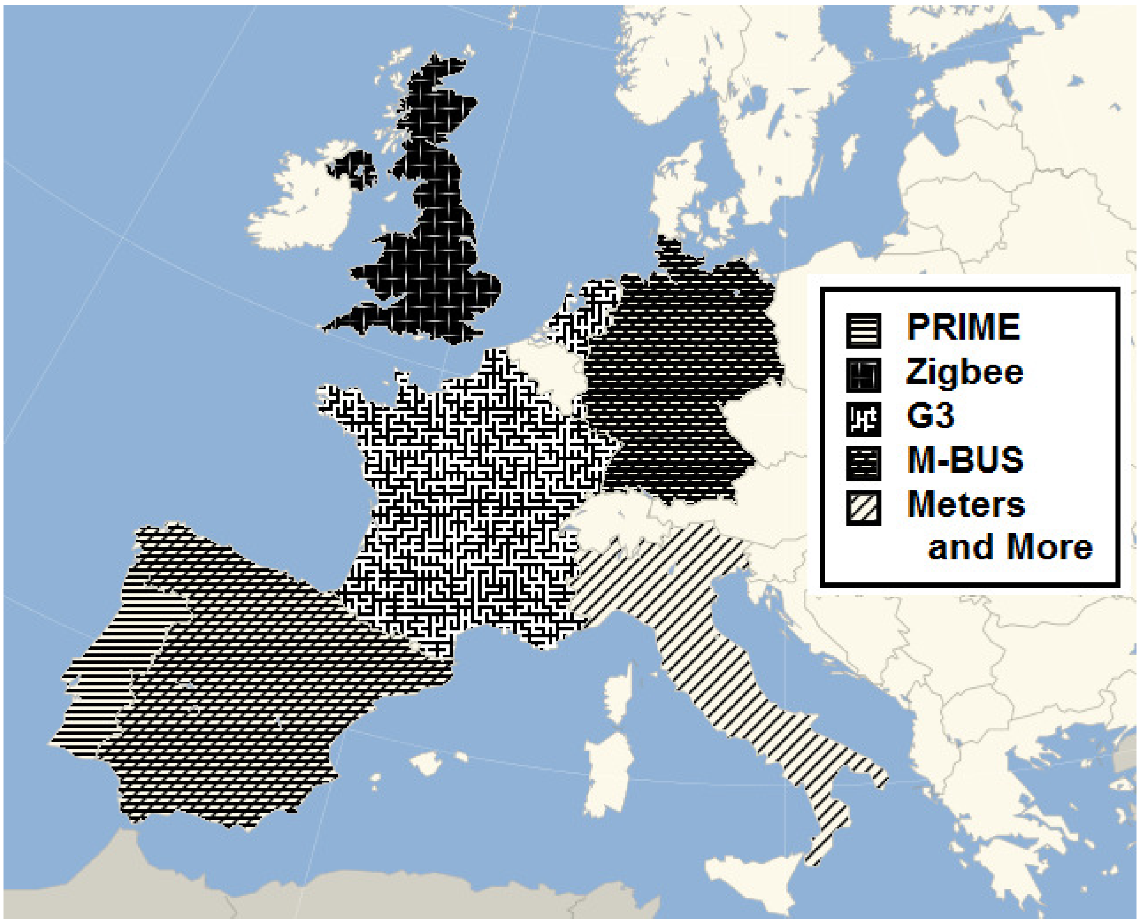


Figure 2.1: Distribution of PLC protocols across Europe.

send all gathered events and static data to the Master station. The received data can be processed all together or divided by classes, allowing different priority schemes.

2.2.2 G3

G3 contains a set of guidelines for various smart grid applications, including electricity utilities, equipment manufacturers, system integrators, and IT vendors [36, 37]. Future versions may include other areas, such as power grid management, remote meter management, and electric vehicle charging, as well as supporting interoperability among vendors. The G3 protocol implements IPv6 on top of the PLC MAC layer to emulate a typical internet connection between two elements of the smart grid. This allows the usage of well-known safe and secure protocols on the smart grid. Two network architectures are proposed, one that is centralized and another decentralized. In the centralized architecture, data concentrators simply act as a network gateway and the meters dialogue only with the central servers. In the decentralized architecture, the data concentrators operate as an application relay, with different levels of autonomy.

2.2.3 Meter-BUS

Meter-BUS (M-BUS) is an European standard (EN 13757-2,3) for the remote reading of meter values [38]. It can be used for multiple types of meters, such as electricity, water or heat. The M-BUS follows a bus topology, where all devices are connected. A master-slave approach is used, where the master coordinates the communications of the slaves, which in this case are the meters.

M-BUS is inspired by the OSI model, providing an implementation of the physical, data, network and application layers. The other layers of the OSI model are not applicable to M-BUS, since M-BUS is a bus system and not a network. The physical, data and network layers handle the data transmission between master and slaves, while the application layer is oriented to the management of the meters and to data collection.

2.2.4 Meters and More

Meters and More Open Technologies is a group focused on protocols for the smart grid from the distribution side. Their guidelines are oriented towards electric metering and distribution from stations to homes. Meters and More maintains a closed protocol description. However, we obtained some information about the protocol through the Open Meter project [39], which has some project with details about Meters and More (Deliverables D5.1, D5.2, and D5.3).

Meters and More uses a tree topology, where data concentrators are in the root position and act as masters. The non-root non-leaf tree nodes are called “A-Nodes”, and are typically repeaters. “A-Nodes” manage the sub-network bellow them in the tree, act as slaves to the concentrators and as masters to the “B-Nodes”. The “B-Nodes” are the leaf nodes of the tree which are the customer end devices, such as smart meters. The Meters and More communication structure is based on the OSI model, defining physical, MAC and application layers.

2.2.5 Open Smart Grid Protocol

The Open Smart Grid Protocol (OSGP) is designed to support the communication requirements of a large scale deployment of smart-grid devices and utility suppliers [40]. OSGP provides its own communication standard inspired by the OSI model, divided in three layers: physical (OSI layer 1), communication (OSI layers 2-6) and application (OSI layer 7). The physical layer handles data transmission over the physical medium. The communication layer ensures communication between parties (like the IP protocol), and implements security on the network. The OSGP application layer offers a “Structured Query Language (SQL)” like language, where all the information of the grid is kept in tables. A global table is maintained in the master, which contains the status of all the elements present in the grid. The various elements of the smart grid make queries to those tables to send/receive data. This SQL like architecture aims to provide a highly efficient data transfer between the different devices in the grid.

2.2.6 Powerline Intelligent Metering Evolution

The Powerline Intelligent Metering Evolution (PRIME) Alliance is defining an interoperable standard for industrial networks [28, 29]. PRIME is designed for the management of the power grid, and is currently deployed in Asia, Oceania, Americas and Europe, including Portugal.

This protocol organizes the network in a tree, where the root node is the coordinator, usually placed on a data concentrator. Other devices in the grid are nodes of the tree. The master controls every node in the network. Branch nodes (as in non-leaf non-root nodes) act as switches for the nodes below them in the tree. Leaf nodes are terminal nodes. Nodes can communicate between themselves (peer-to-peer), with the master node or with broadcasts to the network. When a leaf node receives a connection request from another device trying to join the network, it asks to master for a promotion to become a branch, and then starts routing communication between the requesting node and the master.

PRIME provides its own communication layers inspired by the OSI model, implementing physical, MAC, and network (IPv4, and IPv6) layers. Above that, there is an application layer, where the smart grid software is implemented.

2.2.7 Zigbee

The Zigbee Alliance produces standards for connections between smart objects in different environments [41]. Zigbee presents a standard for PLC, although the core of its products is wireless communication.

The Smart Energy Profile (which is the Zigbee protocol for smart grids) implements a Representational State Transfer (REST) [42] architecture. It is built around the core actions of `get`, `head`, `put`, `post`, and `delete`, with the addition of a lightweight subscription mechanism. This mechanism is used when a client wishes to be notified of changes to a resource on a server.

The objective of this protocol implementation is to avoid distinctions between servers or clients. When a device exposes a resource acts as a server, and when interacts with a resource is a client. This allows a fully flexible network.

2.2.8 IEEE 1901.2 standard

The IEEE 1901.2 standard specifies communications for low-frequency narrowband powerline devices [32]. It addresses grid to utility metering, grid automation, and electric vehicle to charging station. Lighting and solar panel PLC are also potential uses of this communications standard. The standard addresses the necessary security requirements that assure communication privacy and allow use for security-sensitive services. This standard defines the physical layer and data link layer specifications.

2.3 Vulnerabilities in PLC

Some of the previous protocols have already been successfully attacked (for example DNP3 [16], HomePlug [17, 18], M-BUS [19], OSGP [20], Zigbee [21, 22]). This section explains these attacks.

2.3.1 DNP3

Attacks on DNP3 are particularly worrisome because almost full control of the DNP3 network is possible, due to the non-existence of security mechanisms [16]. DNP3 implementations typically do not employ encryption, authentication and authorization, and devices simply assume that all messages are valid.

The DNP infrastructure is composed of a master unit (which is the central controller of the grid), outstation devices and network/communication paths. Ninety-one different attacks against the DNP infrastructure have been identified, with different impact degrees ranging from denial of service to obtaining control of the master unit.

2.3.2 HomePlug

The key implementation of HomePlug AV networks is unsafe [17]. All HomePlug devices have by default the same Network Membership Key (NMK), which is used to control the admission of devices into a network. After a successful admission, a device receives a Network Encryption Key (NEK), which is used to protect communications between devices of the network. Since this key is always the same, a malicious user can enroll into any network using the default NMK and listen to the traffic (since it receives the NEK).

The attack previously described can be further amplified [18]. A third key is used by devices in a HomePlug network, the Device Access Key (DAK). This key is derived from the device's MAC address, and can be used to force the enrolment of a device into a network. Note the difference between keys: one is used by devices to join a network; the other is used to make a device join a network. The author believes that these MAC addresses/DAK were not supposed to be discoverable. However, since they are, an attacker can bring devices into his network, thus eventually obtaining full control of their communications (usually home devices, like computers).

In this thesis, we present other possible attacks to the HomePlug protocol using adaptors as attack vectors. Since an adaptor is a relay between two points, by accessing the adaptor and the flowing traffic, an attacker can act as Man-in-the-Middle [11].

2.3.3 M-BUS

A security analysis of the M-BUS protocol was performed, disclosing a list of major issues [19], ranging from inadequate key length to disclosure and manipulation of encrypted message contents:

- The standard recommends a 64-bit key, while the current suggested key size is at least 128-bit [43];
- Inappropriate key and Initialization Vector (IV) [11] derivation may disclose plaintexts [11];
- Missing integrity protection allows the modification of messages without detection;
- The synchronization mechanism is not authenticated, and by attacking the system's clock an attacker may cause a repetition of the key used in stream cypher [11];
- Due to lack of authentication for network management an attacker can become a rogue network relay point;
- The M-BUS system possesses alarms for attacks, and an attacker can recognize if these alarms have been triggered - this is useful to known if messages sent to a device contain (in)correct values;
- The key update mechanism is flawed and may lead to key disclosure.

The security analysis classifies M-BUS as insecure and incapable of competing with the current security challenges.

2.3.4 OSGP

A series of vulnerabilities in the OSG protocol have been identified [20]. The encryption and message-authentication mechanisms were designed to be lightweight, which reduces the security robustness. It is possible to completely recover the authentication and encryption keys by exploiting weaknesses in the digest function [44] and RC4 stream cipher algorithm [45].

The OSGP digest function does not adhere to any crypto standard and its design does not provide a secure message authentication code (MAC). A robust MAC function is irreversible, but the OSGP digest does not possess this property. RC4 should not be employed due to the known issues [46, 47, 48, 49].

Moreover, the standard does not provide entity authentication on the source of messages. Broadcasts are authenticated using the flawed digest function, which provide message authentication but not source authentication - a proper digital signature should provide both. Since firmware updates in OSGP are sent through broadcast messages, an attacker could send malicious firmware to all devices of the network. The firmware would be correctly authenticated with the flawed digest, and since the message source is not verified, the devices accept the update as legitimate.

2.3.5 Zigbee

Zigbee networks are vulnerable to a series of different attacks [21, 22]. The Zigbee specification supports two security levels: High Security (also referred to as Commercial Security) and Standard Security (also referred to as Residential Security) [21]. The Standard Security level transmits the network key in plain text during a device registration. An attacker who sniffs the network easily captures the key, and then can listen to all network traffic.

Zigbee devices use *nonces* (a unique random number) as part of the encryption key for further protection. However, this leaves the network open to the so called *Same-Nonce Attack*. If an attacker can cause devices to choose the same nonce twice, he will gather information about the plaintext. This attack is possible if the attacker can force devices to reboot to default values, which makes them repeat the nonces.

Zigbee devices are also vulnerable to denial of service attacks. These devices utilize a frame counter to prevent replay attacks, where each time a frame is received the device's frame counter is updated with the frame's sequence number. This counter has a maximum value given by a 32-bit counter, and is reset when the network key is refreshed. A frame is discarded if it has a sequence number lower than the counter. An attacker can send a frame with the maximum sequence number, which will be set as the device's frame counter. From that point on, all frames will be discarded because the frame counter will always have a sequence number higher than the received frames. This attack can be performed even if the frames are encrypted.

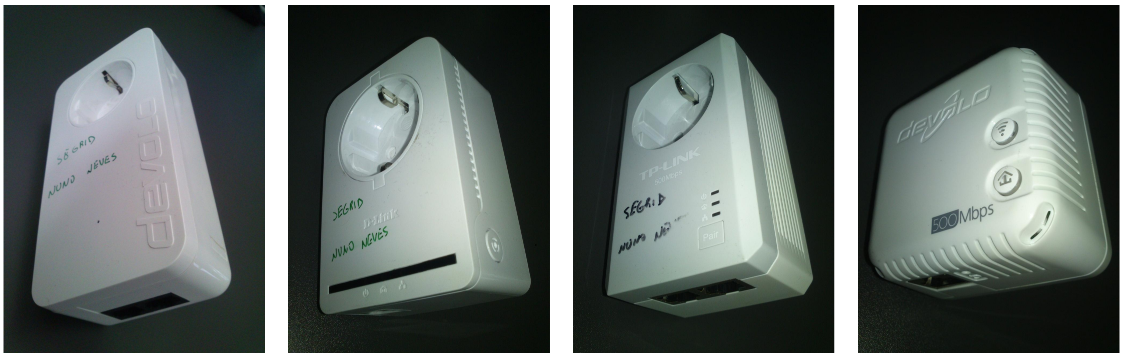
The Zigbee devices are also vulnerable to physical attacks, since keys can be stolen with direct access to a Zigbee device [22]. Unless devices are protected from physical tampering, an attacker can steal keys and then emulate the device to obtain access to the network.

2.4 HomePlug protocol and devices

This section describes the most important features of the HomePlug standard. We begin with a simple example of a network created using HomePlug adaptors, and then present the most relevant aspects of the protocol. We also explain the functioning of HomePlug compliant devices and some details of their topology.

2.4.1 HomePlug in practice

The HomePlug protocol is designed for use inside homes, to ease the extension of network connectivity. In the rest of this document, HomePlug complaint equipment will be referred to as *adaptors*, and other electronic gadgets linked to adaptors for connectivity as *devices*. Examples of adaptors are displayed in Figure 2.2, and devices are equipment like computers, routers, printers, fridges, and televisions.



(a) Devolo dLAN 500 duo+ (b) D-Link DHP-P309AV (c) TP-Link TL-PA4020PKIT (d) Devolo dLAN 500 WiFi

Figure 2.2: Examples of powerline adaptors.

Figure 2.3 shows a practical example of a powerline connection representing our test bed. An Ethernet cable connects the computer to the adaptor on the left. The other adaptor (on the right) is connected to a router through another Ethernet cable. The router is not seen in the figure due to the configuration of the room. The two adaptors communicate through the electric extension, representing two plugs of a house.

To establish a connection between devices (in the figure, between the PC and the router), we couple the adaptors to power plugs, and link the devices to the adaptors. However, this does not suffice. First, the adaptors must establish their own network. The most simple mechanism to establish a HomePlug network is to push the HomePlug buttons of each adaptor - all adaptors are manufactured with at least one button. We push the buttons, wait a short time period, and the network between adaptors is created. Only then the connection between the computer and the router is enabled. In summary, first the adaptors establish a primary connection between themselves, then provide network connectivity to the devices linked to them.

2.4.2 The HomePlug protocol

Since home environments are characterized by low voltage/high frequency electric capabilities, it is possible to offer broadband powerline communication between devices. The protocol is focused on providing a simple and secure network extension solution, while being user friendly.

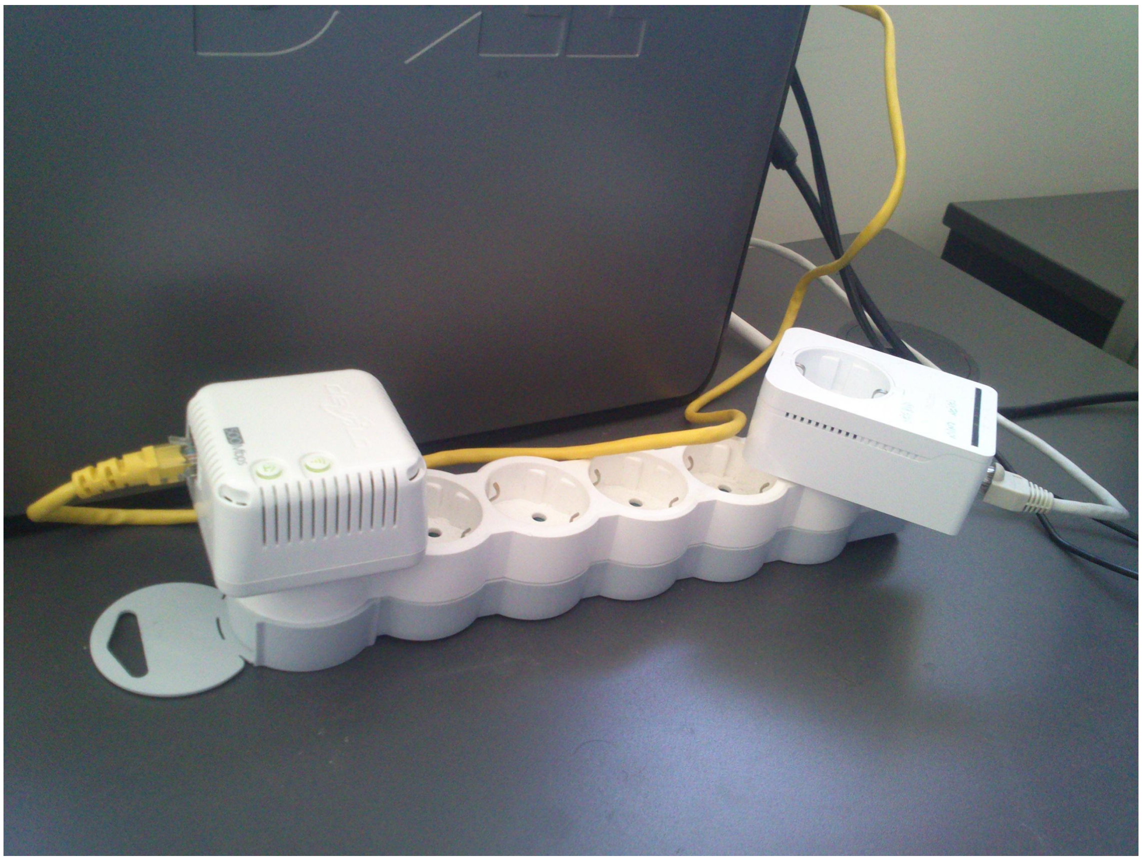


Figure 2.3: Practical example of a simple powerline network.

HomePlug adaptors establish their own private network, and is through this network that the devices connected to the adaptors exchange the packets. In the following sections we provide an overview of the HomePlug networks.

Establishing a network

The first step to create a HomePlug network is to connect HomePlug compliant adaptors to power plugs on the same power strip (these adaptors are presented in more detail in Section 2.4.3). adaptors in different power strips are physically isolated, and therefore cannot communicate. Then, a HomePlug network must be established. These are defined by all adaptors possessing the same Network Membership Key (NMK). A network is composed of one or more adaptors. Given n adaptors, any subset of them can be combined to form different networks, with the restriction that an adaptor can only be part of one network. adaptors in different networks are logically isolated from each other. To create networks or add adaptors to an existing network, one of the NMK provisioning methods described in Section 3.1 would be employed. Each network will have its unique Network Identifier (NID), calculated with an hash of the NMK.

Once the adaptors are in the desired network they can communicate among themselves, and provide connectivity to their devices (e.g., computers, routers, printers). Figure 2.4 shows an example of a set of adaptors creating two networks, where the dashed lines represent the separation between powerline networks. Since there is a separation amongst adaptors, the devices connected to them are also logically separated. The devices on the left network are unable to communicate through the powerline with the devices on the right.

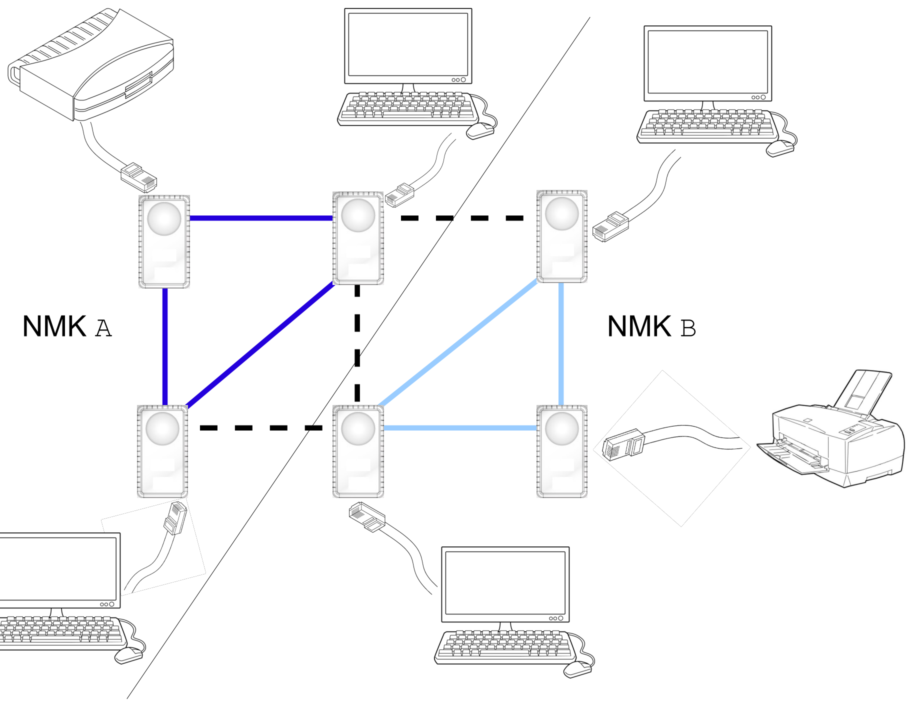


Figure 2.4: Example of a complex powerline connection with two networks.

Communication in HomePlug networks

The HomePlug protocol has its foundations in the physical (electrical) medium. As with any physical shared medium, adaptors must communicate in turns to avoid collisions between two or more messages - overlapping messages generate noise and preclude efficient communication. To prevent message tangling, HomePlug defines transmission beacons, during which participants have the medium for themselves. The network coordinator (see below) defines beacon lengths and attributes beacons to the participants in round-robin. Network participants can also ask for more beacons or longer beacons, by sending a request to the coordinator.

The HomePlug protocols are organized in layers. On top of the physical layer there is an Ethernet layer. Each network participant has a pre-defined MAC address, which uniquely identifies it. Participants communicate with each other through Ethernet messages.

Figure 2.5 shows a simple powerline connection and the different network interfaces included in an adaptor. Messages sent by computer `A` reach the HomePlug participant (adaptor) `HA` through an Ethernet cable (in interface `eth1`) and then they are sent through the powerline to adaptor `HB` (through interface `eth0`). The adaptors isolate the network interfaces `eth0` and `eth1`, mediating the connection between them. This means that there is no direct access from the computer to the (physical) powerline.

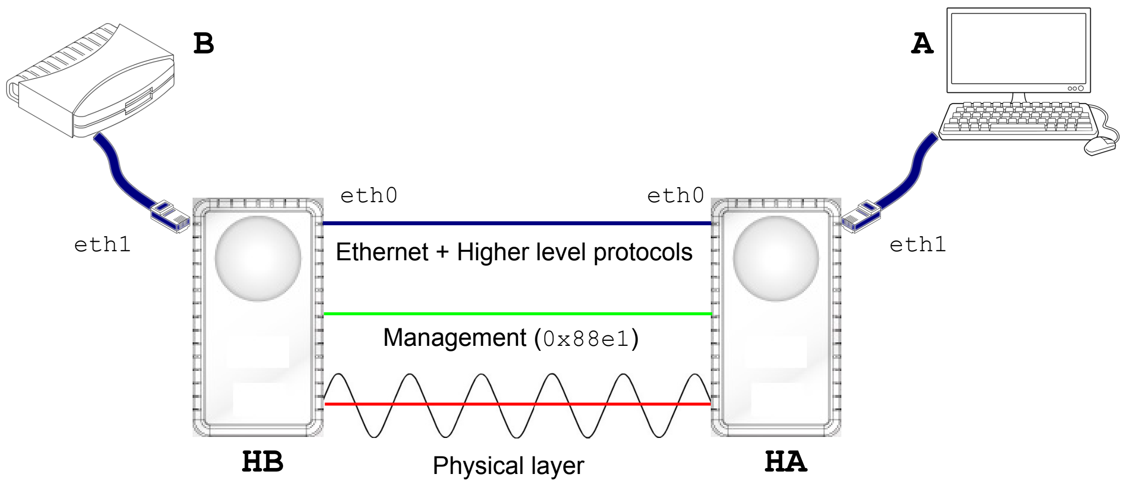


Figure 2.5: Examples of the logical interface isolation and the HomePlug network layers.

Since the HomePlug adaptors control which messages go through a specific interface, they can exchange management messages among themselves through the powerline interface, hidden from the devices connected to them. This creates the HomePlug management layer. Management messages are solely transmitted over the powerline and the adaptors never redirect these messages to the devices connected to them. In addition, in the HomePlug standard it is mentioned that some management messages are meant to be only transmitted over the powerline and never through the Ethernet (MAC layer). This is a security measure, since it prevents computers from sniffing or spoofing management messages. Only adaptors can listen or tamper with these messages, since they have exclusive access to the management layer. The management messages exchanged between HomePlug adaptors are normal Ethernet messages, with a specific *ethertype* `(0x88e1)`. Figure 2.5 presents a visual representation of these layers.

HomePlug network coordinators

Each HomePlug network has a coordinator. The coordinator manages the other adaptors present in the network, and both the physical and the Ethernet layers. It is also the coordinator that accepts adaptors into its network. The role of coordinator is dynamic, in the sense that any of the remaining participants can become the new coordinator if the current one leaves the network (depends on adaptor configuration; see Section 2.4.3 for more details).

The coordinator can decide to separate its network into sub-networks, "*for secure distribution of different Network Encryption Keys*" (NEK - presented below) [25]. Each sub-network will have its own sub-coordinator, and all the sub-coordinators communicate with the main coordinator for message exchanges that need to cross the sub-networks, as well as for management messages between sub-networks. Each sub-network will have its NMK and NEK. The main coordinator will possess all of the NMKs of the sub-networks it manages. In addition, each network and sub-network has a Network Identifier (NID).

Security

The traffic generated by the devices is encrypted while transferred between adaptors, and integrity is checked (only) at physical level. All key exchange mechanisms define specific keys to encrypt the protocol messages. Most of the management messages exchanged are also encrypted. The HomePlug standard defines multiple encryption keys, each for a different context. Nonces are also used in some management messages to prevent replay-attacks [1] [11]. The encryption algorithm employed by the HomePlug is the Advanced Encryption Standard (AES) [50], which is the current standard for encryption in networks.

In the next subsection we give a summary of the most relevant keys. First we present the default keys, placed in adaptors during the manufacturing process. Then we describe the keys to encrypt the frames exchanged in the network, and finally we present the keys to establish private channels between two adaptors.

Default keys

- **Device Access Key (DAK):** The DAK is a unique key placed in the adaptor during the manufacturing process. Besides identifying the adaptor, the DAK can be used to send the NMK to another adaptor, through the "Providing NMK using DAK" protocol (detailed in Section 3.1). The DAK must never be sent through the network, which means that if adaptor `A` wants to share the NMK with adaptor `B` using `B`'s DAK, `A` must obtain `B`'s DAK by some other mean than the network - through the user or some other out-of-band mechanism. This key is used as a MAC encryption key.

- **Network Membership Key (NMK):** Adaptors always come with a default NMK. The network membership key defines which adaptors are part of a network. HomePlug defines alternative methods for the transmission of new NMKs (see Section 3.1). It is through the NMK that the NEK is distributed to the participants of a network. NMK is a network encryption key, changed when an adaptor joins a network.

[1]Attacks where previously sent messages are re-sent by an attacker, re-enacting a previous interaction among participants of a network.

Network encryption keys

- **Network Encryption Key (NEK):** This key is used to encrypt almost all messages exchanged between adaptors, as only a few management messages are allowed to be sent unencrypted. The traffic generated by the devices (e.g., computers) linked to the adaptors is encrypted with this key when it is transferred over the powerline. The device's traffic is encrypted before being sent over the powerline and decrypted when leaving it.

Encryption keys for private channels

- **Point-to-point Encryption Key (PPEK):** This key is used to encrypt point-to-point physical messages. Point-to-point encryption is supported on optional basis by the adaptors. Please, see below the End-to-end Encryption Key.
- **Temporary Encryption Key (TEK):** The TEK is used to encrypt messages during key exchange protocols on private channels between two adaptors. A new TEK is generated for each instance of a key exchange. TEKs have a limited life span and are never re-used. TEKs are provisioned using the adaptor's DAK or through the Unicast Key Exchange (described in Section 3.1).
- **End-to-end Encryption Key (EEK):** This key is used to encrypt point-to-point Ethernet messages. Both this key and the PPEK are used only when two adaptors wish to establish a private long duration channel between them. This key is distributed using the EDK (see below).
- **EEK Distribution Key (EDK):** This key's sole purpose is to distribute the EEK. The message providing an adaptor with the EDK has to be encrypted with the adaptor's DAK.

As mentioned in Section 2.4.2, only adaptors have access to the HomePlug management layer, where network configurations and encryption keys are exchanged. This can be considered a security measure, since it increases the difficulty to attack these messages. Nevertheless, it is possible to build modules capable of accessing the physical layer of powerline communication. An example module has already been built, using specific hardware components and is capable of sniffing simple powerline communications [51].

2.4.3 HomePlug adaptors

HomePlug adaptors are small electronic gadgets that when connected to the same power strip can communicate through the powerline. These adaptors can modulate the electric current's sinusoidal to create a custom wave. This wave is perceived only by other adaptors, and does not influence the devices that are connected to a plug just for electric power (like televisions). The technique used by these adaptors to modulate the wave is outside the scope of this work.

Adaptors may not work as expected if there are current controlling/modelling devices between them. Devices like Uninterruptible Power Supply (UPSs) may change the modulation created by the adaptors, thus changing/removing the information imprinted in the electric wave. In addition, these adaptors are sensitive to the noise that normally exists in the power strip, as any device that is connected may change the electric sinusoidal [8]. Since the adaptors depend on the quality of the sinusoidal to communicate, noise in the power strip influences its communication speed and range. The quality of the cables of the power strip also impacts the quality of the signal. On environments with noise, the adaptors reduce the communication speed to increase their robustness to errors.

The adaptors have at least one Ethernet plug, and they may have a power plug that can be used just like any other plug on walls. This plug is present for commodity. Since the adaptor must use a plug, it provides one so that users do not "spend" a plug for each adaptor. The Ethernet plug is used to connect the adaptor to other devices, like computers, printers, etc. The adaptors generally have between one to three blinking light emitting diodes (leds). The leds usually indicate if the adaptor is on, if it is connected to other adaptors and if there is a device connected to it. The adaptors also possess a button. A short press on the button makes an adaptor join/create a HomePlug network. A long press removes the adaptor from its current network. The purpose of the button is further explained in Section 3.1.

There are also HomePlug adaptors with Wi-Fi capabilities. These adaptors provide a Wi-Fi signal just as a normal router, except that it gets its Internet connection from the powerline. They may also contain Ethernet plugs.

Device organization

The HomePlug adaptors that were analysed had chips made by Qualcomm Atheros [52], which is a developer of semiconductors for network communications. Besides producing the chips, Qualcomm released an open source toolkit called Open Powerline Toolkit (open-plc) [53]. This software suite is used to interact with these chips, which are configured using two files with names ending in `.pib` and `.nvm`.

The Parameter Information Block (PIB, stored in a `.pib` file) contains configurable attributes of the adaptor. The main attributes are the adaptor's MAC address, the keys DAK and NMK, and the coordinator status. Note that the coordinator status attribute is not discriminated in the standard but defined by Qualcomm. Adaptors can be in one of the following five statuses:

- **Auto:** The adaptor joins existing networks if capable and in alternative creates its own network.

- **Never:** The adaptor never takes the role of coordinator, which means that the adaptor will never create its own network.
- **Always:** The adaptor will always try to be the coordinator of the network where it is present. If this is not possible, it will form its own network.
- **User Assigned:** The user chooses the role of the adaptor, meaning it is not in a preconfigured status.
- **Covert:** We could not find what this status means.

The `.nvm` is a file that stores the firmware for the chip. This file is divided into six components: Chain Manifest, Memory Control Applet, Custom Module Update Applet, Power Management Applet, Generic Image, and Runtime Firmware. The information about the `.nvm` was retrieved from the `chknvm` tool that is included in the open-plc toolkit. We have no further details about the chip's firmware, since its structure is closed source. As stated in `chknvm`'s manual: "*Qualcomm Atheros firmware file structure and content is proprietary to Qualcomm Atheros, Ocala FL USA. Consequently, public information is not available.*"

Typical HomePlug adaptors only have a MAC address and as uploadable files, a `.pib` and an `.nvm`. However, the HomePlug adaptors with Wi-Fi capabilities require extra internal management - the Devolo adaptors with Wi-Fi capabilities we analysed include a minimalist Linux kernel and a file system.

Chapter 3

HomePlug key exchange mechanisms and our attack plan

This chapter describes mechanisms used for key distribution in the HomePlug protocol. We identify a flaw in the Unicast Key Exchange (UKE), and present a solution to overcome the vulnerability using a simple out-of-band mechanism.

We executed some initial tests to disclose the UKE flaw using a computer, but these were unsuccessful. In an attempt to demonstrate experimentally the vulnerability, we will modify a Devolo adaptor to make it perform malicious actions, such as listening to the network. This chapter explains the planning of the attack, and the technical details of the target adaptor.

3.1 NMK exchange mechanisms

The HomePlug protocol supports three NMK exchange mechanisms. These are used to allow an adaptor to join a network. Recall that a HomePlug network is defined by all adaptors in possession of the same NMK.

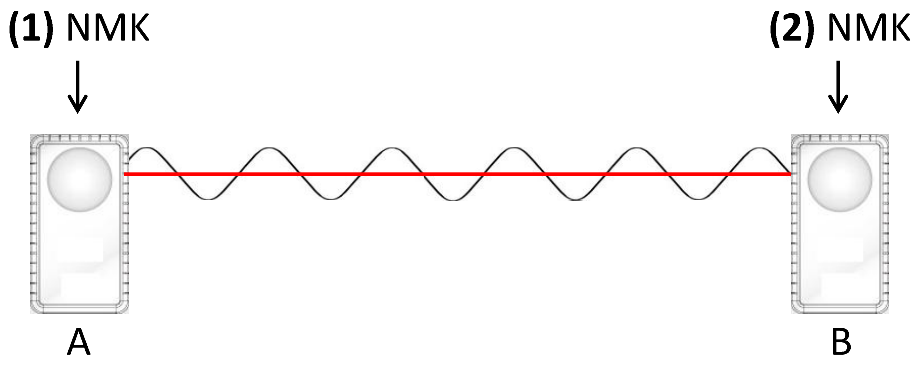
Providing NMK by direct entry (NDE)

With this method a user inputs the NMK directly into the adaptors. The standard does not describe how or by what means the key is provided. Adaptors are not required to provide a text entry interface, and during this work we did not come across an adaptor with such capability. However, there are mechanisms to directly provide the NMK to the adaptor, like programs: faifa [54], Qualcomm Atheros Open Powerline Toolkit [53], or Devolo's cockpit [55].

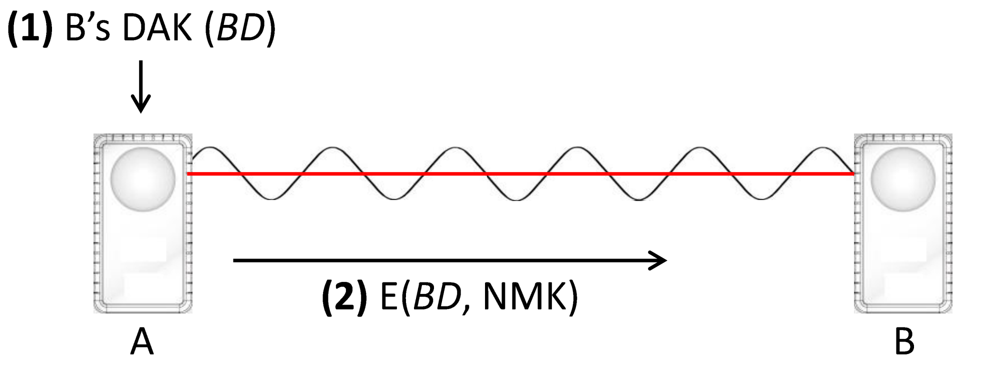
This method has the disadvantage of burdening the user with the task of providing the NMK, which may present some issues - for example, faifa is a command line program, which is typically a difficult to use interface for non-technical users. Besides, it is only available in Linux.

Providing NMK using DAK (NUD)

The NUD can be employed to add an adaptor to an existing network. Recall that a single adaptor creates its own network, and therefore if two adaptors A and B have different NMKs each adaptor has its own network. If the user wants to form a single network with both adaptors, the adaptor A can provide B its NMK by encrypting it with B's DAK key. From then on, B will use A's NMK and both adaptors will be in the same network. This method is illustrated and compared to NDE in Figure 3.1. E(*BD*, NMK) means that the message containing the NMK is encrypted with the key *BD*, the B's DAK key.



(a) Providing NMK by direct entry (NDE).



(b) Providing NMK using DAK (NUD).

Figure 3.1: Comparison of NDE and NUD.

However, since the DAK can never be sent through the network, this means that A can only receive B's DAK through an out-of-band mechanism. As with NDE, this method requires a user with access to programs and knowledge on how to provide a key to an adaptor.

Providing NMK using Unicast Key Exchange (UKE)

The UKE is also known as the *simple connect* method. HomePlug adaptors are not expected to possess an interface for text entry, but are expected to have a button. This button is used to call the UKE and to remove the adaptor from the current network (by pressing the button for 10 seconds or more). The UKE is a simple protocol with four messages, where in the end both adaptors possess the same NMK, thus being in the same network (depicted in Figure 3.2, further details in Appendix A).

To call the UKE, the user pushes the button of one adaptor for one second. This causes the adaptor to enter into *simple connect* mode, and to periodically send *CM_SC_JOIN.request* messages. These "join request" messages are used to signal neighbouring adaptors that there is an adaptor trying to join an existing network. Then, the user pushes the button of a second adaptor. This also causes the second adaptor to enter the *simple connect* mode, and to periodically send *CM_SC_JOIN.request* messages. Once an adaptor in *simple connect* mode receives a *CM_SC_JOIN.request*, it sends a *CM_SC_JOIN.confirm*. The "join confirm" is transmitted only by an adaptor in *simple connect* mode who received a "join request", meaning that the UKE will only be executed by adaptors who have been commanded by a user to be part of the same network. There is a time interval in which both buttons must be pushed, defined by the manufacturer of the adaptor. This interval is suggested to be between 30 seconds and 2 minutes [25].

Then the UKE protocol is executed. The first two UKE messages contain a random key generated by each participant (*CM_GET_KEY.request/confirm*). The keys included in these two messages are concatenated to produce a TEK key, used to encrypt the remaining two UKE messages (*CM_SET_KEY.request/confirm*) where the NMK is exchanged. Note that the standard states that UKE messages may be sent through the Ethernet, but the "join" messages may not (refer to Section 2.4.2).

3.1.1 Security analysis

The main characteristic of the key exchange mechanisms of the HomePlug protocol is that they always require direct intervention from a user to be safe. In any of the first two methods mentioned in Section 3.1 there are secure ways to place the NMK in the adaptors - we assume a network administrator would handle this task with ease. An attacker cannot (or should not) be able to obtain the keys exchanged using the out-of-band mechanisms. The UKE is vulnerable to a sniffing attack because the first two messages of the protocol are sent unencrypted over an open network. It is not possible to safely exchange keys in the insecure network because adaptors have no previous information about each other, and the presence of a trusted third party in the environment is not predicted. Even if a trusted third party would be present - for example, a Kerberos system - adaptors would have to be registered in Kerberos before being able to authenticate each other. The burden of such task would be put upon the user, and would be similar to the first two NMK exchange mechanisms - direct user intervention and user know-how are required.

In the environments where the UKE is assumed to be employed - users without knowledge or means to use the other key distribution methods - the key exchange is insecure. An

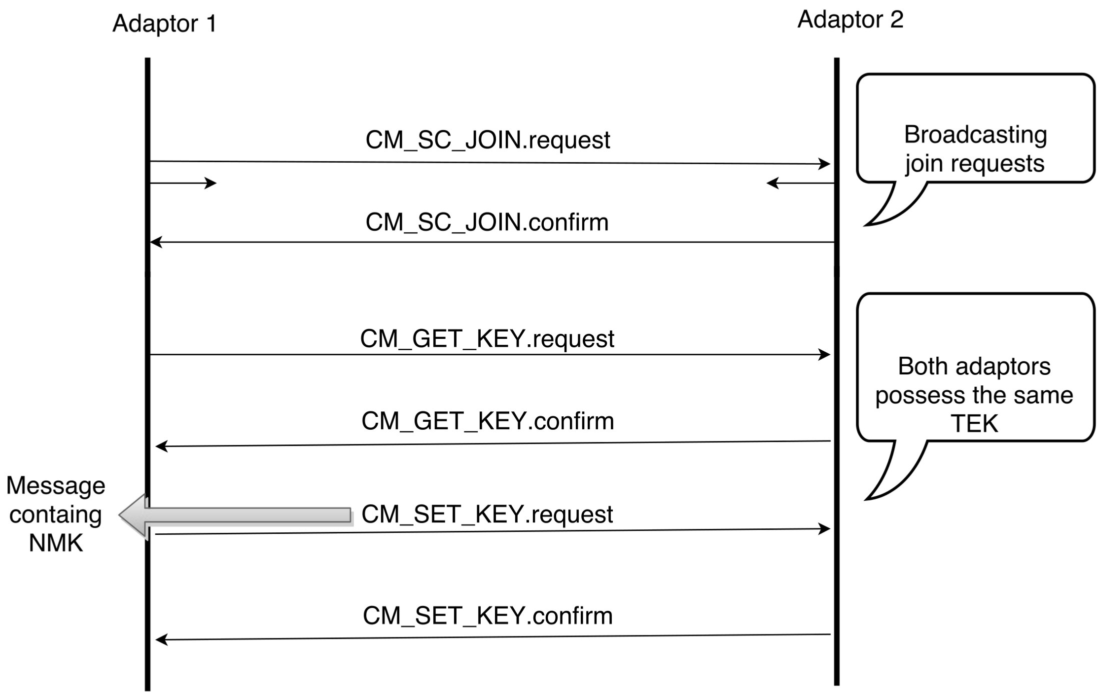


Figure 3.2: Diagram of the UKE protocol.

attacker with capabilities to listen to the powerline can obtain the two messages, creating the same TEK key as the adaptors. With this key, the attacker can decrypt the following two messages, obtaining the NMK. With the NMK the attacker can be part of the network. Recall that it is through the NMK that the NEK is transmitted (see Section 2.4.2) - the attacker waits for the NEK provisioning messages, encrypted with the same NMK that was obtained. With the NEK the attacker can listen to all traffic in the network, insert new packets, etc. These conclusions answer affirmatively research questions *Q1* and *Q2*.

Furthermore, the HomePlug protocol says the following about the UKE: "*This mechanism trades off convenience for users and low-cost user interfaces for lowered security. Simultaneous execution of this mechanism can cause networks to admit stations other than those desired by the user, and sufficiently equipped and sophisticated attackers can compromise the key exchange itself, so this is only recommended for nonsensitive information applications.*" The UKE protocol was designed trading safety for usability. The protocol designers know the flaw is there, all that remains is to practically prove it.

3.1.2 An alternative more secure to UKE

After explaining the security flaw, one can think of potential solutions. We propose an alternative to the UKE using a simple out-of-band mechanism. Together with every set of adaptors we purchased (most come in pairs) came two Ethernet cables. One of these cables could be used to connect the two adaptors. This creates an effective out-of-band mechanism, since no other entity can read the traffic exchanged through that cable. Figure 3.3 presents a representation example of our approach.

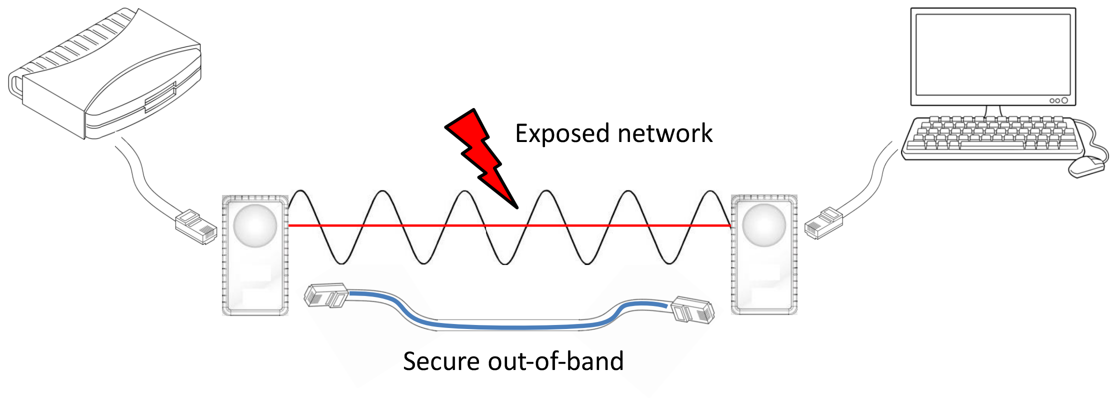


Figure 3.3: Proposed solution to the UKE vulnerability.

If an Ethernet cable is connected between two adaptors, they can assume that the user desires to make them part of the same network. Using this out-of-band connection the adaptors could identify themselves and use the UKE to exchange the NMK, exactly as explained in Section 3.1 but now resorting to the Ethernet cable as the communication medium. This solution is simple and safe, since key exchange is accomplished over a secure channel. This mechanism does not prevent a malicious user from adding a malicious adaptor to a network, but preventing physical access to the adaptors is outside the scope of this work. We focus only on remote network access and infiltration.

3.2 Exposing the UKE security flaw

To expose the UKE security flaw we need to listen to the packets exchanged during protocol execution. To do this, we tried using the wireshark program [56] to collect all messages exchanged in the network. The tool was run on a computer connected to an adaptor. We were able to observe HomePlug messages about:

- Bridging informations [57];
- Software version information;
- Network information: NID, MAC addresses of the present adaptors, transmission/reception rates, amongst others;
- Ethernet physical settings;
- Vendor messages (messages designed by adaptor manufacturers, not discriminated in the standard), for which we have no information about their format.

This very simple attack allowed us to obtain the MAC addresses of adaptors that are present in the network. However, these are mostly innocuous messages and not the ones we require. We experimented triggering the UKE in the adaptors, but the protocol messages did not reach wireshark. The UKE packets only travel in the HomePlug management layer, and are never forwarded to the computer. So, to capture these packets, it is necessary to be in the same position as an adaptor and access the management layer. To do so, we intend to modify an adaptor to make it collect the UKE packets. As mentioned in Section 2.4.3, the Wi-Fi adaptors include a Linux kernel and file system. Since these have well known formats, and given the need for a malicious adaptor, a part of our work was to modify a Devolo dLAN WiFi 500 powerline adaptor (described in Section 3.2.1). This was our target, hereafter mentioned as the malicious adaptor.

Creating a device capable of listening to the powerline medium is possible, but it would probably take too much time and effort because we do not have any experience assembling hardware devices. Therefore, the attack plan is as follows:

(1) Obtain remote access to the adaptor;

(2) Read the UKE packets:

  (a) Connect to the network;

  (b) Get access to the management layer;

  (c) Read the UKE packets.

To accomplish step (1) we need to analyse the firmware of the adaptor to understand if it is possible to include programs for remote access like `telnet` [58] or `ssh` [59]. By accessing the adaptor one should be able to connect to the network, thus completing step (2a). To read the traffic, we need to run a traffic analyser. When this task is completed, we should have full access to the network, thus accomplishing steps (2b) and (2c).

3.2.1 The Devolo dLAN WiFi 500 powerline adaptor

The Devolo dLAN WiFi 500 [60] has a chip that handles both powerline communication and Wi-Fi. The chip includes a *MIPS* processor, and complex circuits that implement the HomePlug management layer. We are not sure if the chip managed all HomePlug packets by itself, and this issue is further discussed in Chapter 4.

A crucial part for this work is the structure of the firmware that can be uploaded to the adaptor. From the Devolo's ftp update site [61], we downloaded a `.deb` file, that contains a series of files. One of these files is the firmware for the adaptor (a `.dvl` file), whose structure is visually represented in Table 3.1. This file is started with the Devolo magic number - in Unix systems, a magic number is a numerical or text value that identifies a file format or protocol - followed by six sections. Each section begins with its length, followed by its Four Character Code (FourCC - in the table represented as *4CC*), which is a sequence of four bytes used to uniquely identify data formats, and terminated by a Cyclic Redundancy Check (CRC). We used the FourCC codes to identify the sections present in the update.

- **File system type (FST):** The type of file system present in the image. In our case it is SquashFS, which is an lzma compressed read-only file system used in embedded systems.
- **Device type (DT):** The type of adaptor to which this update is meant for. The type is defined by the code name given by Devolo to its products. In our case, the type is “norwich”.
- **Original Equipment Manufacturer (OEM) variant:** We assume this field is meant for variants of this product. In our case, this field comes empty.
- **Kernel:** A minimalist linux kernel used for embedded systems. Our adaptor has a uImage Debian kernel, version 2.6.31. A uImage is a kernel image file that has a U-Boot wrapper that includes the OS type and loader information. The Das U-Boot [62] (Universal Bootloader) is an open source, primary boot loader used in embedded devices to package the instructions to boot the device’s operating system kernel.
- **File system (FS):** The actual file system included in the update. This file system will be decompressed and will replace the one in the adaptor.
- **Version:** The version of the firmware update in the following format: v2.3.5_2014-08-25_0025, where v2.3.5 is the firmware version, 2014-08-25 is the release date, and 0025 is the Subversion revision. Subversion is a version control system, that can track changes in projects [63].

Devolo magic number		FST length	FST 4CC	**FST**		FST CRC
DT length	DT 4CC	**DT**		DT CRC	OEM length	OEM 4CC
OEM CRC	Kernel length	Kernel 4CC	**Kernel**			
Kernel						
Kernel CRC	FS length	FS 4CC	**FS**			
FS						
FS						
FS CRC	Version length	Version 4CC	**Version**			Version CRC

Table 3.1: Representation of the firmware update structure.

Instead of using the GNU implementation of the C standard library (glibc) as is usual on Linux, the adaptor has the *μClibC* [64], version 0.9.30. Since the amount of persistent memory is only 32MB, the adaptor only stores the bare minimum files. While *glibc* is intended to fully support all relevant C standards across a wide range of hardware and kernel platforms, *μClibC* focuses on reducing space usage. *μClibC* is much smaller than *glibc* by enabling or disabling features according to space requirements.

The dLAN 500 WiFi offers a web interface for management tasks, such as password protecting the adaptor, checking the adaptor's status, or updating its firmware. The web interface is supported by a thttpd [65] server, which is a minimalist http server used for embedded devices. This server has its root in */var/www/*.

The adaptor has six network interfaces, `ath0`, `br0`, `eth0`, `eth1`, `lo` and `wifi0`. Each interface has its own MAC address.

- `ath0` is the virtual interface created by Atheros chips for `wifi0`;
- `br0` is the Ethernet bridging interface [57];
- `eth0` is the interface where the adaptor receives packets from the device connected to the Ethernet plug;
- `eth1` is used to exchange with other adaptors the packets received from `eth0`;
- `lo` is the loopback interface;
- `wifi0` is the interface for wi-fi connections.

Chapter 4

Implementation and evaluation

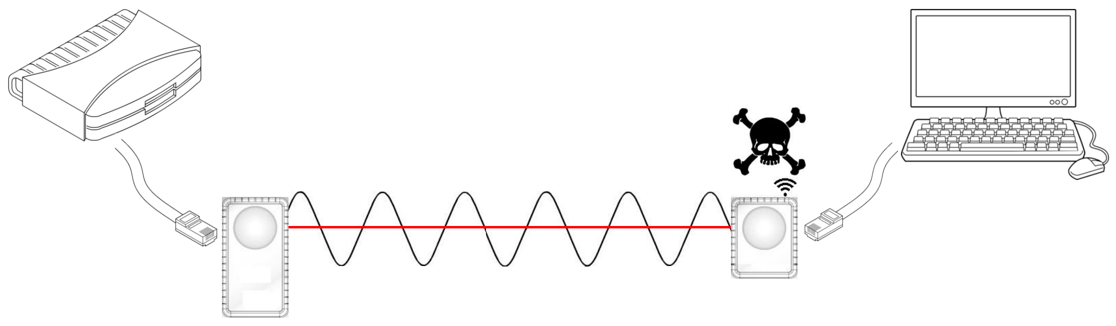
This chapter presents the actual changes made to our adaptor, how they were performed, and the results obtained. Here we provide a full explanation on how to place malicious firmware in the adaptor by exploiting the update mechanism, in order to obtain remote root access. This led us to a series of possible attacks with powerline adaptors. We also explain the tracing of the execution of the UKE protocol, followed by a description on how to cross-compile binaries, kernel objects and the kernel/bootloader for the adaptor. Each section contains the reasoning behind each step of this work and a summary of the major achievements. But first, we begin by presenting the experimental setup.

4.1 Experimental setup

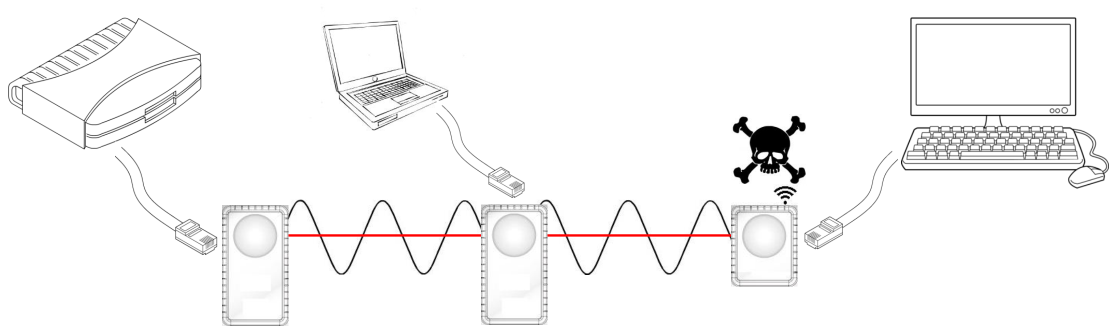
Our setup is composed of:

- **Dell Optiplex-380:** The *x86* computer where most of this work was done, including accessing to the adaptor and cross-compiling. This computer is running Elementary OS version Luna, which is based on the Ubuntu kernel version 12.04.
- **Devolo dLAN 500 WiFi:** This is the HomePlug powerline adaptor that was modified and used as an attack vector. The original firmware version was 3.1.0.
- **D-Link DHP-P309AV, TP-Link TL-PA4020PKIT and Devolo dLAN 500 duo+:** For the D-Link the firmware version is 3.01, and 1.1 for the TP-Link. The Devolo does not indicate its firmware version. These adaptors were used as victims, and no changes were made on them.

We have two setup configurations for our experiments (see Figure 4.1). Our adaptor is the Wi-Fi adaptor marked with the crossbones. In Setup 1, our adaptor is used as an active part of the network, while in Setup 2 our adaptor is passively listening to the network established by the other two adaptors. All adaptors have a computer connected to inject communications. In Setup 2 we connected a laptop to a router through the victim adaptors, while our malicious adaptor and computer are in a separate network.



(a) Setup 1



(b) Setup 2

Figure 4.1: Representation of our experimental setups.

4.2 Modifying the Devolo dLAN 500 WiFi

After downloading and analysing the firmware of a Devolo dLAN 500 WiFi, we found out it could be modified. Since it runs Linux, we could change it to get full access to the adaptor. We explain in the next subsections how to update the adaptor with a malicious firmware.

4.2.1 Extract firmware

As mentioned in Section 3.2.1, the updates for the Devolo adaptor come in a `.dvl` file, which includes a linux kernel and file system. To extract the various components of the `.dvl` file, we used a tool called Firmware Modification Kit (FMK) [66]. FMK has a script *extract-firmware.sh* that separates the file system from the various other components of the update. FMK's *extract-firmware.sh* extracts components identified by the bash tool `binwalk`. Although `binwalk` does not recognize the six sections of the `.dvl` file, it obtains the file system, which suffices for our purposes. In alternative, we could also manually extract the sections using the `dd` command.

4.2.2 Modifying/adding files

Now that we have the file system, we can modify it. We can freely edit text files or scripts, including shell scripts. One valuable target to change is the *rcS* script, present in */etc/init.d/*. The *rcS* is executed by the kernel at root level once the file system is mounted [67].

Adding executables can also be done. However, recall that our adaptor runs in a *MIPS* architecture, while most personal computers run a *x86* architecture. This means that executables compiled for *x86* do not run correctly in the Devolo adaptor. Therefore, extra effort is required to add executables to our adaptor, since our machines are *x86*.

4.2.3 Rebuilding the firmware

After all the changes have been done to our adaptor's firmware, we must rebuild the update image to then feed it to the adaptor. To achieve this, we use FMK again, more specifically the *build-firmware.sh* script. This script re-unites the image parts and updates internal checksums (for example, the file system includes a checksum). If files are added to the file system and it becomes larger than the original size, it is necessary to use the `-min` option on the rebuild script. Even if with this option sometimes the new file system is still larger than the original, and in this case the rebuild script refuses to create the image. This protection exists because, as said on the script, "*Building firmware images larger than the original can brick your device!*". ("Bricking a device" is a slang term for leaving a device unusable). Since this is a bash script, we commented the section that prevents the image building. However, the creators of FMK are correct, and attention to the size of the new image must be paid. If we feed to the adaptor an image larger than what it supports, it will likely break the update process and leave the adaptor permanently unusable.

4.2.4 Bypassing the security

As mentioned in Section 3.2.1, the firmware update contains six sections. Each section begins with its length and ends with a 128-bit Cyclic Redundancy Check (CRC) [68]. Any changes to any of the sections of the firmware update imply modifications to its length and CRC. This can be done using a binary edition program, like *ghex* [69], where we can change directly the octets present in a file. Furthermore, the sections *file system type*, *OEM variant*, and *version* are encrypted using XOR keys, one for each section. The keys were extracted from the binary executable `chunk` in the folder */usr/bin/* of our adaptor's file system using the IDA program [70]. The CRC is updated after the encryption of these sections.

The remaining step is to change the version of the firmware update. When the adaptor receives an update it checks the received firmware version. As long as the version is different from the one currently installed, the adaptor accepts it. The version is also a section, so after modifying the version its CRC must also be updated.

4.2.5 Updating the target adaptor

Updating our adaptor can be done in one of two ways: through its web interface or through a program (`faqfwupdate`) provided in the downloaded `.deb` (not the `.dvl` file). Through the web interface we just have to navigate to the update firmware menu (Main Page → Management → Update Firmware) and feed it the update file. The web update sequence can be seen on Appendix B.

The `faqfwupdate` program is a command line executable, and the command to run it is in the same folder, in the file *commandline_update.txt*. To use the program, the update must be located in the same folder as the program, and with the same name as the original update. We find the web interface easier to use, but found no noteworthy advantages or disadvantages of each method.

4.3 Obtaining access to the adaptor

In this section we explain how to change the adaptor to access it remotely. We analysed the files of our adaptor's file system, in order to understand where we could make changes that would allow us root access. A starting point is the *shadow* file, where passwords are kept. No changes were required because, as we can see in the following excerpt, the user root has no defined password:

```
root::10933:0:99999:7:::
```

A password for a user in the *shadow* file should appear between the first and second colons. If there is no password defined, that field is empty (as is in our case). To have remote access to the adaptor we use the already present `telnet` program [58]. The `telnet` daemon is started with the *rcS*, but is only available when the adaptor is booting. We removed the encasing condition, making the `telnet` daemon always available.

In addition, the adaptor is protected through `iptables` [71], configured to block `telnet`'s default port (23), amongst others. So, to open the port 23 of our adaptor (so we can have `telnet` connections to it), in the end of *rcS* we added the following code to insert a new rule in `iptables`:

```
/sbin/iptables -A INPUT -m state --state NEW -m tcp \
-p tcp --dport 23 -j ACCEPT
```

To update our adaptor we need its IP address if we want to use the web interface. The easiest way to discover our adaptor's IP address is to use the `faqfwupdate` program (mentioned in Section 4.2.5), but using the command in the *commandline_check.txt* file.

The `faqfwupdate` sends *who-is-there* broadcasts and identify the Devolo adaptors with IP addresses in the network. Thus, using the terminal we can easily get the adaptor's IP address to access the adaptor's web page:

```
$ pwd
(...)/devolo-firmware-dlan500-wifi_3.1.0-1_i386/firmware/
devolo-firmware-dlan500-wifi
$ rm update.log && ./faqfwupdate /onlycheck
/parsableprogress /logfile:update.log ; grep -m 1 http \
update.log
##FIRMWARE_UPDATE_STATUS## 1 0 100 0 f4:06:8d:0d:af:d9 pen-
ding
[faq] received hello on random port: <dLAN 500 WiFi>
<f4:06:8d:0d:af:d9> <10.10.104.98> <255.255.0.0>
<http://10.10.104.98/cgi-bin/htmlmgr?_file=sysinfo>
<0017eb29ad> -> OK
$
```

In this example the adaptor's IP address is `10.10.104.98`.

With these changes we successfully modified a firmware update to our adaptor and successfully updated the adaptor with it, granting us remote root access to it (using `telnet`). This is a major step, since it allows us to run custom programs directly on the adaptor providing us access to the powerline from the adaptor's perspective - which is the most realistic attack scenario. By having remote access to our adaptor we answer positively research question *Q3.1*.

Through modifications in the *rcS* script we also had access to some important information. The `sysmgrd` program (started in the *rcS*) manages the majority of the system, such as handling calls like the button push (as can be seen in Section 4.6). During the adaptor's boot process, the `sysmgrd` reads a file in */etc/devolo/* called *defaults.xml*, which contains the default settings of the adaptor. That file is processed and a new file is created in */tmp/*, called *config*, that contains the current configuration of the adaptor, including the Wi-Fi Protected Access (WPA) key for Wi-Fi access, DAK and NMK. This file is deleted when the boot process is complete. So, during boot we copy it to the root of the http server running on the adaptor (and named it *def2.xml*), adding the following code to the end of *rcS*:

```
cat /tmp/config > /var/www/def2.xml
```

By accessing the adaptor through its web interface we can read this file remotely, typing in a browser `<adaptor's IP address>/def2.xml`. Obtaining these keys is itself a security breach and a contribution, although not the main objective of this work. Nevertheless, stealing the NMK and accessing the adaptor open new attack possibilities:

- An update like the one described above is carried out in a most seamless way. There is only a brief period during which the adaptor is unavailable, which is during its boot process. A user will experience only a short network interruption (about 15 seconds), which will most likely pass unnoticed. Even if the user notices the network disruption, its cause cannot be directly associated with a malicious firmware update.

- Using this update an attacker can steal the NMK and place new adaptors in the network without notice.
- By stealing the WPA key it is possible to connect other devices with Wi-Fi transceivers to the network.
- A powerline adaptor is a new place to attack a network or its elements. By remotely accessing one, an attacker can to listen to the network, inject packets, etc.
- As explained in Section 2.4.2, the adaptor mediates all traffic to and from the connected devices. Since it is a routing point, a malicious adaptor can redirect all the traffic of the connected devices to another computer, where it could be all listened, spoofed, modified, etc.
- By updating the adaptor it is possible to change its NMK (by changing the *defaults.xml* file). This means an attacker can update adaptors to change the network configuration. This is a powerful capability - it allows to isolate devices from the underlying network, create network partitions, network bottlenecks, etc. Isolated devices can be attacked without the detection of the remaining devices [72], and network partitions are a challenge for distributed systems [73].

4.4 Execution flow of the UKE protocol on our adaptor

In this section we present our analysis of the UKE protocol running in our adaptor. In order to understand if the HomePlug messages reach kernel/user space, we started by running the already included `tcpdump` program [74]. Since `tcpdump` is running directly on the adaptor, it should provide us with the packets passing through the powerline with no restrictions (as mentioned in Section 2.4.2). Initially, we ran `tcpdump` with no options on all interfaces. This provided the expected results - a huge amount of IP and IPv6 traffic originated from our computer. It shows that `tcpdump` is running correctly, but it is hard to analyse a huge amount of traffic. So, we added filters. First, we restricted to packets with the *ethertype* `0x88e1`, which is the *ethertype* of HomePlug messages, using the following options:

```
# tcpdump -XX -e -i any ether proto 0x88e1
```

The `-XX`, `-e` switches are for making `tcpdump` more verbose and to show packet headers. Unfortunately, this did not provide the messages meant to go over the powerline only. We experimented placing each interface in promiscuous mode using `ifconfig`, running `tcpdump` and then triggering the UKE protocol. Unfortunately, we still got no UKE packets or HomePlug messages. We also repeated this test on each interface but using less restrictive options - just filtering IP packets:

```
# tcpdump -XX -e -vv -i <iface> not ip and not ip6
```

In this case we use the `-vv` option for making `tcpdump` even more verbose - the `-vv` is not available when listening to all interfaces at the same time (with the “any” interface). We tried with both our setups, using our malicious adaptor as part of the protocol and only as a Sniffer, but the results were the same - we did not read the UKE packets. No new packets were captured - the results were the same as running wireshark in our computer.

Since `tcpdump` provided no results, we traced the execution of the UKE on our adaptor. When the button is pushed to start the UKE protocol, the `sysmgrd` starts another program, the `simpleconnectd`. By reading the kernel logs we saw that when the `simpleconnectd` is started the `sysmgrd` opens a new network interface, `ath1`. The technical descriptions we found say that Atheros chips use `ath`*X* interfaces as virtual nodes, in place of directly accessing the `wifi0` interface. We find odd that a virtual node for Wi-Fi is created when we start the UKE, so we ran `tcpdump` directly on `ath1`:

```
# tcpdump -XX -e -vv -i ath1
```

`tcpdump` prints three IPv6 packets shortly after the button is pressed, and produces no more results until the interface is closed when the UKE ends. These three packets are always the same even in different executions of the UKE, and show no relevant information - we thought they could be parts of legitimate UKE packets, but they would have to possess different parts between executions. Also, none of the packets show the static parts of the UKE messages, such as the correct *ethertype*, or the management message type. Also, wireshark running in our computer does not capture these three IPv6 packets. We do not know why this happens or what the meaning of those packets is.

We did not get UKE packets on `ath1`, but the command `ifconfig` shows that the interface is being used to send packets - the transmission number (*tx*) and byte count increase while the `simpleconnectd` program is executing. We did not find which program is sending those packets.

We explored the `simpleconnectd`’s execution using the `strace` command - we get the `simpleconnectd`’s `pid` using the `ps` command, then feed the `pid` to `strace` with the `-p` option. Using the `strace` we observe that the `simpleconnectd` keeps trying to read packets from the kernel using the *recvfrom* function in a network socket. When we push the button on the second adaptor to start the pairing, that adaptor also sends UKE packets. Shortly after the second adaptor starts the UKE, `simpleconnectd` writes to log “[SimpleConnect#AJoinerHasBeenSuccessfullyAdded]”. We assume this message indicates a “join confirm” was received. Then, `simpleconnectd` writes to log “[SimpleConnect#EventEnd]”, which we correlate with the end of the UKE. `strace` shows the packets read using *recvfrom*, but none of the packets shown are part of the UKE protocol. However, some information must reach the user level, or else we would not find an application managing the UKE. Our conclusion is that the received HomePlug packets are hidden from us, as well as those sent - we did not find the process that is sending the UKE messages. We also used `strace` on the `sysmgrd`, but the tracing did not reveal any connections between this program and the networking system or the UKE.

Other files present in the adaptor have functions mentioning the NMK or NEK, such as `sysmgrd`, `dlancontrol`, *libdlan.so*, or *libdevolo.so* (seen using the `strings` command[1]). This reinforces the idea that the HomePlug messages (or at least some information) reaches user level but are in some way filtered and passed only to specific programs developed by Devolo.

After observing the behaviour of the adaptor during the UKE, we see that the HomePlug packets are in some way covert. Our next step is to build a program that triggers the UKE. But first, we update the `tcpdump` program included in our adaptor.

4.5 Running binaries on the adaptor

The adaptor contained an older version of `tcpdump`. After realizing that `tcpdump` was not capturing the packets, we decided to update it to the latest version at the time (*libpcap* 1.7.2 and `tcpdump` 4.7.3). This requires compiling the latest version of `tcpdump` and place it on our adaptor. This section explains the mechanisms for cross-compiling [26, 27], the binaries that were produced and the results that were obtained.

4.5.1 Cross-compiling

We cannot just place compiled binaries in the adaptor, since it possesses a different CPU architecture from our computer (*x86* in our computer, *MIPS* in our adaptor). To overcome the different architectures, we need to cross-compile [26, 27]. A cross-compiler is capable of compiling on a computer with a certain CPU architecture a program for another CPU architecture. In our case, we use a cross-compiler to compile programs on a *x86* CPU for a *MIPS* CPU.

We used the Buildroot *μClibC* cross-compiler toolchain [75]. This toolchain was selected because it uses the micro C library *μClibC* instead of the standard C library, just as our adaptor does. Also, we used an old version (2009.11), which uses the same *μClibC* version (0.9.30), GNU Compiler Collection (GCC) version (4.3.3) and kernel headers (2.6.31) as our adaptor. Once the cross-compiler is set, we can use it to generate executables for our adaptor.

[1]Prints the strings of printable characters contained in files.

To use the toolchain, we replace the native GCC compiler calls with calls to the toolchain's GCC, while adding a few extra parameters - specially the target architecture we want to cross-compile to. Since this work was developed in Linux, we added the toolchain GCC to the system's path for ease of use. Therefore, we can call the toolchain's GCC on the terminal by typing `$ mips-linux-uclibc-gcc`. Table 4.1 presents examples of equivalent native and toolchain GCC calls.

	Native GCC
1	`gcc example.c -o example` $< flags >$
2	`./configure --with-shared --prefix=/usr`
3	`make default V=1`

	Toolchain GCC
1	`mips-linux-uclibc-gcc example.c -o example` $< flags >$
2	`CC=mips-linux-uclibc-gcc ./configure --with-shared --prefix=/usr --target=mips`
3	`make ARCH=mips CROSS_COMPILE=mips-linux-uclibc- default V=1`

Table 4.1: Equivalent native and *MIPS* cross-compile toolchain GCC calls.

Even with the toolchain set up, we must be careful when placing executables in the adaptor since library issues may arise. Even though *μClibC* contains the standard C library, other libraries may be needed and will have to be added manually, because the adaptor only contains the libraries it needs. This means obtaining the libraries compiled to *MIPS* or compiling the libraries to *MIPS*, which may not be trivial (or feasible since the source code is required). Furthermore, to correctly run an executable, the libraries used to compile and, later on, dynamically linked during runtime must be the same [76]. To avoid this issue we can create static executables that contain in themselves all the libraries and binaries they need to run (no linking is done in runtime - only in compile time). The advantage of static executables is that they do not depend on the libraries in the device, thus avoiding library issues. The disadvantage is that these binaries are much larger than their dynamically linked equivalents.

After the executable is compiled, we can test it. However, the executable only runs correctly on a *MIPS* processor, and, following our case, it was generated on a *x86* machine. So, to run generated executables for *MIPS* or other architectures on a *x86* machine, we need an emulator. The Quick Emulator (QEMU) [77] has such capabilities as it can be used as a full virtual machine, or just as a runtime emulator. After installing QEMU, on the command line we can call `$ qemu-mips` $< args >$ (or `$ qemu-`$< platform >< args >$ for other CPU architectures) and give as argument our *MIPS* executable. System dependent features (such as networking) are not guaranteed to run correctly, but for simple programs it suffices.

4.5.2 Cross-compiling `tcpdump`

`tcpdump` depends on *libpcap*. Consequently, to compile `tcpdump` both *libpcap* and `tcpdump` sources must be in the same folder as in the following example:

```
$ tree -L 1
.
|-- libpcap-1.7.2
|-- tcpdump-4.7.3
$
```

First, we cross-compiled the *lipcap* by using the following command on the *libpcap* folder:

```
$ CC=mips-linux-uclibc-gcc ./configure --host=mips \
--with-pcap=linux && make
```

resulting in a *libpcap.a* compiled for *MIPS*. Then, we moved to the `tcpdump` folder and used the same command. The final result was a fully functional updated `tcpdump` successfully cross-compiled for *MIPS*.

4.5.3 Cross-compiled binaries results

Following these steps we successfully placed cross-compiled binaries for *MIPS* on our adaptor. Some binaries were compiled using the *static* GCC directive, others using the adaptor's libraries. Amongst the binaries placed in the adaptor we emphasize the latest version of `tcpdump` and the program *startUKE*, presented below.

Updated `tcpdump`

We used our newly cross-compiled `tcpdump` on our adaptor in an attempt to obtain the HomePlug management messages. We re-ran all mentioned tests, but got no new results.

Updated `strace`

As with `tcpdump`, we tried to update `strace` to the latest version since it could provide new results. The version in the adaptor was the 4.5.20, while the latest release at the time was the 4.10. However, there were incompatibilities with the *MIPS* system since it does not support the large file system (64-bit pointers to files - *MIPS* only supports 32-bit pointers), and the cross-compiling was unsuccessful.

Triggering the UKE protocol

The *startUKE* program was made by us in an attempt to trigger the UKE protocol. The idea was that by triggering the UKE we could see the remaining packets. Since the standard specifies that the UKE packets can be transmitted over the MAC layer, we hoped that triggering the protocol by sending packets over the MAC layer, the remaining packets would also be sent over the MAC, where we could read them with `tcpdump`. We used the example packets of the HomePlug documentation for the UKE protocol, in Annex J (presented succinctly in Appendix A), and followed the standard's description of the *CM_SC_JOIN.request/confirm* messages. As mentioned in Section 3.1, the "join" messages are not supposed to be visible through the Ethernet, which could ruin this test since we are sending these messages through the Ethernet - the only interface we have access to. In this test we take the place of adaptor 1 (so we are in place of sending the first UKE message).

We successfully built the program and used it on the adaptor, and it successfully sends messages to the other powerline adaptor, as depicted in Setup 1 (Section 4.1, Figure 4.1). The program starts by sending *CM_SC_JOIN.request* messages, every second. We captured these messages with `tcpdump`. Then, we pushed the button of the second adaptor, to make it enter *simple connect* mode and also send *CM_SC_JOIN.request* messages. We should get a *CM_SC_JOIN.confirm*, but we did not capture it. Nevertheless, we continued our attempt. Then, the program sent the first UKE message, the *CM_GET_KEY.request*, and we captured it on `tcpdump`. The other powerline adaptor always answered our message with the correct following UKE message (*CM_GET_KEY.confirm* - also captured on `tcpdump`) but using the *deny request* code, thus terminating the UKE.

Pressing the button manually in the second adaptor is not a completely realistic scenario for a network infiltration, since physical access to the adaptor is not assumed. Nevertheless, it could be used to demonstrate that the UKE is vulnerable to a sniffing attack. In addition, the push button action can be emulated using the open-plc's programs `int6k`, `plctool` or `amptool`, with the `-B` switch. These programs provide a push button message that can be send to the adaptor's chip, having the same effect as the physical button push. The different programs are compatible with the different versions of the Atheros' chips. Note that adaptor is not required to answer the push button message - the remote push button capability is configured in the `.pib`.

The main reason we found for the UKE being cancelled by the other adaptor is that we may have never received the *CM_SC_JOIN.confirm*, which is a requirement for the UKE. Triggering the UKE without using the "join" messages produced the same results. There are other possibilities for our failure. The format of our packet could be wrong, or some fields present in the message could be outdated or incoherent with the current state of the network. For example, one of the fields is the *protocol run number*. If this value is kept by the adaptors and is sequential, we have no way of knowing which value we should place in this field - we would require a correct message to extrapolate what would be the next sequence number. Making the *protocol run number* sequential can avoid/difficult replay attacks - it is a reasonable assumption.

Since we do not have access to HomePlug messages, this leads us to one of two conclusions:

(1) HomePlug messages remain in the chip, never reaching the kernel/user space;

(2) HomePlug messages reach the kernel/user space but are in some way hidden.

Case (1) means that our only options is to put the chip in some sort of debug mode, hoping that the debugging would be seen in user space. Case (2) implies that we change some parts of the kernel or drivers, in order to remove the protection that is hiding from us the HomePlug messages. This protection is most likely at the level of the physical/Ethernet drivers. Either option implies cross-compiling drivers (kernel objects), since (1) requires direct access to the chip (which is made through drivers).

A third option is to make direct calls to the *athrs_gmac.ko* driver using the `syscall` api. Using `syscall` directly on a kernel module is hard, as it implies knowing offsets dependent of the system architecture, amongst other features. Due to time constraints we did not test this option.

4.6 Adding drivers

Following our conclusions, we needed to place a driver on the adaptor. The kernel has access to the chip through specific drivers. By modifying the drivers in the adaptor or by adding our own drivers, we could access the chip and then forward the HomePlug messages to the user level. This section provides the basics on the Linux kernel and it's drivers, and how to cross-compile them.

4.6.1 Kernel, kernel objects, drivers, and physical/Ethernet drivers

A kernel is a computer program that manages input/output requests from software, and translates them into instructions for the CPU or other hardware components. This is an abstraction layer for software, so that programs do not have to be designed for specific hardware parts. A kernel is composed of a multitude of small binary objects. These objects offer functionalities, and software running on the kernel can easily make calls to them through simple includes.

A kernel object is an external binary compliant with the kernel that can be loaded and unloaded into it during runtime. They extend the functionality of the kernel without the need to reboot the system. An example is loading a device driver to the kernel, allowing it to access new hardware connected to the system.

Drivers "*are distinct* black boxes *that make a particular piece of hardware respond to a well-defined internal programming interface; they hide completely the details of how the device works. User activities are performed by means of a set of standardized calls that are independent of the specific driver; mapping those calls to device-specific operations* *that act on real hardware is then the role of the device driver. This programming interface is such that drivers can be built separately from the rest of the kernel and* plugged in *at runtime when needed.*" [78].

The networking system is composed of two parts, a physical driver and an Ethernet driver. The physical driver handles the connection to the network hardware, while the Ethernet driver uses the physical drivers' API to effectively establish communication - send/receive packets, transmission/reception rates, amongst other components.

4.6.2 Cross-compiling drivers

A driver external to the kernel is contained in a kernel object (`.ko` file). To compile a driver, the kernel where the driver will be inserted is necessary (since a kernel object must be compliant with the kernel where it will be inserted), as well as a correct Makefile [79, 78]. Since the Linux kernel is open source, Devolo is legally bound to provide the source code of the kernel they used in the adaptor. So, we downloaded our adaptor's code bundle [80], which includes all open source code used in our adaptor.

Simply having the kernel's source code is not enough. In order to use the kernel to compile a driver, we must configure and prepare it. To do so, we used the configuration file present in the bundle and our toolchain, and ran the following commands in the downloaded kernel folder:

```
$ make ARCH=mips CROSS_COMPILE=mips-linux-uclibc- oldconfig

$ make ARCH=mips CROSS_COMPILE=mips-linux-uclibc- prepare
```

With this the kernel is prepared, and now we can compile the driver.

In Appendix C, we provide an example of a simple Makefile used to compile a driver for the native kernel, and its equivalent for cross-compiling to *MIPS* (Appendices C.1 and C.2). Note that the only difference between a Makefile for native compiling and for cross-compiling is that the kernel paths differ: for native compiling we point to the kernel running in our *x86* machine, while for cross-compiling we point to kernel of our adaptor, which we previously prepared. To cross-compile a driver, we run the following command:

```
$ make ARCH=mips CROSS_COMPILE=mips-linux-uclibc- default
```

The result is a `.ko` (kernel object) file, cross-compiled for *MIPS*, compliant with our adaptor's kernel. A kernel object can be loaded using the system's `insmod` command, and removed using `rmmod`. Loaded kernel objects can be listed using `lsmod`.

4.6.3 Cross-compiling *dvlbutton*

To test if we could make a functional driver, we started with a very simple driver that does nothing but print a message to the kernel when it is loaded and removed. We did this successfully. The next step was to compile a more complex driver. We decided to modify the driver that handles the button pressing of the adaptor. If something went wrong with the driver, the adaptor would still be functional and we would just have to replace the damaged driver with the original one.

The code of this driver (called *dvlbutton*) is open source and is included in our adaptor's code bundle. By analysing the source code we saw that a function is called when a button is pressed, and one when a button is released. There is the necessity for both functions, since the button pressing has different actions depending on the duration. Our modification is simple, but acts as a proof of concept: write to a log a debug message when a button is pressed, and another when a button is released. Our adaptor has a limited offer of programs, including logging programs. So, we used the kernel logging (function `printk`), which is available. The calls were made with the `KERN_DEBUG` parameter, which is the lowest loglevel available [81] - we used it since we were doing simple debug.

To do this test, we modified the source code of *dvlbutton*, adding the following code to *dvlbutton.c* in the function *shortlong_stop_handler*:

```
printk(KERN_DEBUG "[hack] a button has been released\n");
```

and the following code to the same file, in function *shortlong_start_handler*:

```
printk(KERN_DEBUG "[hack] a button has been pressed\n");
```

This completed all the changes we needed.

The next step was to cross-compile the driver using the Makefile in Appendix C.3. This Makefile is similar to the one in Appendix C.2, but we add the files upon which *dvlbutton.c* depends on line 3. Then we compiled the driver using the command of Section 4.6.2.

To test our modified driver, we must place it in the adaptor. We replaced the default driver with our modified driver the extracted file system, and updated our adaptor following the steps detailed in Section 4.3. Lastly, we had to confirm that our driver worked. To do this, we logged in on our adaptor using `telnet`, and did a short press on the button. The, we used the `logread` command, which reads kernel logs among others. With it, we could see the following lines:

```
dlanwireless login: root
# logread
(...)
Jan  7 04:24:42 dlanwireless user.debug kernel: [hack] a
button has been pressed
Jan  7 04:24:42 dlanwireless user.debug kernel: [hack] a
button has been released
(...)
Jan  7 04:24:42 dlanwireless daemon.info sysmgrd[96]:
SvcMgrLedsAndButtons: dLAN button short press
```

```
(...)
#
```

As we can see in this log excerpt, our debug messages are printed and the `sysmgrd` is called because the short press happened. This proves that our change was successful, and reinforces that we positively answer *Q3.1*.

4.6.4 Cross-compiling a network driver

As can be seen in the *rcS* script of our adaptor, the `eth0` and `eth1` interfaces are initialized when the kernel object *athrs_gmac.ko* is loaded. We did a simple experiment to confirm that this is the driver responsible for networking. We removed the driver using the `rmmod` command, and not only the `telnet` crashed, the internet connection of our computer connected to the adaptor was disrupted. This confirms this is the driver responsible for networking.

The source code of this driver is unavailable. This, plus the object being external to the kernel, leads us to believe that the HomePlug messages are filtered in the driver, and that is why they are not seen in the `tcpdump`. We analysed this driver using the `strings` command, and the functions listed are the functions expected of a physical driver (e.g., setting physical channels). However, there is also a function called *athr_gmac_recv_packets*. This function seems out of place, since it is the Ethernet driver that handles the transmission/reception of packets. In fact, the `strings` command reveals that this driver seems to be a mixture of both physical and Ethernet drivers, which strikes us as something complex and unusual - usually each layer is managed by a separate driver.

We needed to modify the source code of the *athrs_gmac.ko*, or to make a new driver. Making a physical driver requires knowledge about the underlying chip - register configurations, register codes, amongst other data. Writing such driver would require a large amount of time since we do not have experience in such task. Obtaining the source code of a physical driver is critical for this step. However, Atheros does not publish the source code of the drivers for their chips, which is a major setback. We did not find the source code of a simple driver, but we did find the source code of a driver for our adaptor's chip [82] using the Distributed Switch Architecture (DSA) [83]. This is a more complex driver, and requires further work to compile and run correctly, since it depends on objects not present in our adaptor's kernel.

A DSA driver has more dependencies, since it depends on physical drivers specific for distributed switching. These drivers are not compiled in the kernel included in our adaptor, but their presence is predicted - the source code of the DSA dependencies is in our kernel's source. So, we copied the source code of the physical network drivers and joined them to the source code of the DSA driver. Since these files are originally kernel modules, we had to strip the parts of the code associated with registering the drivers.

After that, we can feed those files as dependencies of the DSA driver. To compile the DSA driver, we used the Makefile of Appendix C.4. In line 3 we can see the physical DSA files as the DSA driver's dependencies. The driver compiled without errors, so now we can test it.

4.6.5 Testing DSA driver

We updated the adaptor to include the DSA driver, so we could insert it manually. We use `telnet` to access the adaptor and use the `insmod` command to insert the driver into the kernel. This step executed successfully. Next, we re-ran the `tcpdump` tests described in Section 4.4 to see if there were any changes in the received packets - we did not capture any new packets. So, we removed the *athrs_gmac.ko* from the kernel, using `rmmod`. We got the same results as with the first test of removing the *athrs_gmac.ko* - `telnet` crashed and no internet connection on our computer. There are two possibilities for this:

(1) The DSA driver does not properly replace the *athrs_gmac.ko* driver;

(2) The network driver always has to be loaded during the adaptor's boot.

If it is option (1), we reached a dead end, since we have no other driver to replace the *athrs_gmac.ko*. To validate option (2), we did a simple test: remove and insert the *athrs_gmac.ko* in one command with the adaptor running, by typing:

```
# rmmod athrs_gmac && \
insmod /lib/modules/2.6.31/net/athrs_gmac.ko
```

If the network driver can be inserted after the adaptor's boot, this should maintain the networking of the adaptor. This test produced the same result as before - no networking - which means that the network driver can only be loaded during boot time. Option (2) means we can continue this work by modifying the *rcS*, replacing the loading of *athrs_gmac.ko* with our DSA driver. This, however, presents a serious risk. Our previous removals of the *athrs_gmac.ko* were solved by restarting the adaptor. Changing the loaded driver during boot is not so simple to solve. If the DSA driver is not a correct replacement for *athrs_gmac.ko*, the most likely scenario is that the networking of the adaptor will not start, blocking us from accessing it permanently since the only connection we have to it is through the network. Leaving the adaptor permanently unusable means we would have to obtain a new one, and this adaptor costs about €70. Each time this test would go wrong, we would need a new adaptor. For costs reasons, we are very careful with changes that could leave the adaptor unusable, and this option has not been tested yet.

Nevertheless, we designed a safe solution to test if the DSA driver is a proper replacement for the *athrs_gmac.ko*. By default, the adaptor will boot with the default driver. The main point of this solution is to use the *rcS* to boot the adaptor with the DSA driver once and only once. We use a “flag” file to indicate what driver to load - if the “flag” file exists, the adaptor loads the default driver.

When booting with the DSA we always reboot the adaptor for safety, which means that it will boot twice. In the first boot we load the DSA driver, and in the second one we load the default driver. We mark the first boot by creating the “flag”. Since errors can occur during the load of the DSA driver or during the remainder of the *rcS*, we create the “flag” and set the adaptor to reboot before loading it. The flag file is created in */var/www* because we are sure we can create files in that directory - recall that a SquashFS is read-only, but */var* is mounted in a read-write partition.

Following the previous description, instead of just loading the *athrs_gmac.ko* with:

```
insmod /lib/modules/2.6.31/net/athrs_gmac.ko
```

we replace this line with the following code:

```
if [ ! -e /var/www/flag.txt ]
then
        touch /var/www/flag.txt
        ./setReboot.sh &
        insmod /lib/modules/2.6.31/net/dsaDriver.ko
        ping -c 1 <ip>
else
        insmod /lib/modules/2.6.31/net/athrs_gmac.ko
fi
```

where $< ip >$ is the IP address of the computer depicted in our setups (Section 4.1, Figure 4.1), and *setReboot.sh* is:

```
sleep 100
reboot
```

The sleep has a duration of 100 seconds to ensure the adaptor has enough time to boot. The `ping` is used to test if the networking is functioning correctly - if it is, we should receive it in our computer.

This “flag” is our safeguard - if anything wrong happens, the next boot should be correct, since the *rcS* will run the exact same steps it did before our changes. Any issues associated with the driver replacement should be avoided, since unless we manually remove the “flag” the adaptor will boot with the default driver.

Although this test should be safe, we cannot guarantee that a major crash is impossible. Any mistakes will likely cause the adaptor to be permanently unusable. Due to this fact and time constraints, we did not test this solution.

4.7 Changing kernel/bootloader

We provide insights on possible modifications to kernel/bootloader and how to compile them in this section. The kernel has drivers that communicate directly with the chip. Modifying one of these drivers could mean placing the chip on debug mode, or accessing it directly. We have the source code, so we can modify these drivers. However, modifications to kernel elements mean compiling it and replacing the kernel of the adaptor with the new one.

Devolo provides a configuration file together with the source code of the kernel. This configuration should provide a compiled kernel equal to the one present in the adaptor. We can modify the objects that are in the kernel by modifying this configuration.

The kernel is built through the *KBuild* system, composed of *KConfig* files which define what kernel modules are built, and are generated after the kernel is configured [84, 85]. These files can be changed manually or through the *menuconfig* interface, called using the `$ make menuconfig` command on the kernel folder. We advise against changing these files manually due to the intricate kernel dependencies. The *menuconfig* interface is simple to use, has an inbuilt help feature and a search. If we want to add a specific module to the kernel, we can use the search to locate it in the *menuconfig*. The search also mentions the module's dependencies.

After the kernel configuration is done, we can build it. Features can be added to an already compiled kernel by reconfiguring and then building it. Only the new modules will be compiled, which are added to the kernel binary. To build the kernel, we run the following command on the kernel folder:

```
$ make ARCH=mips CROSS_COMPILE=mips-linux-uclibc-
```

After being compiled, the new kernel must be placed in the firmware update. Since FMK does not extract all the image parts for us, we must do it ourselves. We know where the kernel starts due to the FourCC identifying it, and its length is in the firmware update. So, using the `dd` command we can replace the kernel.

A process similar to modifying the kernel can be used to change the bootloader. The bootloader also has important features, since it manages the chip's configuration during the adaptor's boot process. To compile the bootloader, we must first configure it to the proper board. In our case, we ran

```
$ make CROSS_COMPILE=mips-linux-uclibc- ARCH=mips \
ap121-2.6.31-2MB_config
```

where *ap121-2.6.31-2MB_config* is the correct configuration for the board in our adaptor, since the hardware board of the adaptor has the codename *ap121*, the adaptor runs a kernel with version *2.6.31*, and *2MB* is the flash size of the board. After the configuration is done, we can compile it with the same command used to compile the kernel.

We successfully compiled both, but did not replace any of these on the firmware running on the adaptor. Our analysis of these drivers did not reveal a clear point where we could make the changes we need, but it is worth mentioning we could do it. Nevertheless, replacing the kernel/bootloader can also leave our adaptor unusable, since if any of those components does not work properly we would permanently lose access to the adaptor.

4.8 Discussion

Given our previous analysis of the HomePlug protocol (see Chapter 3), we discovered a design flaw in one of the key exchange mechanisms (positively answering research questions *Q1* and *Q2*). To expose the vulnerability, we need access to the HomePlug messages. Since an external computer does not have access to such messages, we need to place ourselves in the position of an adaptor. To this end we updated a HomePlug adaptor with malicious firmware, to provide us with remote root access to it - thus giving an affirmative answer to research question *Q3.1*.

Tracing the execution of the UKE protocol in our adaptor revealed that the HomePlug messages are in some way hidden from traffic analysers. It can be a mechanism in the kernel or it could be that the adaptor's chip handles the protocol by itself. We discard the last possibility since we discovered that some information about the HomePlug reaches user level - if the chip alone manages the protocol, why would information about it reach the kernel? A full separation of responsibilities between chip and kernel is possible, where the chip would be used as a simple network hardware, and the HomePlug management would remain in there.

Chasing the possibility of existence of HomePlug management at user level, we went further into the kernel trying to discover some protection mechanism. The network driver (called *athrs_gmac*) shows functions of both a physical and Ethernet driver. Such complex driver could be coincidence or a necessity of these chips, and yet we cannot provide a good reason for a lack of separation between physical and Ethernet layers. That, and our tests revealing packets being sent at a kernel level (transmission rate of the interface increasing on `ifconfig`) but hidden to traffic analysers leads us to believe that the packets are being hidden by the driver.

Changing the driver is the next step. However, the source code of *athrs_gmac* is unavailable, and writing a device driver is no easy task. Besides connecting the driver to the kernel properly, knowledge of the underlying hardware is required. We did find the source of a more complex driver for the same chip, which we successfully compiled. Loading the driver into our adaptor's kernel showed no errors, but that does not confirm that it replaces correctly the *athrs_gmac*. Since replacing *athrs_gmac* with an incorrect driver could lead to an unusable adaptor, we did not perform this test.

We were unable to practically prove the vulnerability. Reverse engineering such complex device is a time consuming task, and we were unable to perform the attack. To finish this work it is required to discover how the packets are being hidden (following our assumption that they reach the kernel). Tracing the driver is an option, but requires a kernel with driver tracing capabilities. Making direct calls to the driver seems to be another possible solution, but note that `syscall` is a very complex function. If the packets do not reach user level and remain in the chip, it is required some mechanism to place the chip in debug and reveal the HomePlug messages.

If any of these solutions work, we could see the HomePlug messages in our adaptor, whether if it is sniffing its own network or another one. In both our setups we would see the packets we require and prove the vulnerability.

Chapter 5

Conclusions

This work studied a powerline protocol in search for security vulnerabilities. We chose the HomePlug protocol due to ease of obtaining compliant devices. Our analysis revealed a flaw in one of the key provision mechanisms - the Unicast Key Exchange (UKE) protocol. This protocol is vulnerable to a sniffing attack, where an attacker who listens to the UKE packets can steal the critical network keys: the Network Membership Key (NMK) and the Network Encryption Key (NEK). After a successful attack on the UKE, an attacker has full access to the network.

To demonstrate the UKE flaw, we needed a powerline Sniffer. Since we do not have the skills to build such physical device, we hacked a Devolo dLAN 500 WiFi powerline adaptor. This adaptor runs Linux, and we used it as our attack vector on the network. Our expectations were that in the adaptor we could sniff all the powerline communications using a traffic analyser like `tcpdump`.

Our hacking was successful: by updating the adaptor with malicious firmware we obtained remote root access to the adaptor and stole its configurations, including some keys. Although this was not the objective of this work, updating an adaptor with malicious software opens a series of new possible network attacks. A powerline adaptor is in a privileged network position, mediating the connection between two points. By controlling such point an attacker could have full access to traffic generated by the devices connected to the adaptor.

The HomePlug protocol describes that most management messages are supposed to be hidden. In fact, a computer connected to a powerline adaptor has no access to the management messages. Our assumption was that by being in the adaptor's perspective we could see all HomePlug messages. We analysed the traffic passing through the adaptor using `tcpdump`, but it did not show the critical hidden packets.

We kept our attempt of obtaining the messages by further modifying the adaptor. We successfully cross-compiled binaries on a *x86* machine for *MIPS*. For example, we placed in the adaptor the latest version of `tcpdump`, and a program we devised in an attempt to trigger the UKE manually. All our experiments were unsuccessful, and we did not obtain the HomePlug messages. Nevertheless, our analysis reveals that at least some information about the HomePlug messages (if not the whole packets) reach user level on the adaptor.

We continued our work by modifying/adding kernel objects (drivers) to the adaptor. These were also successfully cross-compiled. We correctly cross-compiled a Distributed Switching Architecture driver for the adaptor, and it loaded correctly. However, our tests cannot confirm that the driver is a correct replacement for the existing network driver, which we believe that may be hiding the HomePlug messages from us. Nevertheless, these tests also did not provide the HomePlug messages. We also explain how to change the kernel/bootloader of our adaptor, but we did not execute this step since an improper kernel/bootloader will likely cause an unusable adaptor.

Summarizing, this work provides some insights on powerline communication. It focuses on the HomePlug protocol, which we present on more detail, including its compliant adaptors. Our attempts at modifying an adaptor were successful, but we were unable to turn it into a full powerline Sniffer - the HomePlug messages are still out of our reach.

5.1 Future work

There are still open possibilities to obtain the HomePlug messages. One is to trace the *athrs_gmac* using the `ftrace` functionalities - `ftrace` is similar to `strace` but traces kernel objects. To use `ftrace` the kernel must be configured to allow such tracing. We would have to replace the kernel of the adaptor with one including `ftrace` capabilities. This could possibly reveal the packets being flowing through the driver.

It is also possible to make calls to *athrs_gmac* using the `syscall` function. Using `syscall` we could call the function *athr_gmac_recv_packets* directly, and possibly read the packets directly from the driver. We expected that no filtering is done when calling the driver directly, thus providing us with the HomePlug messages.

Another test yet to be done is with the DSA driver as a replacement to *athrs_gmac*. Following the steps mentioned in Section 4.6.5, we could possibly substitute *athrs_gmac* with our DSA driver, that does no filtering. This would allow us to completely read the network, and perform the attack.

With the exception of the possibility of using `syscall`, all options require some step that may leave the adaptor unusable. If we had direct connection to the adaptor we could safely perform these tests. Since we do not, we leave these possibilities open for future work.

One last possibility is to disassemble the *athrs_gmac.ko*. Disassembling is a process where a binary file is reverse engineered, resulting in the (approximate) source code that generated the binary [86]. We did this using the Retargetable Decompiler [87], which provided us with a C file with over 19000 lines of code, some containing direct calls to memory addresses. Cross-compiling this file without any changes was unsuccessful, and correcting such code is not trivial. Nevertheless, it is possible to disassemble the driver to C or assembly and understand how the networking of the driver works. Then, we could remove any filtering (assuming it exists) and replace the driver.

Appendix A

UKE details

* Random number field.

‡ Refer to the protocol for more details.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	003132333435
OSA	6	Original Source Address	004647484950
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned Ethertype)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0C60 – CM_GET_KEY.request
FMI	2	Fragmentation Management Information‡	0000
Request Type	1	Request Type	00 – direct
Requested Key type	1	Requested Key Type	04 – Hash Key
NID	7	Network ID	3F5B4FDC4D3D05
My Nonce	4	Random number	$*$
PID	1	Protocol ID	03
PRN	2	Protocol Run Number	AB34
PMN	1	Protocol Message Number	01
HASH KEY	variable	Random number	$*$

Table A.1: UKE message 1.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	004647484950
OSA	6	Original Source Address	003132333435
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned EtherType)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0D60 – CM_GET_KEY.confirm
FMI	2	Fragmentation Management Information	0000
Result	1	Result	00 - key granted
KeyType	1	Key Type	04 - Hash Key
My Nonce	4	Random number.	33221100
Your Nonce	4	Last nonce received.	FFEEDDCC
NID	7	Network ID	3F5B4FDC4D3D05
EKS	1	Encryption key select	03 – Tek type from Hash key
PID	1	Protocol ID	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number	AB34
PMN	1	Protocol Message Number	02
HASH KEY	variable	Random number	$*$

Table A.2: UKE message 2.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	003132333435
OSA	6	Original Source Address	004647484950
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned Ethertype)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0660 – CM_ENCRYPTED_PAYLOAD.indication
FMI	2	Fragmentation Management Information	0000
PEKS	1	Payload Encryption Key Select	03
BSS Status	1	BSS status of source. (Unencrypted)	08 - BM of a BSS
PID	1	Protocol ID (Unencrypted)	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number (Unencrypted)	AB34
PMN	1	Protocol Message Number (Unencrypted)	03
IV	16	AES Initialization Vector (Unencrypted)	$*$
LEN	2	Length of MM, in octets (Unencrypted)	3900 - Length of CM_SET_KEY.request
RF‡	0-15	Random Filler	123456789A
MM‡	variable		
CRC	4	Checksum on MME (Encrypted Payload)	607F75C6
PID	1	Protocol ID (Encrypted Payload)	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number (Encrypted Payload)	AB34
PMN	1	Protocol Message Number (Encrypted Payload)	03
Padding	variable		DBF4C91A3CDA2F169B
RFLen‡	1		05

Table A.3: UKE message 3.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	003132333435
OSA	6	Original Source Address	004647484950
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned Ethertype)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0860 - CM_SET_KEY.request
FMI	2	Fragmentation Management Information	0000
Key Type	1	Key Type	01 - NMK (AES-128)
My Nonce	4	Random number	FFEEDDCC
Your Nonce	4	Last nonce received	33221100
PID	1	Protocol for which Set Key is asserted‡	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number	AB34
PMN	1	Protocol Message Number	03
BM Capability‡	1		02
NID	7	Network ID of transmitting STA	3F5B4FDC4D3D05
NewEKS‡	1	New Encryption Key Select or New Payload	01 - NewEKS is ignored when Key Type is NMK
NewKEY	0, 16 or 384	New Key (128-bit AES)	$*$ (NMK)

Table A.4: UKE message 3 encrypted payload.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	004647484950
OSA	6	Original Source Address	003132333435
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned Ethertype)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0660 - CM_ENCRYPTED_PAYLOAD.indication
FMI	2	Fragmentation Management Information	0000
PEKS	1	Payload Encryption Key Select (Unencrypted)	01 - NMK known to STA (AES 128-bit key)
BSS Status‡	1		05 - Associated with a BSS and PBM Capable
PID	1	Protocol ID (Unencrypted)	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number (Unencrypted)	AB34
PMN	1	Protocol Message Number (Unencrypted)	FF
IV	16	AES Encryption Initialization Vector (Unencrypted)	*
LEN	2	Length of MM, in octets (Unencrypted)	2100
RF	0–15	Random Filler	3456789012
MM	variable		
CRC	4	Checksum on MME (Encrypted Payload)	B1FBF73D
PID	1	Protocol ID (Encrypted Payload)	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number (Encrypted Payload)	AB34
PMN	1	Protocol Message Number (Encrypted Payload)	FF
Padding	variable		34
RFLen	1		05

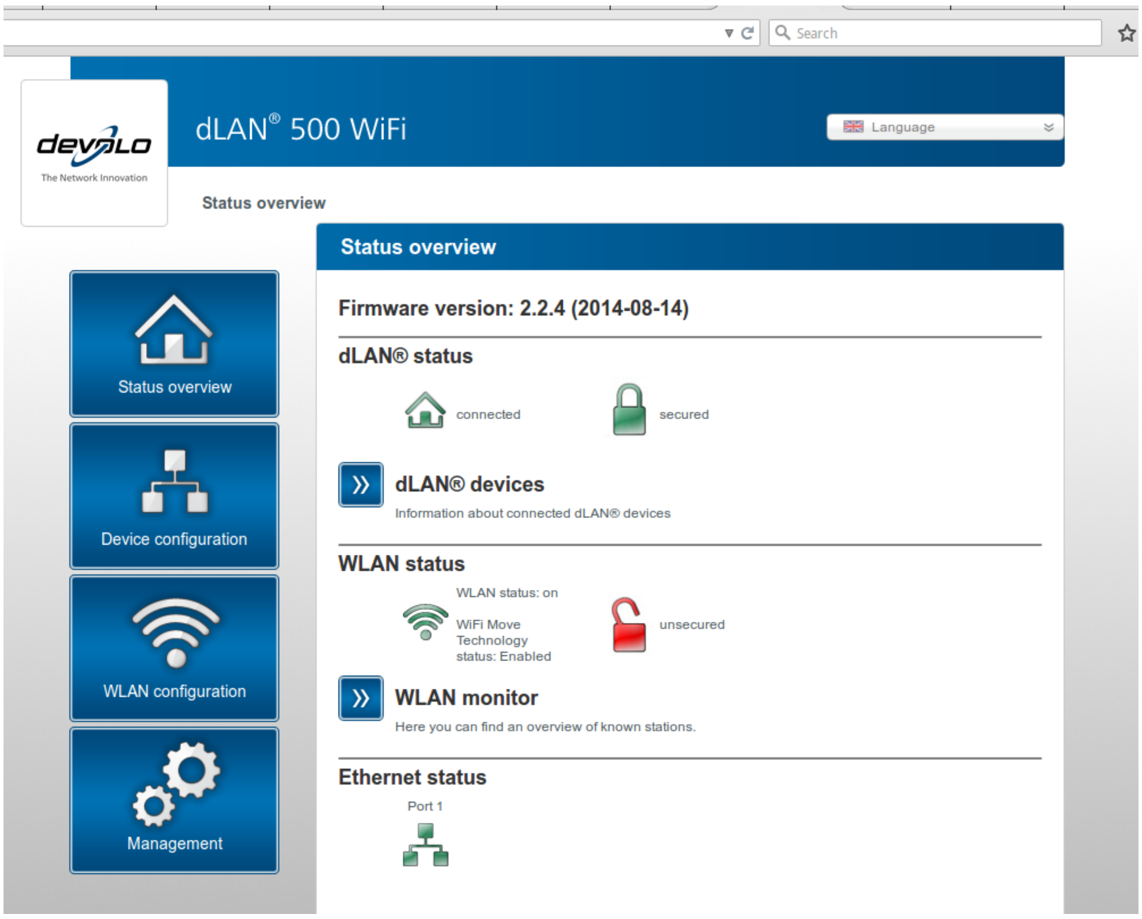
Table A.5: UKE message 4.

Field	Size (octets)	Definition	Example Value
ODA	6	Original Destination Address	004647484950
OSA	6	Original Source Address	003132333435
VLAN Tag	0 or 4	IEEE 802.1Q VLAN Tag (optional)	None
MTYPE	2	0x88E1 (IEEE-assigned Ethertype)	88e1
MMV	1	Management Message Version	01
MMTYPE	2	Management Message Type	0960 - CM_SET_KEY.confirm
FMI	2	Fragmentation Management Information	0000
Result	1	0x00 = success	00
My Nonce	4	Random number	33221100
Your Nonce	4	Last nonce received	FFEEDDCC
PID	1	Protocol for which Set Key is confirmed	03 - Provision STA with NMK using UKE
PRN	2	Protocol Run Number	AB34
PMN	1	Protocol Message Number	FF
BM Capability	1		02

Table A.6: UKE message 4 encrypted payload.

Appendix B

Devolo dLAN 500 Wifi update sequence



Status overview

Firmware version: 2.2.4 (2014-08-14)

dLAN® status

connected

secured

dLAN® devices

Information about connected dLAN® devices

WLAN status

WLAN status: on

WiFi Move Technology status: Enabled

unsecured

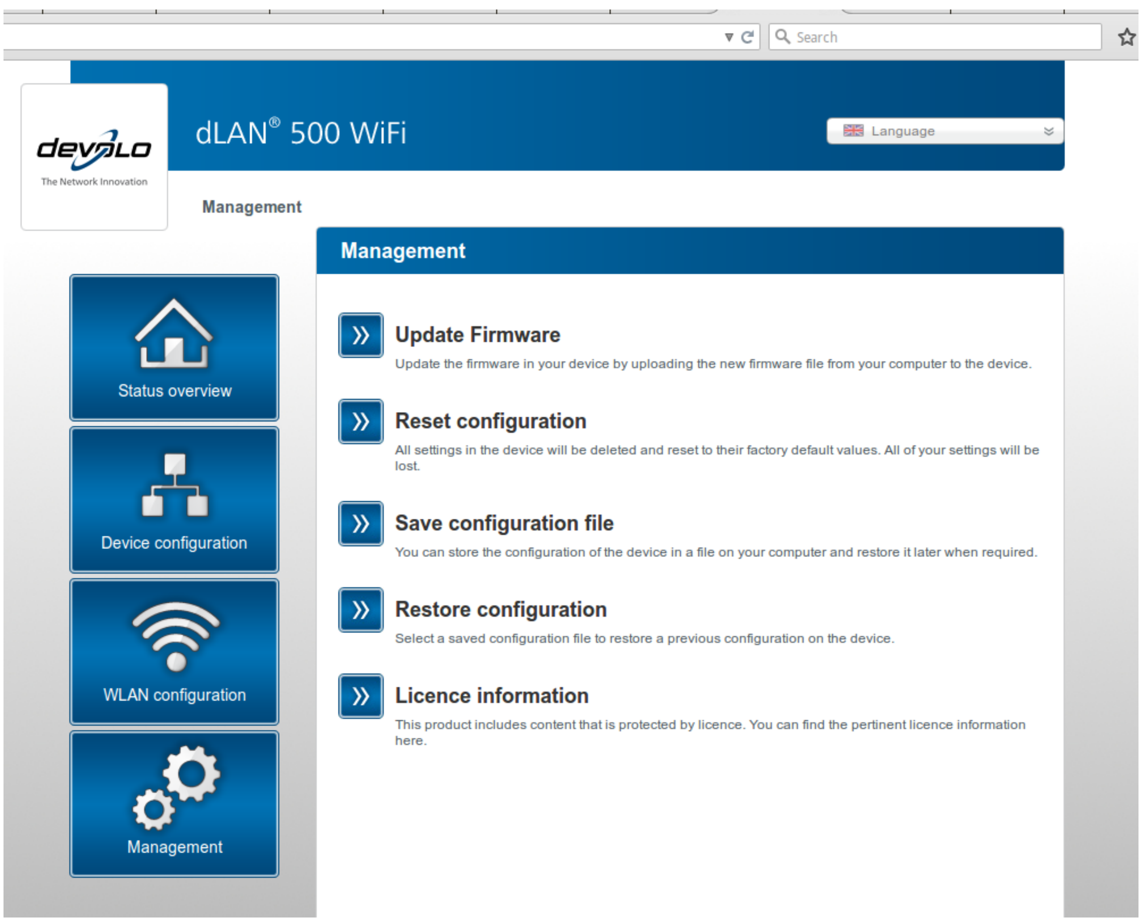
WLAN monitor

Here you can find an overview of known stations.

Ethernet status

Port 1

Figure B.1: Web update sequence (1).



Management

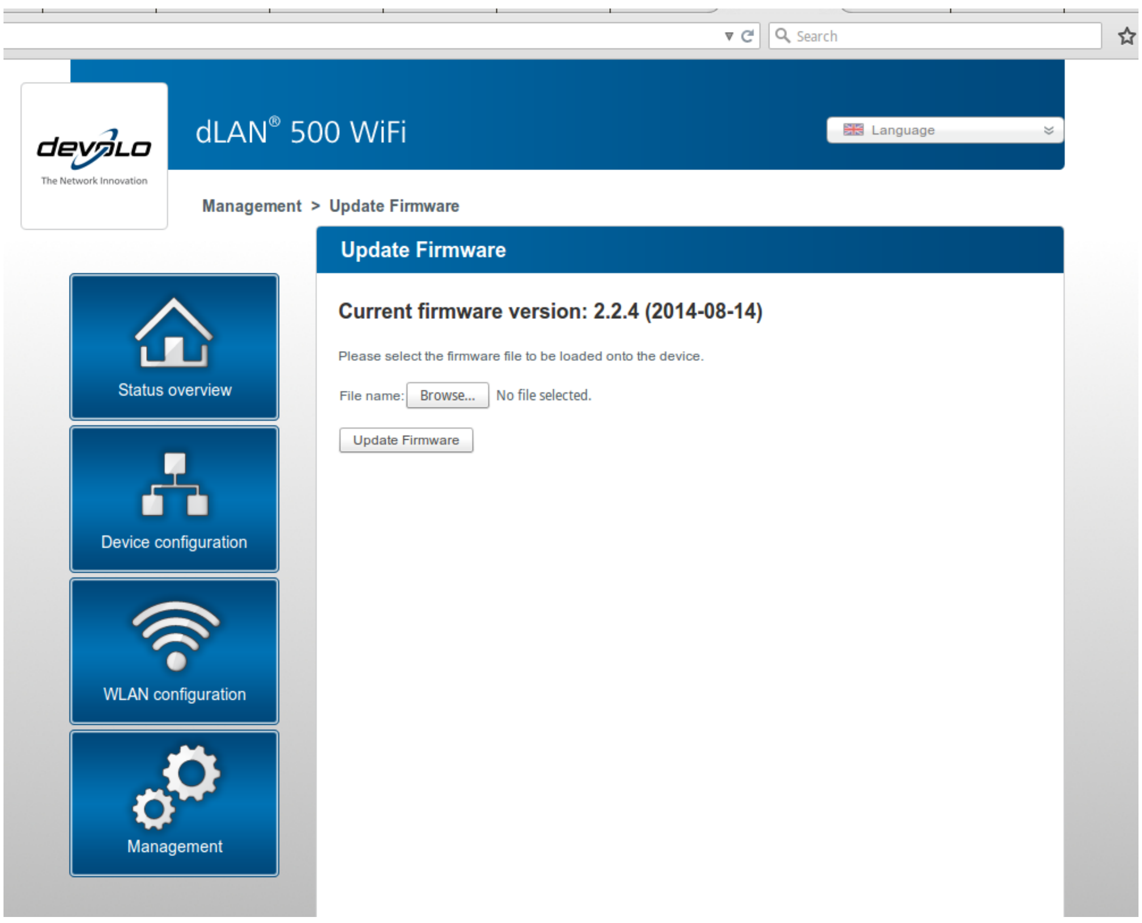
Update Firmware
Update the firmware in your device by uploading the new firmware file from your computer to the device.

Reset configuration
All settings in the device will be deleted and reset to their factory default values. All of your settings will be lost.

Save configuration file
You can store the configuration of the device in a file on your computer and restore it later when required.

Restore configuration
Select a saved configuration file to restore a previous configuration on the device.

Licence information
This product includes content that is protected by licence. You can find the pertinent licence information here.



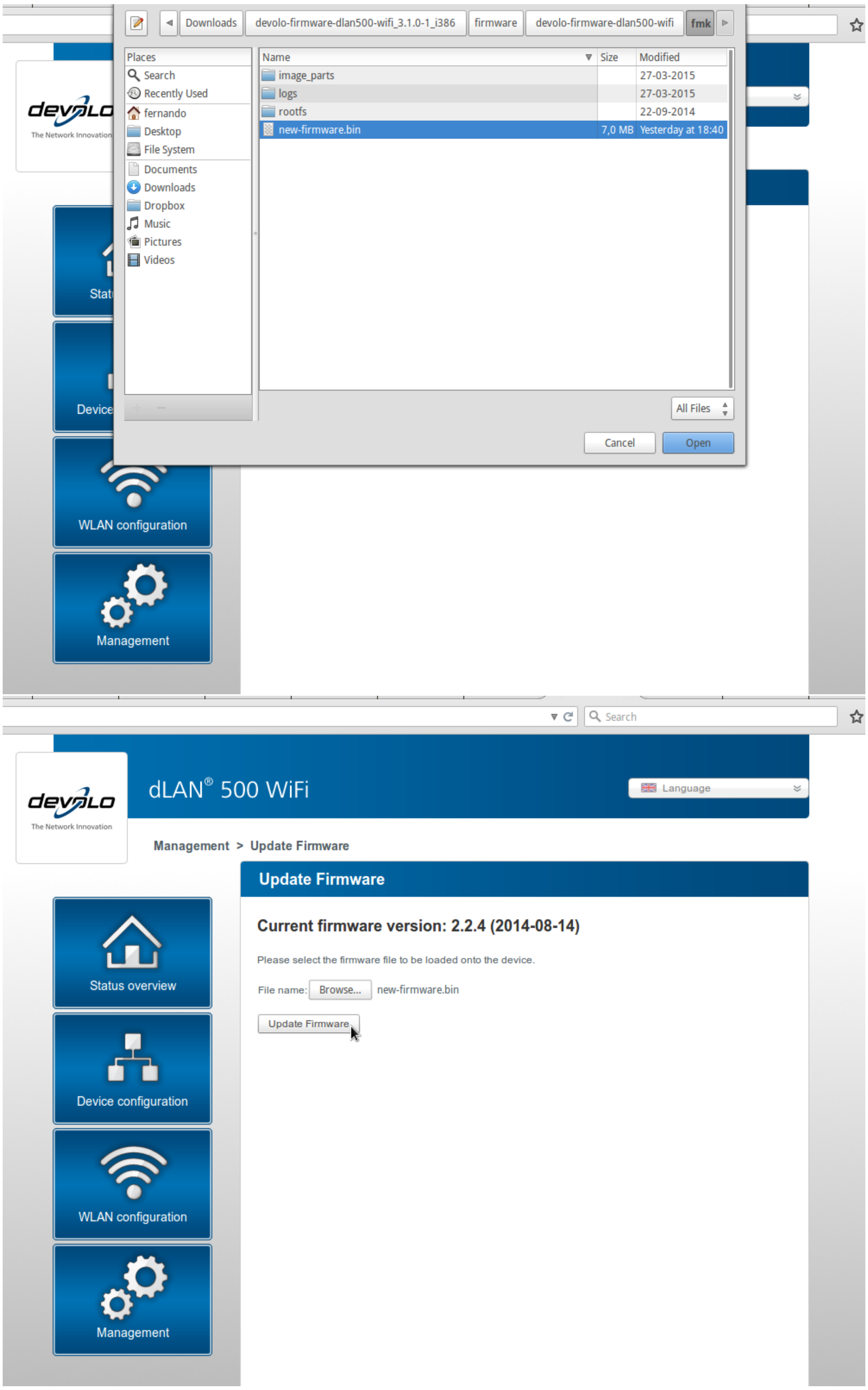
Update Firmware

Current firmware version: 2.2.4 (2014-08-14)

Please select the firmware file to be loaded onto the device.

File name: No file selected.

Figure B.2: Web update sequence (2).



Name	Size	Modified
image_parts		27-03-2015
logs		27-03-2015
rootfs		22-09-2014
new-firmware.bin	7,0 MB	Yesterday at 18:40

dLAN® 500 WiFi

Management > Update Firmware

Update Firmware

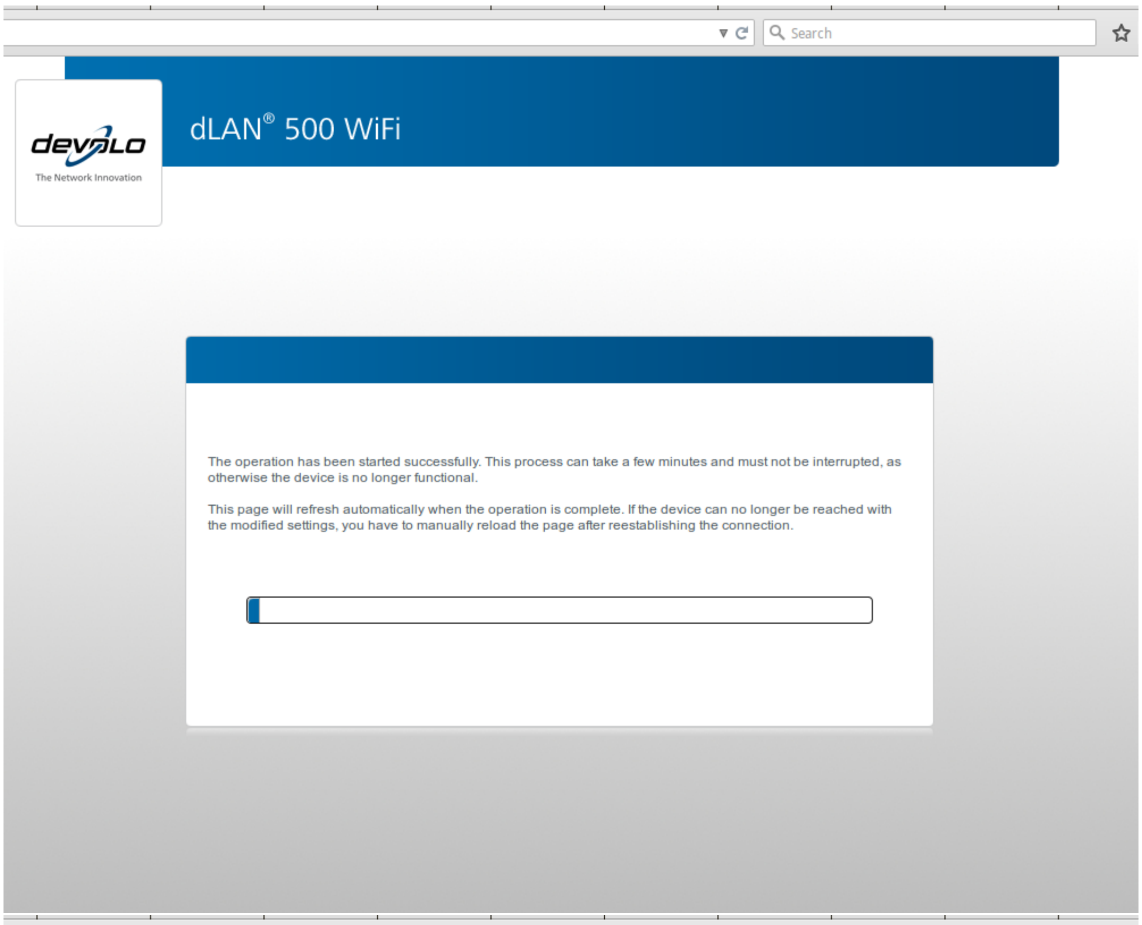
Current firmware version: 2.2.4 (2014-08-14)

Please select the firmware file to be loaded onto the device.

File name: Browse... new-firmware.bin

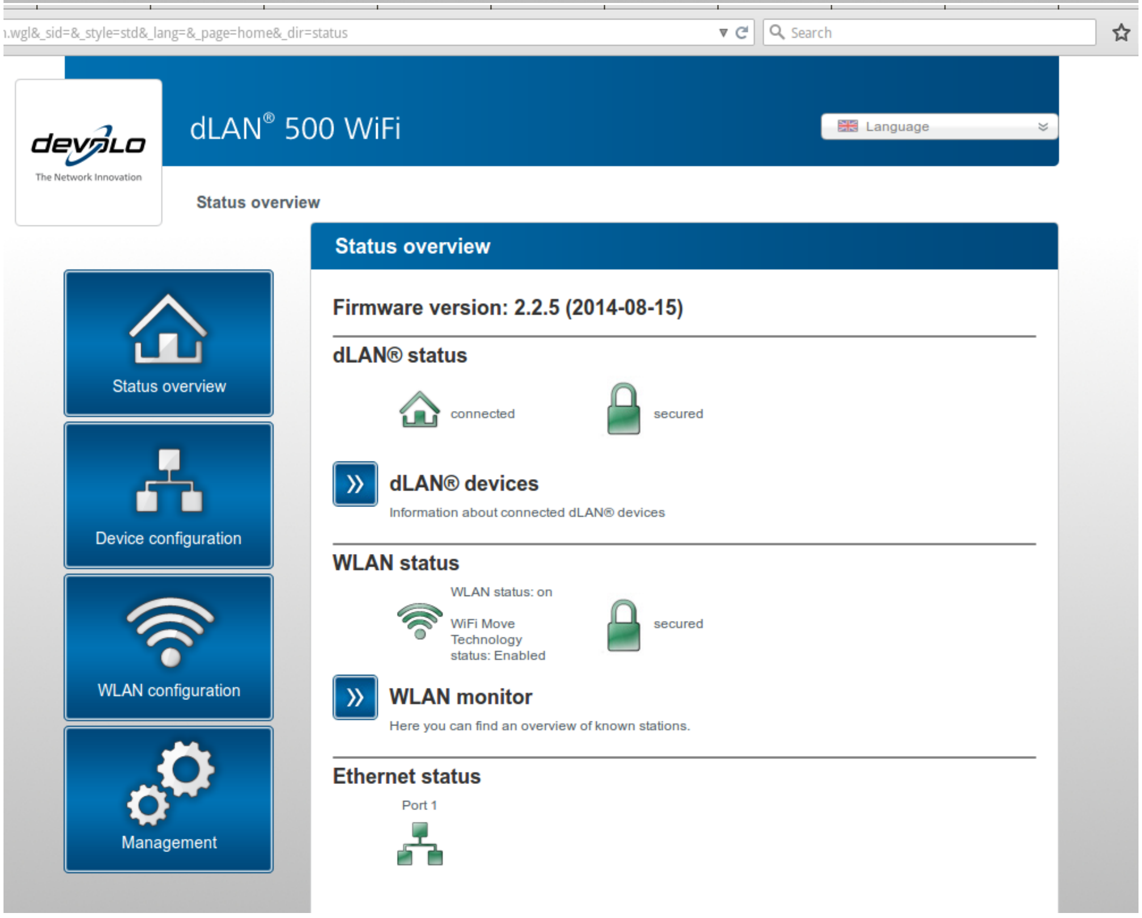
Update Firmware

Figure B.3: Web update sequence (3).



The operation has been started successfully. This process can take a few minutes and must not be interrupted, as otherwise the device is no longer functional.

This page will refresh automatically when the operation is complete. If the device can no longer be reached with the modified settings, you have to manually reload the page after reestablishing the connection.



Status overview

Firmware version: 2.2.5 (2014-08-15)

dLAN® status

connected secured

dLAN® devices

Information about connected dLAN® devices

WLAN status

WLAN status: on

WiFi Move Technology status: Enabled

secured

WLAN monitor

Here you can find an overview of known stations.

Ethernet status

Port 1

Figure B.4: Web update sequence (4).

Appendix C

Makefiles

C.1 Simple native Makefile

```
ifneq ($(KERNELRELEASE),)
  obj-m := driver.o
else
  KERNELDIR ?= /lib/modules/$(shell uname -r)/build)
  PWD := $(shell pwd)
default:
  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

C.2 Simple cross-compiling Makefile

```
ifneq ($(KERNELRELEASE),)
  obj-m := driver.o
else
  KERNELDIR ?= (...)/gpl-source/kernel/2.6.31
  PWD := $(shell pwd)
default:
  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

C.3 Makefile for *dvlbutton*

```
ifneq ($(KERNELRELEASE),)
  obj-m := dvlbutton.o
  dvlbutton-objs := dvlbutton.o gpio.o
else
  KERNELDIR ?= (...)/gpl-source/kernel/2.6.31
  PWD := $(shell pwd)
default:
  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

C.4 Makefile for DSA driver

```
ifneq ($(KERNELRELEASE),)
  obj-m := dsa.o
  dsa-objs := ar7240.o dsa.o mdio_bus.o phy_device.o phy.o
else
  KERNELDIR ?= (...)/gpl-source/kernel/2.6.31
  PWD := $(shell pwd)
default:
  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

Glossary

µClibC A minimalist C library for embedded systems. 34, 42, 43

MIPS Microprocessor without Interlocked Pipeline Stages. 32, 37, 42–44, 47, 55

rcS Script ran as root after the file system is mounted. 37–39, 49–51

`.pib` See PIB. 24, 25, 45

AC Alternate Current. 1, 3

Adaptor A HomePlug compliant electronic device. 1, 2, 5–7, 10, 15, 17–25, 27–56

Bootloader A computer program that loads an operating system. 6, 35, 52, 53, 56

CRC Cyclic Redundancy Check. 33, 37, 38

Cross-compiling A technique for compiling programs across different CPU architectures. 6, 35, 42–44, 46–48, 55, 56

DAK Device Access Key. 15, 22–24, 28, 39

Device An electronic equipment linked to an adaptor. 1, 2, 9, 10, 18–24, 34, 40, 55

Device driver A program that controls a particular type of device that is attached to your computer. 6, 46–54, 56, 57

Devolo Powerline adaptors manufacturer. 6, 25, 27, 32, 33, 35–37, 39, 42, 47, 52, 55

DSA Distributed Switching Architecture. 49–51, 56

Firmware A program for a hardware device. 16, 25, 32–40, 52, 53, 55

FMK Firmware Mod Kit. 36, 37, 52

FourCC Sequence of four bytes used to uniquely identify data formats. 33, 52

HomePlug A protocol for PLC used in home environments. 5–11, 15, 17–25, 27–32, 35, 40–42, 44–46, 49, 53–56

HomePlug management layer Virtual network layer only accessed by HomePlug adaptors used to exchange management messages. 21, 23, 32

HomePlug management message Message used for configuration/control of a HomePlug network. 21, 22, 41, 44, 55

ISP Internet Service Provider. 1

Join confirm (CM_SC_JOIN.confirm) A pre-UKE message indicating that two adaptors can execute the UKE. 29, 42, 45

Join request (CM_SC_JOIN.request) A pre-UKE message indicating an adaptors wants to execute the UKE. 29, 45

Kernel The central module of an operating system. 6, 25, 32–37, 40–42, 46–50, 52–54, 56

Kernel object An external binary compliant with a kernel that can be loaded and unloaded into it during runtime. 6, 35, 46, 47, 49, 56

Key exchange Protocol used for the distribution of cryptographic keys. 4–6, 8, 22, 23, 27, 29–31, 53

NDE Providing NMK by direct entry. 28

Network coordinator Member and manager of a network. 14, 20–22, 24, 25

NID Network Identifier. 19, 22, 31

NMK Network Membership Key. 15, 19, 22, 24, 27–31, 39, 40, 42, 55

NUD Providing NMK using DAK. 28

NVM Firmware for the Atheros's chips (stored in a `.nvm` file). 24, 25

Open-plc A software suit used to manage HomePlug adaptors with Atheros' chips. 24, 25, 45

OSI Open Systems Interconnection. 11, 13, 14

Out-of-band mechanism Secondary channel used for secure information exchange. 4, 22, 27–31

PIB Parameter Information Block (stored in a `.pib` file). 24

PLC Powerline Communication. 1, 3–5, 9–12, 14

Qualcomm Atheros A developer of semiconductors for network communications. 24, 25, 27, 34, 41, 45, 49

Signal modulation Technique used to change properties of a wave. 3, 4, 10, 23, 24

Signal noise Variations to the electric wave caused by electronic devices. 4, 24

Sniffer A device capable of sniffing. 6–8, 41, 55, 56

SquashFS An lzma compressed read-only file system used in embedded systems. 33, 51

To sniff To capture network traffic. 17, 21, 23, 29, 45, 54, 55

To spoof To change the source of a message. 21, 40

UKE Unicast Key Exchange. 27–32, 35, 40–42, 44, 45, 53, 55

Bibliography

[1] S. Galli, A. Scaglione, and Z. Wang. *For the grid and through the grid: The role of power line communications in the smart grid*. In *Proceedings of the IEEE*, volume 99(6):pages 998–1027, 2011.

[2] H. Kopetz. *Internet of Things*. In *Real-Time Systems*, Real-Time Systems Series, pages 307–323. Springer US, 2011. ISBN 978-1-4419-8236-0. doi:10.1007/978-1-4419-8237-7_13. URL `http://dx.doi.org/10.1007/978-1-4419-8237-7_13`.

[3] S. A. Boyer. *SCADA: supervisory control and data acquisition*. International Society of Automation, 2009.

[4] R. van Gerwen, S. Jaarsma, and R. Wilhite. *Smart metering*. In *Leonardo-energy. org*, page 9, 2006.

[5] B. K. Bose. *Power electronics and AC drives*. In *Englewood Cliffs, NJ, Prentice-Hall, 1986, 416 p.*, volume 1, 1986.

[6] T. Saeki. *Orthogonal frequency division multiplexing*, 1999. US Patent 5,956,318.

[7] V. C. Ramasami. *Orthogonal frequency division multiplexing*. In *KUID report*, (698659), 2001.

[8] C. D. Motchenbacher and J. A. Connelly. *Low-noise electronic system design*. Wiley New York, 1993.

[9] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. *Private Memoirs of a Smart Meter*. In *Proceedings of the 2Nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 61–66. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0458-0. doi:10.1145/1878431.1878446. URL `http://doi.acm.org/10.1145/1878431.1878446`.

[10] S. Taherian, M. Pias, G. Coulouris, and J. Crowcroft. *Profiling Energy Use in Households and Office Spaces*. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 21–30. ACM,

New York, NY, USA, 2010. ISBN 978-1-4503-0042-1. doi:10.1145/1791314.1791318. URL `http://doi.acm.org/10.1145/1791314.1791318`.

[11] H. C. Van Tilborg and S. Jajodia. *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2011.

[12] B. C. Neuman and T. Ts'o. *Kerberos: An authentication service for computer networks*. In *Communications Magazine, IEEE*, volume 32(9):pages 33–38, 1994.

[13] W. Diffie and M. E. Hellman. *New directions in cryptography*. In *Information Theory, IEEE Transactions on*, volume 22(6):pages 644–654, 1976.

[14] A. Freier, P. Karlton, and P. Kocher. *The secure sockets layer (SSL) protocol version 3.0*. In , 2011. RFC 6101; republication of original SSL 3.0 specification by Netscape of November 18, 1996.

[15] R. Kainda, I. Flechais, and A. W. Roscoe. *Usability and Security of Out-of-band Channels in Secure Device Pairing Protocols*. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 11:1–11:12. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-736-3. doi:10.1145/1572532.1572547. URL `http://doi.acm.org/10.1145/1572532.1572547`.

[16] S. East, J. Butts, M. Papa, and S. Shenoi. *A Taxonomy of Attacks on the DNP3 Protocol*. In *Critical Infrastructure Protection III*, pages 67–81. Springer, 2009.

[17] R. Newman, L. Yonge, S. Gavette, and R. Anderson. *HomePlug AV security mechanisms*. In *Power Line Communications and Its Applications, 2007. ISPLC'07. IEEE International Symposium on*, pages 366–371. IEEE, 2007.

[18] B. Tasker. *Infiltrating a Network Via Powerline Homeplugav Adapters*. `https://www.bentasker.co.uk/documentation/security/282-infiltrating-a-network-via-powerline-homeplugav-adapters`. Online, accessed: 2014-11-13.

[19] C. Brunschwiler. *Wireless M-Bus Security Whitepaper*. `http://www.csnc.ch/misc/files/2013/wmbus_security_whitepaper.pdf`, 2013. Online, accessed 27/06/2015.

[20] K. Kursawe and C. Peters. *Structural Weaknesses in the Open Smart Grid Protocol*. Cryptology ePrint Archive, Report 2015/088, 2015. `https://eprint.iacr.org/2015/088.pdf` Online, accessed 27/06/2015.

[21] N. Vidgren, K. Haataja, J. L. Patino-Andres, J. J. Ramirez-Sanchis, and P. Toivanen. *Security Threats in ZigBee-Enabled Systems: Vulnerability Evaluation, Practical Experiments, Countermeasures, and Lessons Learned*. In *Proceeding of the*

Hawaii International Conference on System Sciences (HICSS), pages 5132–5138. IEEE, 2013.

[22] T. Goodspeed. *Extracting Keys from Second Generation Zigbee Chips*. `http://www.blackhat.com/presentations/bh-usa-09/GOODSPEED/BHUSA09-Goodspeed-ZigbeeChips-PAPER.pdf`. Online, accessed: 2014-11-18.

[23] E. J. Chikofsky, J. H. Cross *et al.*. *Reverse engineering and design recovery: A taxonomy*. In *Software, IEEE*, volume 7(1):pages 13–17, 1990.

[24] B. Beizer. *Black Box Testing: Techniques for Functional Testing of Software and Systems*. In *Software, IEEE*, volume 13(5):pages 98–, 1996. ISSN 0740-7459. doi: 10.1109/MS.1996.536464.

[25] *HomePlug specifications*. `http://www.cise.ufl.edu/~nemo/plc/refs/`. Online, accessed: 2014-11-18.

[26] I. L. Taylor. *The GNU configure and build system*. In *Website: http://www.airs.com/ian/configure*, 1998.

[27] F. Yellin, D. R. Long, and R. D. Tuck. *Apparatus and method for cross-compiling source code*, 1999. US Patent 5,946,489.

[28] P. Alliance. *PRIME 1.3.6 specification*. `http://www.prime-alliance.org/wp-content/uploads/2013/04/PRIME-Spec_v1.3.6.pdf`. Online, accessed: 2014-11-18.

[29] P. Alliance. *PRIME 1.4 specification*. `http://www.prime-alliance.org/wp-content/uploads/2014/10/PRIME-Spec_v1.4-20141031.pdf`. Online, accessed: 2014-11-18.

[30] BioBankCloud. *Plataform as a service for Biobanking*. `http://www.biobankcloud.com/`. Online, accessed 28/06/2015.

[31] F. Alves, V. Cogo, S. Wandelt, U. Leser, and A. Bessani. *On-demand indexing for referential compression of dna sequences*. In *PLoS ONE*, volume 10(7):page e0132 460, 2015. doi:10.1371/journal.pone.0132460. URL `http://dx.doi.org/10.1371%2Fjournal.pone.0132460`.

[32] *IEEE Standard for Low-Frequency (less than 500 kHz) Narrowband Power Line Communications for Smart Grid Applications*. In *IEEE Std 1901.2-2013*, pages 1–269, 2013. doi:10.1109/IEEESTD.2013.6679210.

[33] *IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications*. In *IEEE Std 1901-2010*, pages 1–1586, 2010. doi:10.1109/IEEESTD.2010.5678772.

[34] H.-P. Alliance. *HD-PLC Complete technical over view for full spec design for AV and PC Equipment*. `http://www.hd-plc.org/modules/about/hdplc.html`. Online, accessed: 2014-11-18.

[35] *IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)*. In *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pages 1–821, 2012. doi:10.1109/IEEESTD.2012.6327578.

[36] G.-P. Alliance. *G3-PLC Guidelines - Low Layers*. `www.erdf.fr/medias/Linky/G3_Specifications_%20low_%20layers.pdf`. Online, accessed: 2014-11-18.

[37] G.-P. Alliance. *G3-PLC Guidelines - Metering Profile*. `www.erdf.fr/medias/Linky/G3_%20Specifications_%20metering-%20profile.pdf`. Online, accessed: 2014-11-18.

[38] *M-BUS specifications*. `http://www.m-bus.com/files/default.php`. Online, accessed: 2014-11-18.

[39] O. Meter. *Open Public Extended Network Metering*. `http://www.openmeter.com/`. Online, accessed 28/06/2015.

[40] O. Alliance. *OSGP specifications*. `http://www.etsi.org/deliver/etsi_gs/osg/001_099/001/01.01.01_60/gs_osg001v010101p.pdf`. Online, accessed: 2014-11-18.

[41] Z. Alliance. *Zigbee specifications*. `http://zigbee.org/zigbee-for-developers/applicationstandards/`. Online, accessed: 2014-11-18.

[42] R. Richards. *Representational State Transfer (REST)*. In *Pro PHP XML and Web Services*, pages 633–672. Apress, 2006. ISBN 978-1-4302-1361-1. doi:10.1007/978-1-4302-0139-7_17. URL `http://dx.doi.org/10.1007/978-1-4302-0139-7_17`.

[43] BlueKrypt. *Crypographic Key Length Recommendation*. `www.keylength.com`. Online, accessed 28/06/2015.

[44] C. Paar and J. Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[45] G. Paul and S. Maitra. *RC4 stream cipher and its variants*. CRC press, 2011.

[46] S. Fluhrer, I. Mantin, and A. Shamir. *Weaknesses in the Key Scheduling Algorithm of RC4*. In S. Vaudenay and A. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-43066-7. doi:10.1007/3-540-45537-X_1. URL `http://dx.doi.org/10.1007/3-540-45537-X_1`.

[47] I. Mantin and A. Shamir. *A Practical Attack on Broadcast RC4*. In M. Matsui, editor, *Fast Software Encryption*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43869-4. doi: 10.1007/3-540-45473-X_13. URL `http://dx.doi.org/10.1007/3-540-45473-X_13`.

[48] J. Golić. *Linear Statistical Weakness of Alleged RC4 Keystream Generator*. In W. Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-62975-7. doi:10.1007/3-540-69053-0_16. URL `http://dx.doi.org/10.1007/3-540-69053-0_16`.

[49] A. Klein. *Attacks on the RC4 stream cipher*. In *Designs, Codes and Cryptography*, volume 48(3):pages 269–286, 2008. ISSN 0925-1022. doi:10.1007/s10623-008-9206-6. URL `http://dx.doi.org/10.1007/s10623-008-9206-6`.

[50] F. P. Miller, A. F. Vandome, and J. McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009. ISBN 6130268297, 9786130268299.

[51] D. Kennedy. *Hacking over power lines*. `http://dangerousprototypes.com/2011/11/04/defcon-19-hacking-over-power-lines/`. Online, accessed 11/06/2015.

[52] Q. A. Inc. `http://www.qca.qualcomm.com/`. Online, accessed 26/06/2015.

[53] Q. Atheros. *Qualcomm Atheros' Open Powerline Toolkit*. `https://github.com/qca/open-plc-utils/`. Online, accessed 08/05/2015.

[54] F. Fainelli. *faifa*. `https://github.com/ffainelli/{faifa}/`. Online, accessed 08/05/2015.

[55] Devolo. *Devolo Cockpit - Software for our home network*. `http://www.devolo.com/en/dLAN-software/devolo-Cockpit-simple-powerline-network-configuration/`. Online, accessed 08/05/2015.

[56] A. Orebaugh, G. Ramirez, and J. Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[57] G. Parsons. *Ethernet bridging architecture [standards topics].* In *Communications Magazine, IEEE*, volume 45(12):pages 112–119, 2007.

[58] J. Postel and J. K. Reynolds. *Telnet protocol specification.* In , 1983. RFC 854.

[59] D. J. Barrett and R. E. Silverman. *SSH, the Secure Shell: the definitive guide.* " O'Reilly Media, Inc.", 2001.

[60] Devolo. *Devolo dLAN 500 WiFi.* `http://www.devolo.com/en/Products/dLAN-500-WiFi/`. Online, accessed 02/06/2015.

[61] Devolo. *Devolo update ftp.* `http://update.devolo.com/linux2/apt/pool/main/d/devolo-firmware-dlan500-wifi/`. Online, accessed 08/05/2015.

[62] D. S. Engineering. *Das U-Boot – the Universal Boot Loader .* `http://www.denx.de/wiki/U-Boot`. Online, accessed 09/06/2015.

[63] A. S. Foundation. *Apache^{TM} Subversion®* - *"Enterprise-class centralized version control for the masses".* `https://subversion.apache.org/`. Online, accessed 21/05/2015.

[64] E. Andersen. *A C library for embedded Linux.* `http://www.uclibc.org/`. Online, accessed 08/05/2015.

[65] A. Laboratories. *Acme laboratories - graphics * unix * networks * fun.* `http://acme.com/software/thttpd/`. Online, accessed 22/05/2015.

[66] C. Heffner and J. Collake. *Firmware Mod Kit - Modify firmware images without recompiling!.* `https://code.google.com/p/firmware-mod-kit/`. Online, accessed 08/05/2015.

[67] M. F. Krafft. *The Debian system: concepts and techniques.* No Starch Press, 2005.

[68] J. S. Sobolewski. *Cyclic Redundancy Check.* In *Encyclopedia of Computer Science*, pages 476–479. John Wiley and Sons Ltd., Chichester, UK. ISBN 0-470-86412-5. URL `http://dl.acm.org/citation.cfm?id=1074100.1074303`.

[69] J. Mocnik and C. Celorio. *GHex - a hex editor for GNOME.* `https://wiki.gnome.org/Apps/Ghex/`. Online, accessed 22/05/2015.

[70] H.-R. SA. *IDA disassembler and debugger.* `https://www.hex-rays.com/products/ida/`. Online, accessed 02/06/2015.

[71] G. N. Purdy. *Linux iptables pocket reference.* "O'Reilly Media, Inc.", 2004.

[72] A. K. Sood and R. J. Enbody. *Targeted Cyberattacks: A Superset of Advanced Persistent Threats*. In *IEEE Security & Privacy*, volume 11(1):pages 54–61, 2013. ISSN 1540-7993. doi:http://doi.ieeecomputersociety.org/10.1109/MSP.2012.90.

[73] E. Brewer. *CAP twelve years later: How the "rules" have changed*. In *Computer*, volume 45(2):pages 23–29, 2012. ISSN 0018-9162. doi:10.1109/MC.2012.37.

[74] *tcpdump*. `http://www.tcpdump.org/`. Online, accessed 02/05/2015.

[75] P. Korsgaard. *Buildroot - Making Embedded Linux Easy*. `http://buildroot.uclibc.org/`. Online, accessed 24/05/2015.

[76] D. W. Barron. *Linkers and Loaders*. In *Encyclopedia of Computer Science*, pages 988–991. John Wiley and Sons Ltd., Chichester, UK. ISBN 0-470-86412-5. URL `http://dl.acm.org/citation.cfm?id=1074100.1074541`.

[77] F. Bellard. *QEMU - open source process emulator*. `http://wiki.qemu.org/Main_Page`. Online, accessed 13/05/2015.

[78] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux device drivers*. "O'Reilly Media, Inc.", 2005.

[79] M. Stallman, R. McGrath, and P. D. Smith. *GNU Make*. Free Software Foundation, Boston, 1991.

[80] Devolo. *Devolo software license information*. `http://www.devolo.com/downloads/data/gpl-source-dlan-500-wifi.tar.bz2`. Online, accessed 21/05/2015.

[81] *Kernel logging system*. `http://linux.die.net/man/2/syslog/`. Online, accessed 04/06/2015.

[82] OpenWrt. `https://dev.openwrt.org/browser/trunk/target/linux/ar71xx/files/net/dsa/ar7240.c?rev=19927`. Online, accessed 28/06/2015.

[83] E. Sneed, G. D. Huensch, J. J. Kulzer, and H. Moerkerken. *Distributed switching architecture trends and concepts*. In *AT T Technical Journal*, volume 73(6):pages 19–27, 1994. ISSN 8756-2324. doi:10.1002/j.1538-7305.1994.tb00616.x.

[84] S. She and T. Berger. *Formal semantics of the Kconfig language*. In *Technical note, University of Waterloo*, page 24, 2010.

[85] C. Dietrich, R. Tartler, W. Schröder-Preikshat, and D. Lohmann. *Understanding Linux Feature Distribution*. In *Proceedings of the 2012 Workshop on Modularity in Systems Software*, MISS '12, pages 15–20. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1217-2. doi:10.1145/2162024.2162030. URL `http://doi.acm.org/10.1145/2162024.2162030`.

[86] B. Schwarz, S. Debray, and G. Andrews. *Disassembly of executable code revisited*. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 45–54. 2002. ISSN 1095-1350. doi:10.1109/WCRE.2002.1173063.

[87] A. Technologies. `https://retdec.com/`. Online, accessed 28/06/2015.